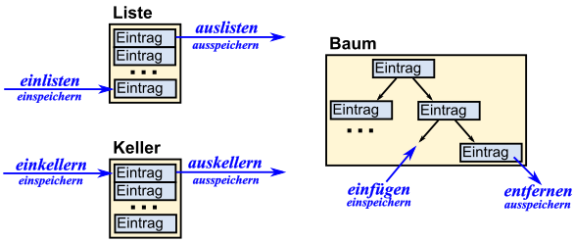
Informatik

für die Sekundarstufe I + II

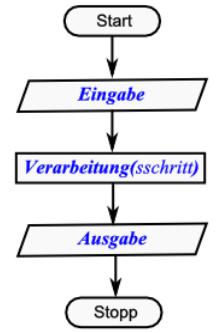
- Programmieren mit Python – Teil 2: für Fortgeschrittene

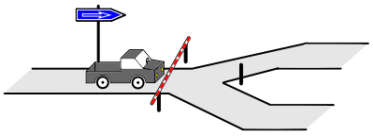
Autor: L. Drews



Grüner Baum-Python
(s) Morelia viridis
Q: de.wikipedia.org (Mwx)

```
while eval(input("?:")) != 0:
    print("Stoppen",end='')
```

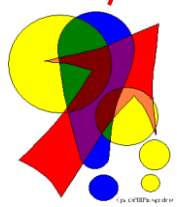




teilredigierte Version 0.10c (2022)

(c,p) 2015 - 2022 lern-soft-projekt: drews

Legende:
mit diesem Symbol werden zusätzliche Hinweise, Tips und weiterführende Ideen gekennzeichnet



Nutzungsbestimmungen / Bemerkungen zur Verwendung durch Dritte:

(1) Dieses Skript (Werk) ist zur freien Nutzung in der angebotenen Form durch den Anbieter (lern-soft-projekt) bereitgestellt. Es kann unter Angabe der Quelle und / oder des Verfassers gedruckt, vervielfältigt oder in elektronischer Form veröffentlicht werden.

(2) Das Weglassen von Abschnitten oder Teilen (z.B. Aufgaben und Lösungen) in Teildrucken ist möglich und sinnvoll (Konzentration auf die eigenen Unterrichtsziele, -inhalte und -methoden). Bei angemessen großen Auszügen gehört das vollständige Inhaltsverzeichnis und die Angabe einer Bezugsquelle für das Originalwerk zum Pflichtteil.

(3) Ein Verkauf in jedweder Form ist ausgeschlossen. Der Aufwand für Kopierleistungen, Datenträger oder den (einfachen) Download usw. ist davon unberührt.

(4) Änderungswünsche werden gerne entgegen genommen. Ergänzungen, Arbeitsblätter, Aufgaben und Lösungen mit eigener Autorenschaft sind möglich und werden bei konzeptioneller Passung eingearbeitet. Die Teile sind entsprechend der Autorenschaft zu kennzeichnen. Jedes Teil behält die Urheberrechte seiner Autorenschaft bei.

(5) Zusammenstellungen, die von diesem Skript - über Zitate hinausgehende - Bestandteile enthalten, müssen verpflichtend wieder gleichwertigen Nutzungsbestimmungen unterliegen.

(6) Diese Nutzungsbestimmungen gehören zu diesem Werk.

(7) Der Autor behält sich das Recht vor, diese Bestimmungen zu ändern.

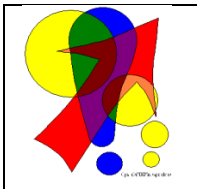
(8) Andere Urheberrechte bleiben von diesen Bestimmungen unberührt.

Rechte Anderer:

Viele der verwendeten Bilder unterliegen verschiedensten freien Lizenzen. Nach meinen Recherchen sollten alle genutzten Bilder zu einer der nachfolgenden freien Lizenzen gehören. Unabhängig von den Vorgaben der einzelnen Lizenzen sind zu jedem extern entstandenen Objekt die Quelle, und wenn bekannt, der Autor / Rechteinhaber angegeben.

public domain (pd)	Zum Gemeingut erklärte Graphiken oder Fotos (u.a.). Viele der verwendeten Bilder entstammen Webseiten / Quellen US-amerikanischer Einrichtungen, die im Regierungsauftrag mit öffentlichen Mitteln finanziert wurden und darüber rechtlich (USA) zum Gemeingut wurden. Andere kreative Leistungen wurden ohne Einschränkungen von den Urhebern freigegeben.
gnu free document licence (GFDL; gnu fdl)	
creative commens (cc) creative commons	BY: od. neu … Namensnennung … nichtkommerziell … in der gleichen Form … unter gleichen Bedingungen

Die meisten verwendeten Lizenzen schließen eine kommerzielle (Weiter-)Nutzung aus!



Bemerkungen zur Rechtschreibung:

Dieses Skript folgt nicht zwangsläufig der neuen **ODER** alten deutschen Rechtschreibung. Vielmehr wird vom Recht auf künstlerische Freiheit, der Freiheit der Sprache und von der Autokorrektur des Textverarbeitungsprogramms microsoft ® WORD ® Gebrauch gemacht.

Für Hinweise auf echte Fehler ist der Autor immer dankbar.

Inhaltsverzeichnis:

Seite

7. Problem-Lösen mit Python 9

7.0. Aufgaben versus Probleme 9

7.0.1. Programm-Entwicklungs-Strategien 11

7.0.2. Strategien zur Lösung von (echten) Problemen 14

kleine Programm-Beispiele 20

7.0.**3,14** Python am Pi-Day 21

Pi-Berechnung durch Monte Carlo Simulation 21

Pi-Berechnung über Verhältnis der Flächen von äußeren und inneren Vieleck 21

(einfache) besondere Zahlen 22

über Teilersummen definierte besondere Zahlen 22

sonstige (ganz) besondere Zahlen 23

seltene oder ungewöhnliche Zahlen und Zahlensysteme (in der Schule) 23

8. Python für Fortgeschrittene 24

8.1. Strings – Zeichenketten 24

8.1.1. einzelne Symbole / Zeichen / Charaktere 24

8.1.2. Sequenzen von Zeichen - Zeichenketten / Strings 25

8.1.1. Objekt-orientierte Nutzung von Strings 27

8.1.2. besondere Möglichkeiten für Strings in Python 28

8.2. Datentypen und Typumwandlungen 29

8.2.1. Zahlen 31

8.2.1.1. ganze Zahlen 31

Zahlendarstellung über spezielle Literale 31

8.2.1.2. Fließkommazahlen / Gleitkommzahlen 31

8.2.1.3. Wahrheitswerte 33

8.2.2. Strings und Co als Datentypen 34

8.2.2.1. einzelne Zeichen 34

8.2.2.2. Sequenzen von Zeichen - Zeichenketten 34

8.2.3. Listen, die I. – einfache Listen 36

8.2.3.0. theortische Vorbetrachtungen 37

8.2.3.0. 1. Listen – eine Form der Datensammlung 37

8.2.3.0.2. Daten-Struktur: Liste 39

8.2.3.1. Definition und Zuweisung von Listen in Python 44

8.2.3.2. Listen-Operationen (Built-in-Operatoren) 46

8.2.3.3. Listen-Indexierung 48

8.2.3.4. Listen-Bearbeitung 50

8.2.3.5. Listen-Abschnitte (Slicing) 54

8.2.3.6. Listen-Erzeugung – fast automatisch 55

8.2.3.7. Listen - extravagant 56

erweitertes Listen-Generieren 56

erweitertes Slicing 57

8.2.3.8. Ringe – geschlossene Listen 59

8.2.3. Dictionarys - Wörterbücher 61

8.4. Interation oder Rekursion? – das ist hier die Frage! 64

8.4.1. Interation 65

8.4.1.1. typische Interations-Anwendungen 66

8.4.1.1.1. Summen-Bildung 66

8.4.1.1.2. Produkt-Bildung 68

8.4.2. Rekursion 69

8.4.2.1. Rekursions-Beispiele: Summen- und Produkt-Bildung 71

8.4.2.2. weitere typische Anwendungen für Rekursionen 73

8.4.2.2.1. Überführung einer Dezimal-Zahl in eine Dual-Zahl 73

8.4.2.2.2. die Fakultät 73

8.4.2.2.3. die FIBONACCHI-Folge 74

8.4.2.2.4. das ggT – der Größte gemeinsame Teiler 75

8.4.2.2.5. Erkennung von Palindromen ... 77
8.4.2.2.x. weitere klassische Rekursions-Probleme ... 78
8.4.2.2. direkte Gegenüberstellung von interativen und rekursiven Algorithmen ... 87
8.4.2.2.1. GGT – größter gemeinsamer Teiler ... 87
8.4.2.2.2. Palindrom-Prüfung ... 88
8.4.2.2.2. Potenz-Prüfung ... 88
8.4.3. komplexe Programmier-Aufgaben: ... 89
8.5. Umgang mit Dateien ... 91
8.5.0. Dateien und Ordner ... 91
8.5.1. Dateien lesen ... 92
8.5.1.1. Lesen von Text-Dateien ... 92
8.5.1.1.1. Lesen von CSV- bzw. strukturierten TXT-Dateien ... 92
8.3.1.1.2. Lesen von XML-Dateien ... 93
8.1.1.1.3. Lesen von JSON-Dateien ... 93
8.5.1.2. Lesen von Binär-Dateien ... 93
8.5.2. Dateien schreiben ... 94
8.5.2.1. Schreiben von Text-Dateien ... 94
8.5.2.1.1. Schreiben einer neuen Datei ... 94
8.5.2.1.2. anhängendes Schreiben ... 94
8.5.2.1.3. Schreiben von CSV- bzw. strukturierten TXT-Dateien ... 94
8.5.2.1.4. Schreiben von XML-Dateien ... 95
8.5.2.1.5. Schreiben von JSON-Dateien ... 95
8.5.2.2. Schreiben von Binär-Dateien ... 95
8.5.3. gepickelte Dateien – Dateien mit gemischten Daten ... 95
8.5.3.1. Schreiben von Dateien mit gemischten Daten ... 95
8.5.3.2. Lesen von Dateien mit gemischten Daten ... 95
8.6. Module ... 96
8.6.1. "built-in"-Funktionen ... 98
8.6.2. wichtige interne Module ... 99
8.6.2.1. die Bibliothek math ... 99
ausgewählte Konstanten ... 99
ausgewählte Funktionen ... 99
8.6.2.2. die Bibliothek random ... 101
8.6.2.x. Verschiedenes zum Modul: sys ... 101
8.6.2.x. Verschiedenes zum Modul: time ... 102
8.6.2.x. Verschiedenes zum Modul:datetime ... 104
8.6.2.x. Verschiedenes zum Modul: os ... 106
8.6.3. externe Module installieren und nutzen ... 109
8.6.3.x. Package-Installer PIP ... 109
8.6.4. Modul / Bibliothek NumPy ... 110
Importieren der Bibliothek ... 110
Erstellen von Array's ... 110
Initialisieren eines leeren Array's ... 111
Initialisieren eines Array's mit Nullen (Null-Matrix) ... 111
Initialisieren eines Array's mit Nullen (Null-Matrix) ... 111
Initialisieren eines Array's mit Zufalls-Zahlen ... 111
Daten aus Dateien einlesen ... 111
Zugriff auf Daten-Elemente ... 112
Operationen / Funktionen mit / zu Array's ... 112
Lineare Algebra (z.B. Lösen von Gleichungs-Systemen) ... 113
8.6.5. Modul / Bibliothek MatPlotLib ... 114
8.6.5.1. allgemeines Vorgehen (Workflow) ... 116
8.6.5.2. Erstellen und Manipulieren von Diagrammen ... 118
8.6.5.2.1. Entscheidung für einen Diagramm-Typ ... 118
8.6.5.2.2. Sichern der Diagramme ... 122
8.6.5.2.3. Diagramm gestalten / formatieren ... 123
8.6.5.2.4. weitere Diagramm-Typen ... 131
8.6.5.3. ein komplexes Diagramm-Projekt – Erdbeben-Anzeige ... 135
8.6.6. Modul / Bibliothek network ... 136

8.6.7. Modul / Bibliothek re ... 137
8.6.8. Modul / Bibliothek pymongo ... 138
8.6.9. Modul / Bibliothek ?? (Word Embedding) ... 139
8.6.99. Cheat Sheet's für einige Bibliotheken ... 140
zu NumPy ... 140
zu Matplotlib ... 140
zu SciPy (lineare Algebra) ... 140
zu Pandas ... 140
weitere Cheat Sheet's ... 141
8.7. Graphik ... 142
8.8. Turtle-Graphik – ein Bild sagt mehr als tausend Worte ... 143
8.8.1. Turtle auf der Shell ... 143
8.8.2. Turtle-Programme und Sequenzen ... 146
8.8.3. Schleifen ... 148
8.8.4. Verzweigungen ... 150
8.8.5. Funktionen ... 152
8.8.6. Rekursion ... 155
8.8.7. Eingaben mit der Maus ... 156
8.8.8. Und wie geht es weiter? ... 157
Windrad aus Rechtecken ... 157
Parkettierung (mit Rhomben) ... 157
Zeichnen eines Strauches ... 158
Baum mit Früchten ... 159
Python-Stern ... 159
8.8.9. Turteln bis zu Umfallen - rekursive Probleme schrittweise Lösen ... 160
8.8.10. Verändern des Schildkröten-Zeigers ... 173
8.8.11. Animationen mittels turtle-Grafik ... 173
8.8.12. Realisierung des Snake-Spiel's mittels turtle-Grafik ... 176
8.9. Musik mit python ... 184
8.9.1. Musik mit Board-Mitteln ... 184
8.9.2. Musik mit python-sonic ... 184
8.10. das Modul "pygame" ... 185
8.10.0. Quellen und Installation ... 185
8.10.1. Ausprobieren / Testen / Grundlagen ... 186
8.10.1. Sound mit pygame ... 188
8.10.1.1. Sound-Dateien abspielen ... 188
8.10.1.2. Sound-Dateien erzeugen / aufnehmen ... 189
8.10.1.3. Musik aus dem Synthesizer ... 189
8.10.2. Grafik mit pygame ... 189
8.11. Objekt-orientierte Programmierung ... 191
Design pattern – Entwurfsmuster ... 196
8.11.x. Objekt-orientierte Programmierung mittels Turtle-Grafik ... 197
8.11.x. Klassen – selbst erstellen ... 200
Klasse-Objekt-Beziehung ... 203
"ist"-Beziehung (Vererbung) ... 203
"besteht_aus"-Beziehung (Aggregation) ... 203
"hat"-Beziehung (Komposition) ... 204
"kennt"-Beziehung ... 204
Übersicht / Legende zu UML-(Klassen-)Diagrammen: ... 206
8.11.x.1. Erstellen einer Klasse ... 207
8.11.x.1.1. der Konstruktor ... 208
8.11.x.2. Attribute einer Klasse ... 209
8.11.x.3. Methoden einer Klasse ... 210
8.11.x.4. Speicher-Bereinigung ... 212
8.11.x.4.1. der Destruktor ... 212
8.11.x.6. eine "Auto"-Klasse ... 220
8.11.x.6.1. Erweiterung der "Auto"-Klasse um LKW's ... 220

8.11.x.7. eine "Personen"-Klasse ... 221
8.11.x.7.1. Erweiterung der "Personen"-Klasse auf eine Familie ... 222
8.11.x.7. eine "Nachrichten"-Klasse ... 223
8.11.x.y. eine Graphik-Beispiel-Klasse ... 224
8.11.x.2. Polymophismus und Vererbung ... 229
8.11.x.y. Tips und Tricks zu Objekt-orientierten Programmen / Klassen-Definitionen ... 233
8.11.x. OOP-Programmbeispiele ... 234
8.12. GUI-Programme mit Tkinter ... 237
8.12.1. … und der erste Programmierer sprach: "Hallo Welt!" ... 239
8.12.2. Nutzung verschiedener Bedienelemente ... 240
8.12.2.1. Button's - Schaltflächen ... 241
8.12.2.1.1. eine eigene Button-Aktion erstellen ... 242
8.12.2.1.2. Button gestalten / formatieren ... 244
8.12.2.2. Nachrichten-Felder / Text-Felder ... 245
8.12.2.3. Eingabe-Felder / Eingabezeilen ... 246
8.12.2.4. Nachrichten-Boxen ... 248
8.12.2.5. Checkbutton-Wdget's – Options-Felder ... 250
8.12.2.6. Radiobutton-Widget – Options-Auswahl ... 251
8.12.2.7. Text-Fenster / Text-Widget ... 254
8.12.2.8. Frames – Group-Box's – Gruppen-Boxen ... 256
8.12.2.9. Menüs / Menu-Widget ... 257
8.12.2.9.2. eine Tool-Bar einbauen ... 259
8.12.2.9.3. eine Status-Zeile (Status-Bar) einbauen ... 260
8.12.2.10. Umgang mit Standard-Dialogen ... 261
8.12.2.11. Listbox-Widget – Auswahl-Listen – List(en)-Boxen ... 262
8.12.2.12. Options-Menüs – Auswahl-Schaltflächen ... 263
8.12.2.13. Scale-Widget – Gleiter / Regler ... 265
8.12.2.14. Scrollbar-Widget - Bildlaufleisten ... 266
8.12.2.15. Widget x ... 266
8.12.x. Tkinter – stark, stärker, noch stärker Objekt-orientiert ... 267
8.12.x.1. nochmal "Hello Welt!" ... 267
8.12.x. diverse Tkinter-Beispiele ... 272
8.13. Internet ... 273
8.13.x. Python und das http-Protokoll ... 273
Variante 1 ... 273
Variante 2 ... 273
8.13.x. einfacher Web-Server ... 275
8.13.x. Python und die eMail-Protokolle (smtp, pop3, imap) ... 276
8.14. besondere mathematische Möglichkeiten in Python ... 277
8.14.1. imaginäre Zahlen ... 277
8.14.2. Matrizen (Matrixes) ... 277
8.14.3. Python numerisch, Python für Big Data ... 279
Numpy ... 279
Scipy ... 279
Matplotlib ... 279
Pandas ... 280
8.15. Behandlung von Laufzeitfehlern – Exception's ... 281
try … except … else ... 281
try … except … finally ... 282
try … finally ... 282
raise 282
pass 282
8.16. Sortieren – eine Wissenschaft für sich ... 283
8.16.x. Bubble-Sort ... 283
8.16.x. Selection-Sort ... 283
8.16.x. Quick-Sort ... 284

8.16.x. Tree-Sort ... 286
8.16.x. Merge-Sort ... 286
8.16.x. Selection-Sort ... 287
8.16.x. Insertion-Sort ... 287
8.16.x. Gnome-Sort ... 288
8.16.x. Counting-Sort ... 288
8.16.x. Radix-Sort ... 289
8.16.x. Tim-Sort ... 289
8.16.x. Heap-Sort ... 289
8.16.x. Bucket-Sort ... 290
8.16.x. -Sort ... 290
8.16.x. Vergleich ausgewählter Sortier-Algorithmen ... 291
8.16.x. das Häufigste Element finden – der Modus ... 292
Beispiel-Implementierung ... 293
8.17. Nutzung weiterer (/ besonderer) graphischer Benutzer-Oberflächen ... 294
8.18. (die hohe Kunst der) Spiele-Programmierung ... 295
8.19. Python im Geheimen - Kryptologie ... 295
8.19.0. Grundlagen ... 295
8.19.0.1. Codierung ... 295
8.19.0.2. Chiffrierung ... 295
8.19.1. symmetrische Verschlüsselung ... 297
8.19.1.x. CÄSAR-Verschlüsselung ... 297
8.19.1.x. ROT13 ... 298
8.19.1.x.1. ROT13 mit einer Funktion ... 301
8.19.1.x.2. Häufigkeits-Analyse ... 303
8.19.1.x. Umsetzung der CÄSAR-Verschlüsselung ... 305
8.19.1.x. moderne CÄSAR-Verschlüsselung mit Schlüssel ... 306
8.19.1.x. POLYBIOS-Verschlüsselung ... 307
8.19.1.x. VIGENÈRE-Verschlüsselung ... 308
8.19.1.x.y. Krypto-Analyse der VIGENÈRE-Verschlüsselung ... 310
8.19.1.x. bifid-Verschlüsselung ... 311
8.19.1.x. ADFGX-Verschlüsselung ... 314
8.19.1.x. trifid-Verschlüsselung ... 316
8.19.1.x. Four-Square-Verschlüsselung ... 320
8.19.2. asymmetrische Verschlüsselung ... 322
8.20. Code verbessern und optimieren ... 323
8.21. Test-gestütztes Programmieren mit Python ... 324
9. Python, informatisch – Datenstrukturen, Klassen, Automaten, …. ... 325
9.1. Tupel ... 327
9.2. Mengen ... 329
9.2.1. Mengen – einfach ... 329
9.2.1.1. Mengen-Erstellung ... 329
9.2.1.2. Mengen-Operationen ... 330
einfache Operationen ... 330
typischen Mengen-Operationen ... 332
Bearbeitung in Schleifen etc. ... 333
9.2.1.x. automatische Mengen-Generierung ... 333
9.2.2. Mengen – objektorientiert ... 334
9.2.3. Anwendung von Mengen ... 335
9.2.3.1. ein bißchen Graphen ... 335
9.3. Dictonary's - Wörterbücher ... 337
9.3.1. Erfassen von unbekannten Objekten und Zählen der Objekte in einem Wörterbuch ... 341
9.3.2. Objekt-orientierte Operationen mit Dictonary's ... 342
9.7. Listen, die II. – objektorientierte Listen ... 343

9.8. Keller 346
9.9. Warteschlangen 351
9.10. Bäume 354
9.11. Graphen 355
9.12. endliche Automaten 356
9.13. Keller-Automaten 358
9.14. TURING-Automaten 358

7. Problem-Lösen mit Python

7.0. Aufgaben versus Probleme

besser wahrscheinlich Aufgaben-Lösen

Problem-Lösen ist eine Stufe komplizierter und geht im Allgemeinen davon aus, das es noch keine Lösung / keinen Algorithmus zur Bearbeitung gibt bzw. dieser nicht sofort offensichtlich ist

meist Umsetzung von Aufgaben-Stellungen / Pflichten-Hefte in Software gemeint

in der Software-Entwicklung wird aber allgemein von einer Problem-Umsetzung gesprochen
das Erledigen von Aufgaben hat so etwas Profanes / Minderanspruchvolles
das Schreiben von Routine-Funktionen ist nicht die Herausforderung, die Software-Entwickler mit dem Image ihres Berufsstandes verbinden
sie brauchen echte Herausforderungen, welche an die Grenzen der Technik oder der Programmiersprache herankommen
es ist quasi ein Kampf Mann (oder Frau) gegen Maschine, in dem man auf sich allein gestellt ist und unbedingt zum großen Helden werden muss
problematisch ist, dass weder die Chef's der Entwicklungs-Abteilungen noch die Nutzer das honorieren, sie können die Komplexität der Programmierung einfach nicht einschätzen
selbst, die im Team mitarbeitenden Programmierer bekommen von der Heldentat nichts mit, weil sie ihren eigenen einsamen Kampf führen

also was macht der unbemerkte Held - er programmiert so, dass kein anderer sein Programm versteht, so ist ihm vielleicht ein verspäteter Ruhm in Aussicht gestellt

ein Mittel dagegen - mit vielen weiteren Vorteilen - ist die Paar-Programmierung (Pair programming, Tandem-Programmierung)
zwei Programmierer arbeiten gemeinsam und gleichzeitig an einem Problem
der eine tippt und der andere kontrolliert gleich mit, der eine ist also aktiv, der andere eher passiv
nach einer aktiven Zeit wechseln die Paar-Mitglieder ihre Rolle
Kombinationen aus Frauen und Männern haben sich i.A. am Besten bewährt, sie gehen unterschiedlich an Probleme heran, hier ergänzen sich diese Vorgehensweisen
da bleibt die Aufmerksamheit länger enthalten, weil sich die Tätigkeiten abwechseln
große Aufmerksamkeit wird auch auf die Verständlichkeit des Code's gelegt
die Paare werden regelmäßig neu zusammengestellt, damit sich keine eingeschworenen Team's bilden

Vorteile der Paar-Programmierung:

- weniger Programm-Fehler
- gegenseitiges Lernen und Lehren
- mehr Freude an der Arbeit (auch bei Routine-Programmier-Aufgaben)
- kleinere / effektivere Programme
- höhere Disziplin bezüglich Absprachen, Team-Regeln, …
- besserer Code
- Arbeitsabläufe werden belastbarer
- geringeres Risiko, das Know how eines Team's zu verlieren, wenn Mitarbeiter Projekte / Firmen / … wechseln
- Paare werden seltener in der Arbeit unterbrochen

- Paar-Programmierung kann auch verteilt / distanziert (z.B. über das Internet) erfolgen

Eine Programmierer-Regel sagt, dass das Finden von Fehlern erst in der Praxis oder bei ersten Tests ungefähr 10x so teuer / aufwändig ist, wie das sofortige Erkennen in der Entwicklungs-Phase
Aber wo es soviele Vorteile gibt, sind die Schattenseiten nicht weit.

Nachteile / Probleme der Paar-Programmierung:

- Paare müssen sich immer wieder aufeinander einstellen, das kostet Arbeitszeit
- Leistungs-Niveau beider Programmierer muss ähnlich sein
- effektiv verlangsamt sich die Programmierung im Vergleich zur parallelen Arbeit beider Programmierer an jeweils anderen Aufgaben
- Urheber-Rechte
- Haftung bei Problemen

Vielfach wird natürlich mit der Entwicklung neuartiger Programm auch informatisches Neuland betreten.

7.0.1. Programm-Entwicklungs-Strategien

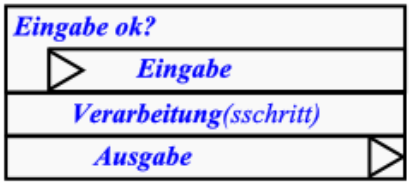
Wie fängt man ein komplexeres Projekt sinnig an? Es einfach von vorne bis hinten in einem Ritt zu schreiben, birgt viele Risiken. Was passiert, wenn es vielleicht gar nicht in die Sprache umsetzbar ist? Oder vielleicht möchte der Auftraggeber auch mal Zwischenergebnisse sehen?

In der Praxis haben sich Grund-Techniken für die Programm-Entwicklung herauskristallisiert. Bei der einen Variante – dem **Top-down-Entwurf** – beginnt man mit einem sehr einfachen Programm-Rahmen. Im einfachsten Fall ist es einfach nur der Aufruf eines leeren Hauptprogramms.

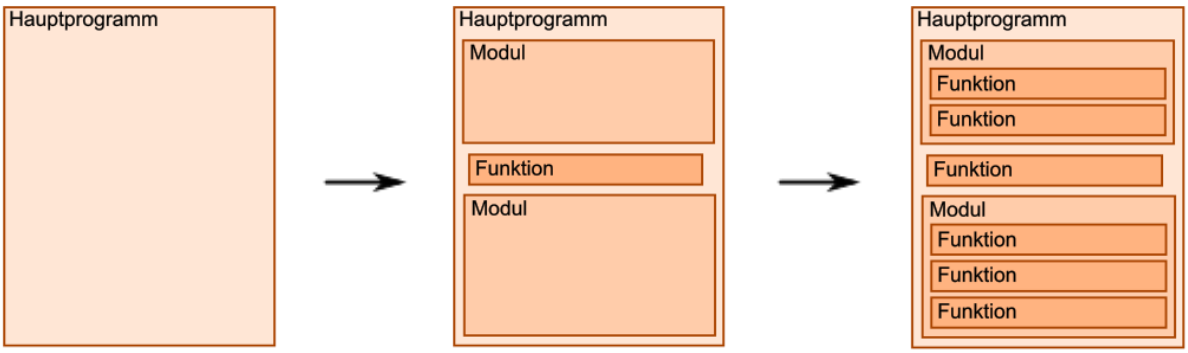
Nach und nach ergänzt man nun einzelne Komponenten. Z.B. könnte man das leere Hauptprogramm in die Teile Eingabe, Verarbeitung und Ausgabe strukturieren.

Im nächsten Schritt erweitert, ergänzt oder verbessert man die einzelnen Komponenten bis man schließlich ein fertiges Produkt erzielt.

Man spricht hier von Deduktion. Die Entwicklung erfolgt quasi von oben nach unten, vom Allgemeinen zum Speziellen.



Der große Vorteil dieser Variante ist, dass man praktisch zu jeder Zeit ein funktionierendes Programm hat, dass nach und nach immer besser / Leistungs-fähiger / Fehler-freier wird.



Erstellen eines Rahmenprogramm's

Erweitern um Modul-Grundgerüste

Ergänzen der Arbeitsfunktionen

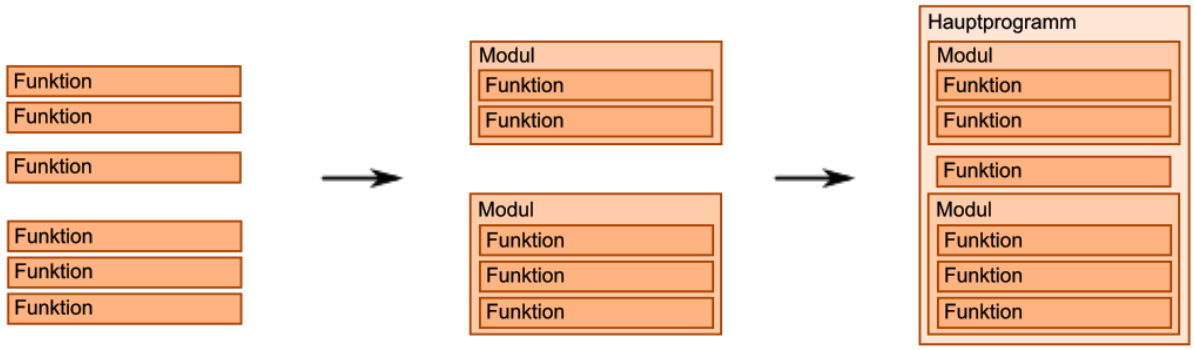
Nachteilig wirkt sich hier aus, dass der komplizierteste Teil – die spezielle Datenverarbeitung – irgendwie fast immer zum Schluss übrig bleibt. Wenn jetzt was nicht läuft, dann hat Huston ein wirkliches Problem. Die möglichen Konsequenzen sind halbfertige Programme, die erst beim Kunden reifen (sogenannte Bananen-Software) oder Verzögerungen beim Zeitablauf.

Sachlich steckt hinter diesem Programmier-Prinzip die Dekomposition. Sie beinhaltet die Zerlegung / Auflösung eines Ganzen in immer kleiner werdende Teile / Segmente. In der Programmierung sind das dann Module, Unterprogramme, Prozeduren oder Funktionen.

Natürlich kann man auch zuerst die spezielle(n) Funktion(en) entwickeln und testen. Schrittweise, werden dann zusätzliche Komponenten hinzugefügt, bis schließlich ein fertiges Programm entstanden ist. Diese Technik nennt man **Bottom-up**.

Es handelt sich hier um eine Induktion, also einer Entwicklung von unten nach oben, vom Speziellen zum Allgemeinen.
Vorteilhaft ist die frühzeitige Fertigstellung der kritischen Programmteile.
Als Nachteil kann sich dabei herausstellen, dass man zwar super Leistungs-fähige Funktionen entwickelt hat, denen aber ein verbindendes Großes-Ganzes fehlt.
Fehlen dann bestimmte Forderungen aus dem Pflichten-Heft, dann sind vielleicht auch sehr aufwändige Nachkorekturen an den Kern-Funktionen notwendig. Das bedeutet dann erneute Tests, Anpassungen usw. usf. Auch bei dieser Projekt-Lösung kann es zu erheblichen Verzögerungen der Fertigstellung kommen.

Die Bottom-up-Technik ist praktisch eine Aggregation. Segmente / Teile / Funktionen / … werden zu einem Großen-Ganzen vereint.

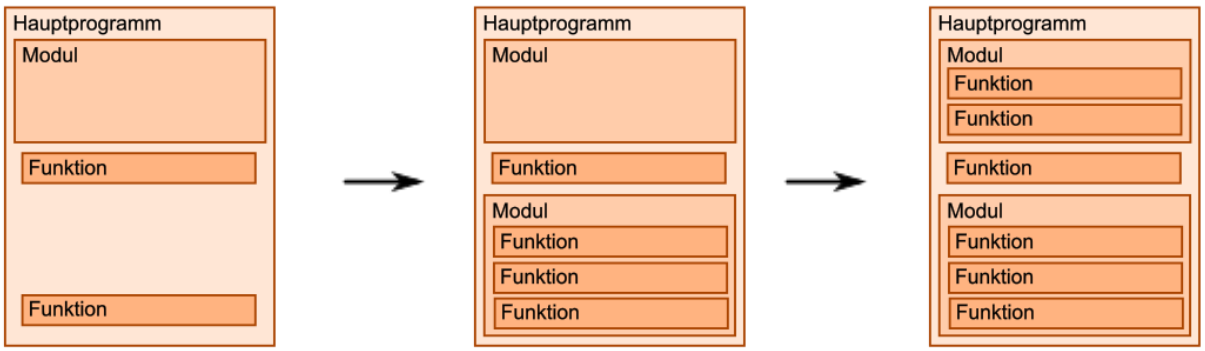


Erstellen der Basis-Funktionen

Zusammenfassen zu Modulen

Zusammenstellen des Hauptprogramm's

Heute werden häufig Top-down- und Bottom-up-Methoden kombiniert. In vielen Software-Schmieden gibt es fertige Sammlungen von Funktionen, die in eine Top-down-Entwicklung nach und nach integriert werden.
Man verlässt sich auf die geprüfte Leistung vorgefertigter Funktionen und hat zu jeder Zeit ein mehr oder weniger gut funktionierendes Programm.



Prototyp mit ersten Funktionen

Erweitern und Modularisieren

Ergänzen fehlender Funktionen

Eine weitere Strategie, die in der letzten Zeit viel von sich reden lassen hat, ist "**Design Thinking**". Darunter versteht man Methoden, Verfahren und Fähigkeiten, sich in Aufgaben-

stellungen und / oder Probleme hineinzufühlen, sie kreativ zu bedenken, Ideen zu kommunizieren sowie produktiv und kollaborativ zusammenzuarbeiten. Vieles wird vorrangig aus der Sicht des Nutzers / Endverbrauchers betrachtet und dieser steht auch im Mittelpunkt. Letztendlich muss dieser mit dem Produkt leben und arbeiten.
In die Entwicklung eines Produkt's oder der Lösung eines Problem's sollen von Anfang an möglichst alle beteiligte Personen-Gruppen einbezogen werden. Team-working, selbstkritisches und kollaboratives Arbeiten sind Kern des Arbeitens. Fehler dürfen gemacht werden, sollen aber möglichst frühzeitig erkannt werden, ohne nach "Schuldigen" zu suchen. Es sollen schnellsens Korrekturen, Verbesserungen und Erweiterungen umgesetzt werden.

Phasen des Design Thinking

• **Phase 1: Situations-Analyse**	Wie ist der aktuelle Stand? Welche Situation ist unbefriedigend? Welches Problem gibt es?
• **Phase 2: Perspektiven entwickeln**	Was wäre das Tollste / Utopischte / …, was man sich als Lösung des Problem's oder der Situation denken könnte? Sei kreativ! (Be creative!) Was gefällt an anderen Lösungen nicht?
• **Ideen generieren**	Was kann mit den gegebenen Mitteln realisert werden? Was soll unbedingt realisiert werden?
• **Konzept-Entwicklung**	Entwickeln erster Prototypen, die einzelne Teil-Probleme lösen, die einen Eindruck von der Lösung geben bzw. die einen ersten Lösungs-Ansatz umsetzen.
• **Testen des Konzept's**	Ausprobieren der Teil-Lösungen sowie des fertigen Produkt's in der Real-Umgebung. Solange die Lösung unbefriedigend oder noch unvollständig oder erweiterbar ist, wird wieder mit Phase 1 gestartet.

Aufgaben:

1. *Vergleichen Sie die Startegien "Top-down", "Bottom-up" sowie die gemischte in einer geeigneten Tabelle!*
2.

7.0.2. Strategien zur Lösung von (echten) Problemen

Zerlegen des Problems in kleinere Aufgaben / Probleme
Analogien zu anderen Problemen suchen (und dann deren Lösungen als Grundlage benutzen)
Versuch und Irrtum (trial and error)
Lernen durch Einsicht

Hilfsmittel / Techniken:

- **Brainstorming**
- **Mind Mapping**
- **Concept Mapping**
- **Kopfstand-Technik**
  Umkehr-Technik
  Flip-Flop-Technik
  1. Aufgabenstellung umdrehen
  2. Lösung für diese Aufgabe suchen
  3. Lösung auf den Kopf stellen
  4. Lösung anpassen / optimieren
- **Negativ-Konferenz**
- **Provokations-Technik**
  Provokation z.B. durch Verallgemeinerung, Pauschalisierung, … als Inspirations-Quelle / zum Verlassen der eingetretenen Denkpfade
- **Superposition**
- **kollektive Notizzettel (collective Notebook, CNB)**
  über einen bestimmten Zeitraum sammeln die Team-Mitglieder ihre Gedanken, Assoziationen, Geistesblitze und Ideen auf Notizzetteln zu notieren → 3 Phasen:
  1. Vorbereitung (Problemstellung formulieren; Teilnehmer auswählen; Notizblöcke bereitstellen)
  2. Durchführung (Notizen machen (spontan und täglich); persönl. Zusammenfassung erstellen)
  3. Auswertung (Zusammenfassungen abgleichen; Notizen durchgehen; Basis-Vorschläge für Lösung heraussuchen / ableiten; Konzept-Erstellung)
- **Pinnwand-Moderation ähnlich: Clustern**
  Sammeln von Ideen- wie beim Brainstorming – allerdings auf Kärtchen / Post-ist; wiederholte Gruppierung der Kärtchen und Gruppen-Benennung; Zusammenfassung des Ergebnisses in möglichst neutraler Form
- **EDISON-Prinzip**
  1. Erfolgs-Chancen erkennen
  2. eingetretene Pfade verlassen
  3. Inspirationen suchen
  4. Spannung erzeugen
  5.Ideen und Erkenntnisse ordnen
  6. Nutzen herausziehen
- **progressive Abstraktion**
  Finden von Zusammenhängen zwischen Problem und erwarteter Lösung; Festlegen von Maßnahmen(-Ebenen) die am Erfolg-versprechendsten erscheinen
- **semantische Intuition**
-

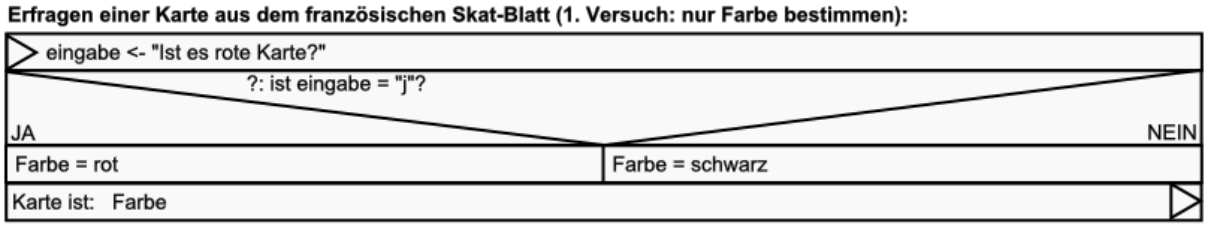
für klassische (sofort lösbare) Aufgaben ("einfache Probleme") bietet sich die folgende Vorgehensweise an:

1. Erfassen des IST-Zustandes (IST-Ananlyse, Ist-Aufnahme, …)
2. Erkennen / Aufzeigen des Unterschiedes / Widerspruchs zum SOLL-Zustand
3. Suchen nach geeigneten Lösungs-Verfahren / Algorithmen
3. Anwendung eines Lösungs-Verfahrens (/ Algorithmus)
4. Prüfen des erreichten Standes bis IST und SOLL gar nicht mehr nur noch im akzeptablen Maß abweichen (ansonsten quasi Zurücksprung zu 1.)

(c,p) 2015 - 2022 lsp: dre

Beispiel für Top-down-Strategie: Erfragung einer Karte aus dem französischen Blatt

1. Prototyp – Test des Verfahrens



Die Umsetzung des Struktogramm wird ganz schematisch erledigt. Für den ersten Versuchs-Prototypen verzichten wir auf fast jeden Schnickschnack. Wir wollen lediglich sehen, ob es funktioniert.

```
eingabe = input("Ist es eine rote Karte? (j)")
if eingabe == "j":
    farbe = "rot"
else:
    farbe = "schwarz"

print("Die Karte ist: ",farbe)
```

Eingabe-Block
Verzweigungs-Block
Ja-Zweig
Nein-Zweig
Ausgabe-Block

```
>>>
```

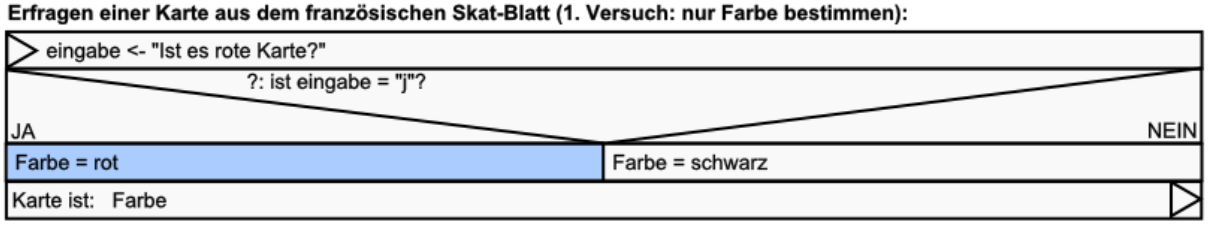
Neben der reinen Funktionalität, sollte man auch gleich die Handhabbarkeit prüfen. Wie kann man die Eingaben für den Nutzer am Einfachsten machen, was macht ein Nutzer intuitiv? Nichts ist nerviger, als eine umständliche Bedienung. Wird der Nutzer z.B. auf deutsch gefragt und muss er dann aber mit "y" für "ja" antworten, da geht das spätestens bei der zweiten Fragen mächtig auf den Docht. Grundsätzlich muss man sich da von seiner egomanischen, selbstbezogenen Zufriedenheit trennen. Es gibt für den Programmierer nur einen "Gott" / eine Werte-Instanz und der sitzt vor dem Computer und versucht das Programm zu bedienen.
Für unsere Ja-Nein-Fragen werden wir nur die Abfrage auf "j" programmieren, das ist verständlich und auch gut zu programmieren. Um vielleicht noch etwas flexibler zu sein, fragen wir auch den Groß-Buchstaben ab. Die notwendige Erweioterung des Programms ist leicht gemacht und wenn wir dann ein Grundgerüst für das Fragestellen und –auswerten haben, dann können wir die restlichen "paar" Fragen mit copy-and-paste dazuprogrammieren.

```
eingabe = input("Ist es eine rote Karte? (j)")
if eingabe == "j" or eingabe == "J":
    farbe = "rot"
else:
    farbe = "schwarz"

print("Die Karte ist: ",farbe)
```

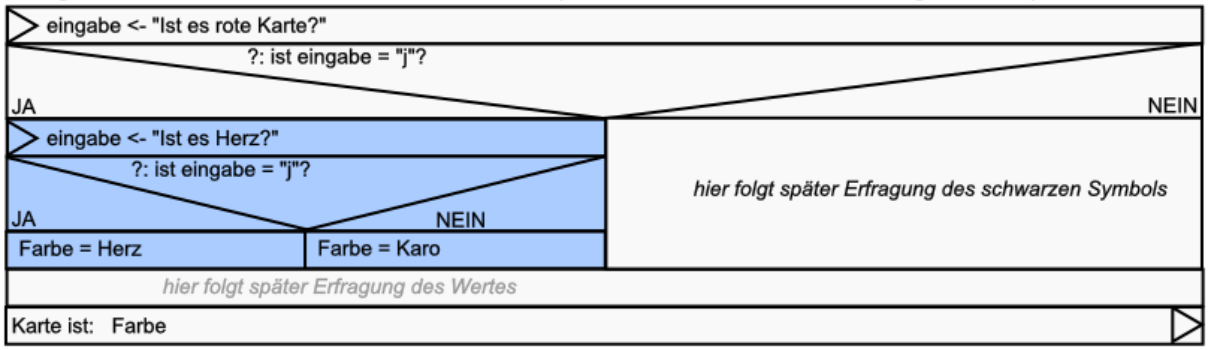
Erweiterung um "J"

2. Schritt – Austausch eines allgemeinen Blockes gegen differenzierte Blöcke

Erfragen einer Karte aus dem französischen Skat-Blatt (1. Versuch: nur Farbe bestimmen):



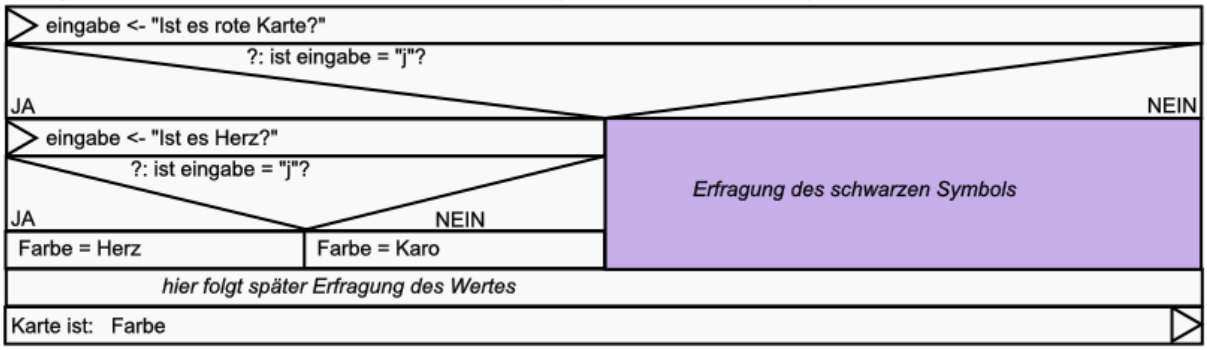
Erfragen einer Karte aus dem französischen Skat-Blatt (1. Teil: Farbe bestimmen; Teillösung rote Karte):



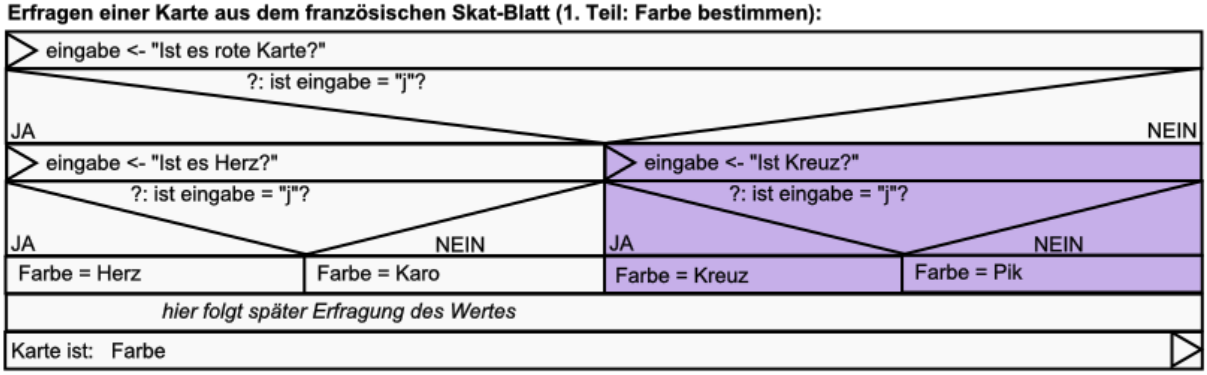

```
>>>
```

3. Schritt – Erweiterung / Vervollständigung

Erfragen einer Karte aus dem französischen Skat-Blatt (1. Teil: Farbe bestimmen):



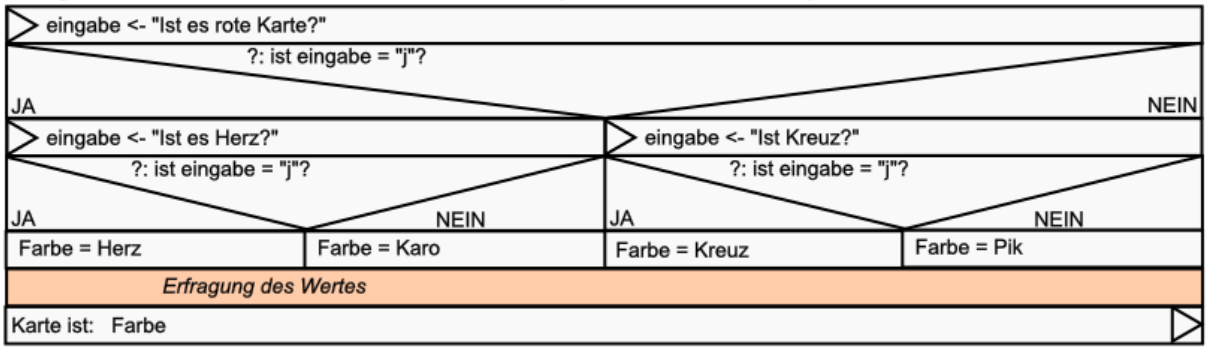
Erfragen einer Karte aus dem französischen Skat-Blatt (1. Teil: Farbe bestimmen):



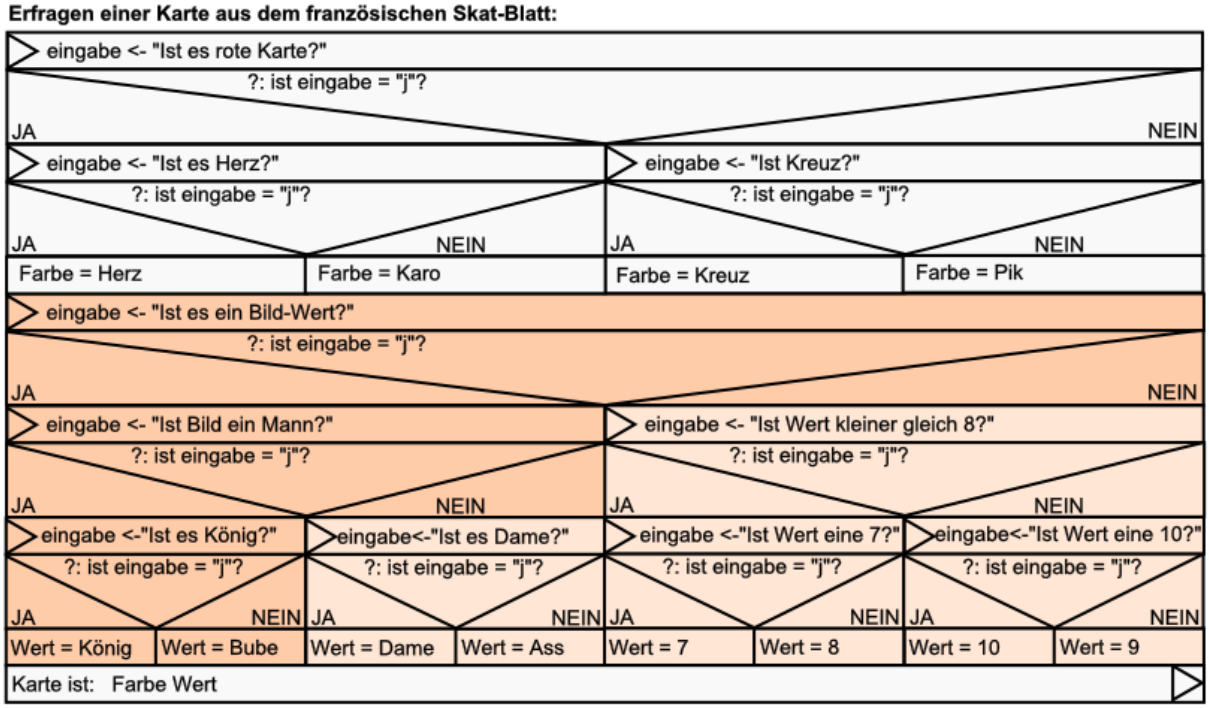

```
>>>
```

4. Schritt – nächster Abschnitt

Erfragen einer Karte aus dem französischen Skat-Blatt (1. Teil: Farbe bestimmen):



Erfragen einer Karte aus dem französischen Skat-Blatt:




```
>>>
```

letzter. Schritt – Verschönerung / Verfeinerung / Benutzerführung optimieren

```
>>>
```

kleine Programm-Beispiele

7.0.3,14 Python am Pi-Day

Am 14. März ist der Pi-Day. An diesem Tag beschäftigen sich Mathematiker und Schüler mit der wohl berühmtesten Konstante der Kreiszahl π. Das Datum ergibt sich aus der amerikanischen Datum-Notation "3/14". Die ganz hart Gesottenen feiern exakt um 01:59:26 Uhr, um Pi bis auf die 7. Nachkommastelle zu ehren.
Auch der 22. Juli wird gelegentlich als Pi-Annäherungstag zelebriert. Hier ergibt sich das Datum aus dem Bruch 22/7, der rund 3,14 – also π ergibt.
Hier seien einige Programme vorgestellt, die Pi auf irgendeine Variante berechnen oder ermitteln.
Vielleicht geht der eine oder andere Quelltext über das gegenwärtige verständnis von Python hinaus, das soll aber bei einem so spannenden Thema nicht das Abbruch-Kriterium sein.

Pi-Berechnung durch Monte Carlo Simulation

```
# pi-monte_carlo.py
# pi-Bestimmung mit der Methode von Monte Carlo
from random import random
print "Monte Carlo Methode zur"
print "Näherung für pi:"
g = input("Gesamtzahl der Tropfen: ")
v = 0
x=0; y=0  # Koordinaten des Punktes P
for i in range(1,g+1):
      x = random()
      y = random()
      if x*x+y*y<= 1:
         v = v + 1
pi_naeh = 4.0*v/g
print g,"Tropfen, davon",v,"Tropfen im Viertelkreis,"
print "pi etwa",pi_naeh
```

Q: http://www.michael-holzapfel.de/progs/python/python_beisp.htm

Pi-Berechnung über Verhältnis der Flächen von äußeren und inneren Vieleck

Interations-Term

$$s_{n+1} = \frac{s_n}{\sqrt{2+\sqrt{4-s_n^2}}}$$

```
# pi-berechn2.py
# pi-Berechnung mit regulären 2n-Ecken
from math import sqrt, pi
n = 6  # Start mit regulärem Sechseck
s = 1  # Seitenlänge des reg. Sechsecks
print "Schrittweise Näherung von pi mit Hilfe eines 2n-Ecks"
for i in range(1,21):
    pi_naeherung = 0.5*n*s
    print pi_naeherung
    s = s/sqrt(2+sqrt(4-s*s))
    n = 2*n  # doppelte Eckenzahl
print "Gute Iteration!"
print "pi =",pi
```

Q: http://www.michael-holzapfel.de/progs/python/python_beisp.htm

Exkurs: besondere Zahlen – Stoff für viele Python-Programme

(einfache) besondere Zahlen

Dreieckszahlen: **Beispiele / Folge:**
1, 3, 6, 10, 15, …

Dreieckszahlen lassen sich über die Formel $x_i = \frac{i\,(i+1)}{2}$ berechnen (mit *i* dem Rang der Zahl).
Die 10 ist nach PYTHAGORAS eine heilige Zahl, weil sie sich aus der Summe der ersten i Zahlen (also: 1+2+3+4) ergibt. Außerdem lässt sich daraus ein vollkommen gleichseitiges Dreieck legen.

EULERsche Zahl: **Wert:**
2,718'281'828'…

Die EULERsche Zahl e berechnet sich als Grenzwert $e = \lim \left(1 + \frac{1}{n}\right)^n$ bzw. als unendliche Folge $e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$ bzw. in der Summen-Schreibweise $e = \sum_{k=0}^{\infty} \frac{1}{k!}$
Die EULERsche Zahl ist eine der bedeutenden Konstanten in der Naturwissenschaft und Mathematik.

goldener Schnitt: **Beispiele:**
irrationale Zahl
$= \frac{1+\sqrt{5}}{2}$
Der Quotient aus zwei aufeinanderfolgenden FIBONACCHI-Zahlen nähert sich immer mehr dem goldenen Schnitt an.

Kreiszahl π: **Wert:**
irrationale Zahl 3,141'592'654'…
stellt Verhältnis von Umfang und Durchmesser eines Kreises dar
Verbindung zum goldenen Schnitt 6/5 [2]

narzißtische Zahlen: **Beispiele:**
Eine narzißtische Zahl mit n Stellen ist gleich groß der Summe der n.-Potenzen ihrer Ziffern.
$153 = 1^3 + 5^3 + 3^3$
$54748 = 5^5 + 4^5 + 7^5 + 4^5 + 8^5$

Vampir-Zahlen: **Beispiele:**
Vampir-Zahlen haben eine gerade Anzahl von Stellen und lassen sich aus einer beliebigen Multiplikation von Zahlen, die halb soviele Stellen besitzen, wie die Vampir-Zahl selbst hat, keine führende Null beinhalten und insgesamt alle Ziffern der Vampir-Zahl enthalten.
1260 = 21 * 60
1530 = 30 * 51
2187 = 27 * 81

Urantia-Zahlen: **Beispiele:**

Zahl des heiligen AUGUSTINUS: **Beispiele:**
Die Zahl des heiligen AUGUSTINUS ist die erste narßistische Zahl. Sie lässt sich auch aus der Summe der Fakultäten von 1 bis 5 berechnen.
$153 = 1! + 2! + 3! + 4! + 5!$
$153 = 1^3 + 5^3 + 3^3$

???: **Beispiele:**

über Teilersummen definierte besondere Zahlen

Teilersumme: **Beispiele:**

die Teiler-Summe σ (einer Zahl) ist die Summe aller ihrer Teiler beginnend bei 1 und abschließend mit der Zahl selbst	$\sigma(12)$ = 1+2+3+4+6+12 = 28
echte Teilersumme: die echte Teiler-Summe σ^* (einer Zahl) ist die Summe aller ihrer Teiler beginnend bei 1 und dabei die Zahl selbst ausschließend	**Beispiele:** $\sigma^*(12)$ = 1+2+3+4+6 = 16
defiziente Zahl: eine Zahl heißt dedizient (oder teiler-arm), wenn die die echte Teilersumme kleiner als die Zahl selbst ist $\sigma^*(n) < n$	**Beispiele:** $\sigma^*(10)$ = 1+2+5 = 8 8 < 10
abundante Zahl: eine Zahl heißt dedizient (oder teiler-reich), wenn die die echte Teilersumme größer als die Zahl selbst ist $\sigma^*(n) > n$	**Beispiele:** $\sigma^*(12)$ = 1+2+3+4+6 = 16 16 > 6
vollkommene Zahl: eine Zahl heißt vollkommen, wenn die die echte Teilersumme die Zahl selbst ist $\sigma^*(n) = n$	**Beispiele:** $\sigma^*(6)$ = 1+2+3 = 6 6 = 6

befreundete Zahlen: **Beispiele:**

Befreundete Zahlen sind zwei unterschiedliche natürliche Zahlen, bei deren die echte Teilersumme genau der anderen Zahl entspricht.

1184 und 1210
5020 und 5564

sonstige (ganz) besondere Zahlen

???: **Beispiele:**

???: **Beispiele:**

seltene oder ungewöhnliche Zahlen und Zahlensysteme (in der Schule)

komplexe Zahlen: **Beispiele:**

→ in Python intern definiert, dadurch sofort nutzbar

???: **Beispiele:**

Q:

8. Python für Fortgeschrittene

Nachdem wir die grundlegenden Elemente von Python besprochen haben, gehen wir jetzt mehr in die Detail's. Natürlich gibt es keine echte Grenze zwischen Grundlagen und fortgeschrittener Programmierung. Ab nun schauen wir auch intensiver hinter die Oberfläche und kümmern uns um spezielle Eigenschaften und Möglichkeiten.
Auf den folgende Seiten verzichte ich jetzt auch dann und wann mal auf die farbige Darstellung der Quell-Texte. Wer an dieser Stelle einsteigt, sollte genug Grundkenntnisse besitzen, um mit Python umzugehen. Natürlich können einzelne Quelltexte und Programme auch im Blindflug benutzt und ausprobiert werden. Ob das Sinn macht, muss jeder für sich entscheiden. Aber ein Blindflug wird auch nicht an der Farbigkeit des Quelltextes im Editor scheitern.

Wegen der besonderen Bedeutung von Texten besprechen wir diese hier in einem gesonderten Abschnitt. Im Programmier-Jargon heißen sie Strings (engl. Fäden) oder Zeichenketten.
Einige Programmiersprachen betrachten Zeichenketten auch als eigenständigen / erweiterten Datentyp. Python trennt hier nicht so streng.

8.1. Strings – Zeichenketten

Wie wir schon vilefach gesehen haben, sind Zahlen für sich nicht sehr informativ. Wir brauchen immer Beschreibungen, um die Zahlen in sinnvolle Zusammenhänge zu bringen. Schon die Ausgabe des Namens einer (physikalischen) Größe oder die Nennung einer Einheit sind mit Text-Symbolen verbunden. Erst so macht z.B. eine "21" Sinn. Wenn denn nämlich noch "Temperatur" und die Einheit "°C" dazu angegeben wird, dann verstehen wir die 21 auch im Speziellen.
Heute ist die Verarbeitung von Zeichenketten eine der häufigsten Tätigkeiten / Aufgaben für Programmierer. Viele Daten liegen zuerst einmal als Zeichenketten vor. Bevor man sie für Berechnungen usw. nutzen kann, müssen sie ersteinmal aufgearbeitet werden (→ 8.2. Datentypen und Typumwandlungen).

8.1.1. einzelne Symbole / Zeichen / Charaktere

wir sprechen auch von Charakteren – abgekürzt in vielen Programmiersprachen mit char oder chr
gemeint ist die Repräsentation von Zeichen im ASCII-Zeichensatz oder in den modernen Versionen der Programmiersprachen im Unicode-Zeichensatz

Symbole müssen im Programm-Text entweder in einfache Hockkommata oder Anführungszeichen gesetzt werden
immer nur ein gültiges Zeichen

Beispiele:
'a'
'1'
' '
'#'

Speicherung in Variablen möglich

Umwandlungen von Symbolen und ASCII-Code

ord()
gibt für ein Zeichen / Charakter den ASCII-Code zurück

chr()
wandelt einen ASCII-Code (Ganzzahl!) in ein Symbol / Charakter um

8.1.2. Sequenzen von Zeichen - Zeichenketten / Strings

erste allgemeine und unterschwellige Besprechung schon weiter vorne (u.a. → 6.1. Ausgaben und 6.4.2.2. Sammlungs-bedingte Schleifen)
hier noch einmal mit zusammenfassendem Charakter

im Programmierer-Jargon Strings genannt

Zeichenkette ist eine Symbol-Folge
Im Programm-Text / Listen usw. müssen Strings / Texte entweder in einfache Hockkommata oder Anführungszeichen gesetzt werden
Empfehlung (aber kein Muss!) einzelne Symbole in einfache Hochkommata und Strings in Anführungszeichen

Zeichenketten sind unveränderlich (immutable), einmal definiert sind sie nur durch direktes oder indirektes Kopieren / Manipulieren zu verändern
eine Zeichenänderung über zeichenkette[3] = 's' ist nicht möglich

Symbole / Zeichenketten lassen sich durch Addition (+) verketten / konkatenieren
ein Symbol / eine Zeichenkette läßt sich durch Multiplikation (*) wiederholend verketten / konkatenieren

Vergleich – wie in Python üblich – über == bzw. !=
für die andereren Vergleiche gelten die lexikalischen Ordnungen
ein längerer String ist immer größer

es läßt sich der **in**-Operator verwenden
also prüfen, ob ein Symbol / Teilstring in einem anderen String enthalten ist

len()
Länge der Zeichenkette / Buchstaben-/Zeichen-Anzahl

str()
Umwandlung in einen String

Zugriff auf einzelne Zeichen über den Index
Zählung beginnt mit 0 für das erste Zeichen

zeichen = zeichenkette[Index]

es ist auch der Zugriff auf Zeichenketten-Abschnitte möglich (Slice-Notation)
zeichenkettenabschnitt = zeichenkette**[:3]** liefert die ersten drei (3) Zeichen (Quasi bis zum 3. Zeichen)
zeichenkettenabschnitt = zeichenkette**[3:]** kopiert alle Zeichen ab dem dritten bis zum Zeichenkettenende
zeichenkettenabschnitt = zeichenkette**[4:7]** in der Variable zeichenkettenabschnitt befindet sich die Zeichen von Position 4 bis 6 (also nicht mehr 7)

die Indizes können auch negative Zahlen sein, dann wird von rechts nach links – also quasi vom Ende her – gearbeitet (0 und -0 ist aber die gleiche Position!)
[:-3] liefert die Zeichenkette ohne die letzten drei Zeichen
[-4:] liefert die letzten vier Zeichen der Zeichenkette

Ausgaben mit Platzhaltern

print("Hauptzeichenkette %s Restzeichenkette" % "Zeichenkette")
print("Hauptzeichenkette %s Restzeichenkette" % ZeichenkettenVariable)

print("Hauptzeichenkette %s Restzeichenkette %s weitere Zeichenkette" % (Zeichenkette1, Zeichenkette2))

8.1.1. Objekt-orientierte Nutzung von Strings

Das hört sich irgendwie gefährlich an – Objekt-orientierte Nutzung von Strings – ist aber eigentlich gar nicht sowas Neues. Viele der schon besprochenen / genutzten Module realisieren genau das moderne Objekt-Konzept. Wir werden uns später genauer damit beschäftigen. Also keine Angst – einfach ran an die Bouletten.

Zeichenkette**.strip()**
Zeichenkette**.strip([zeichen]):**:
Entfernt Leerzeichen und Zeilenumbrüche von den Enden des Strings
innere Leerzeichen und Zeilenumbrüche bleiben erhalten

Zeichenkette**.lower()**
Umwandlung in Kleinbuchstaben

Zeichenkette**.upper()**
Umwandlung in Großbuchstaben

Zeichenkette**.append(e)**

Zeichenkette**.extend(l)**

Zeichenkette**.count(e)**

Zeichenkette**.index(e)**

Zeichenkette**.insert(i,e)**

Zeichenkette**.pop(i)**

Zeichenkette**.remove(e)**

Zeichenkette**.reverse()**

Zeichenkette**.sort()**
Zeichenkette**.sort(reverse=True)**

Zeichenkette**.find(e)**
Zeichenkette**.find(e,istart)**
Zeichenkette**.find(e,istart,iende)**

Zeichenkette**.rfind(x,istart,iende)**

Zeichenkette**.rjust()**

Zeichenkette**.ljust()**

Zeichenkette**.replace(ealt,eneu)**

Zeichenkette**.endwith(zeichen,anzahl)**

Zeichenkette**.split()**
Zeichenkette**.split(Trennzeichen)**
split() teilt eine Zeichenkette in Wörter auf. Diese Wörter werden als Liste zurückgegeben. Als Ternnzeichen wird das Leerzeichen benutzt.
Soll ein spezielles Trennzeichen verwendet werden, dann kann dieses bei splitt() als Argument angegeben werden.
Auf diese Art lassen sich z.B. Zeilen aus CSV-Dateien in ihre Elemente Zerlegen, wenn man das gültige Trennzeichen kennt. Da alle Elemente der Liste Texte sind, muss u.U. noch eine Umwandlung in Zahlen – wenn es denn solche sind – erfolgen.

Zeichenkette**.rsplit()**

join(Liste)
erzeugt aus den Elementen der Liste einen verketteten Text
braucht man z.B. Leerzeichen zwischen den Elementen, dann kann man dies so notieren:
" ".join(Liste)
Für die Erzeugung von CSV-Zeilen lässt sich statt dem Leerzeichen natürlich auch ein anderes Trennzeichen verwenden.

8.1.2. besondere Möglichkeiten für Strings in Python

zwei aufeinanderfolgende Literale werden automatisch verknüpft
'Pyt' 'hon' ergibt 'Python'
schöner natürlich mit **+**-Operator: 'Pyt' + 'hon'

gilt nicht für beliebige Kombinationen mit Zeichenketten(-Funktionen)
mit **+**-Operator aber beliebige Zeichenketten-Kombinationen realisierbar

Zahlen müssen ev. vorher mittels **str()**-Funktion in eine Zeichenkette umgewandelt werden

8.2. Datentypen und Typumwandlungen

Irgendwie hat Python fast immer erkannt, mit was für eine Art Daten wir arbeiten. Diese Flexibilität wird von jüngeren Programmierern gelobt und von den älteren / klassischen Programmierern als deutlicher Mangel von Python hervorgehoben.

Grundsätzlich hatten wir es bis hierher mit zwei Datentypen zu tun, die Zahlen und die Texte. Weitere Datentypen sind None als leeres Objekt oder eben "Nichts" und

Bei den Zahlen unterscheiden Informatiker mehrere klassische Zahlen-Arten, die sich zwar an mathematischen Typen orientieren, aber im Wesentlichen unterschiedlich im Prozessor (CPU) verarbeitet werden.
Die einfachste Art Zahlen sind die ordinären bzw. ganzen Zahlen. Sie sind die praktische Darstellung einer Zahl im Dualsystem. Da sie keine Nachkommstellen haben und somit das Komma immer an der rechten Seite haben, spricht man auch von Festkomma-Zahlen. Für das Vorzeichen ist bei einigen (bei Python bei allen) Zahlen-Formaten das höchstwertige Bit reserviert. Diese Art der Zahlen-Darstellung haben wir prinzipiell schon vorgestellt (→ 3.1.2.1. Mathematik für Informatiker – binäres Rechnen). In Python ist das kleinste Festkomma-Zahlenformat der Typ int. Das früher vorhandene und größer als int definierte Typ lon ist in int aufgegangen. Somit gibt es nur noch int, was den Umgang mit Festkommazahlen erleichtert. Die darstellbaren Zahlen sind nicht mehr begrenzt . Für eine Zahl oder eine Variable mit diesem Typ werden also immer viele Bytes vom Haupt-Speicher verbraucht.

Typ	Beschreibung	ev. Grenzen, …	Beispiel	
None	nichts; NULL			
Integer	Ganzzahl	mit führender 0 wird Wert als Oktalzahl, mit führenden 0x als Hexadezimalzahl interpretiert!	x = 3 o = 0127 h = 0x6f4ea	int()
Float	Fließkommazahl Gleitkommazahl		f = 7.85 f1 = 2e6 f2 = -7.25e-3	float()
Complex	komplexe Zahl			complex()
Bool	Wahrheitswert		w = True w = False	bool()
String	Zeichenkette			
List	(veränderbare) Liste (von Elementen) Sequenz			
Tuple	unveränderliche Liste / Sequenz			
Dictionary	Kombinations-Feld Kombinations-Liste Lexikon-Eintrag assoziatives Feld			
Set	(veränderbare) Menge (von Elementen)			
Frozenset	unveränderliche Menge			

Wo sind die Variablen abgespeichert und wieviel Platz (Byte) werden für sie verbraucht?

In Python sind Variablen Referenzen (Verweise / Zeiger) auf bestimmte Speicherzellen. Mittles der id-Anweisung bekommt man die (erste) Speicher-Adresse zurückgeliefert.

Den verbrachten Speicherplatz kann man über die Typ-Bestimmung für die Variable ermitteln. Dazu gibt es die type-Anweisung.

```
>>>
```

eigener Speicher-Verbrauch des Programms:

8.2.1. Zahlen

8.2.1.1. ganze Zahlen

int mit einem Werte-Bereich von -9’223’372’036’854’775’808 bis 9’223’372’036’854’775’807 (-9 Trillionen bis 9 Trillionen)
Das entspricht dem Maximum, was in einer 64bit-Variablen möglich ist

Zahlendarstellung über spezielle Literale

oktale Literale
0**o**724

binäre Literale
0**b**01001

hexadezimale Literale
0**x**A1F31

Umwandlungs-Funktion: **bin()**, **oct()** und **hex()**
arbeiten nur mit int-Zahlen
es handelt sich um Funktionen, die direkt in Python zugänglich sind, ein Modul-Import ist nicht notwendig

8.2.1.2. Fließkommazahlen / Gleitkommzahlen

Zahlen mit Kommastellen nennen die Informatiker auch Fließkomma-Zahlen. Die Zahl besteht dabei immer aus einer vorzeichenbehafteten Mantisse. Die Mantisse hat eine Spanne von 1,0 bis 9,9999… Dazu gehört immer ein Exponent der die zugehörige Zehner-Potenz charakterisiert. Vielleicht kennen Sie die Zahlen-Darstellung als wissenschaftliches Zahlen-Format oder .

$3{,}9745 * 10^{-20}$ → 3.9745e-20

Die Möglichkeit, dass auch nur ein e schon eine – zumindestens theoretisch – mögliche Fließkommazahl ist $(0{,}0 * 10^{0})$, kann in einigen Programmen und / oder Programmiersprachen zu Problemen führen.

float für Gleitkommazahlen ebenfalls als 64bit-Variable
durch spezielle Verteilung der Bit’s für Matisse und Exponent kommt man auf einen möglichen Bereich von $-1{,}797’693’134’862’315’7 * 10^{308}$ bis $+2{,}225’073’858’507’201’4 * 10^{308}$

In der Mathematik gibt es auch noch eine etwas ungewöhnlich anmutende Zahlen-Menge – die imaginären oder komplexen Zahlen. Sie ergeben sich aus der Lösung des Problems um die Berechnung der Wurzel aus -1. Im Bereich der "normalen" Zahlen (natürliche, ganze, reele, rationale Zahlen) gibt es keine Lösung für die Wurzel aus -1.
Zu diesem Daten-Typ und den zugehörigen Verarbeitungs-Möglichkeiten kommen wir in einem späteren Kapitel genauer. Hier ist erst einmal nur wichtig, dass eine komplexe Zahl in Python vom Typ complex ist und aus zwei Bestandteilen besteht, den reelen und den imaginären Teil. In der Mathematik wird der imginäre Teil mit einem i (imaginäre Einheit) gekennzeichnet, in Python wir dafür das j verwendet.

komplexe Zahlen lassen sich als Summe (besser auch in Klammern) aus reelen und imaginären Teil zusammensetzen 4+5j

8.2.1.3. Wahrheitswerte

Zu den Zahlen- oder nummerischen Formaten zählen auch die BOOLEschen Werte für WAHR (TRUE) und FALSCH (FALSE). Sie sind gleichfalls durch 1 bzw. 0 und die Ausdrücke **True** und **False** repräsentiert.
In Python haben alle Werte und Daten-Strukturen einen bestimmten Wahrheitswert. Das ist oft sehr praktisch, kann aber auch schwierig für die Lesbarkeit eines Programm-Codes sein.

Da boolsche Werte zu den nummerischen Datentypen zählen sind neben den typisch boolschen Operatoren auch die arithmetrischen Operatoren zugelassen.
Bei der Verwendung ungewöhnlicher Ausdrücke sollte man ev. eine offensichtlichere Schreibung verwenden oder gut kommentieren.

Operator	Beschreibung / Operation	
a **&** b	bitweise UND	AND
a **|** b	bitweise ODER	OR
a **^** b	bitweise exklusives ODER	XOR
a **>>** n	Bit-Verschiebung um n Stellen nach rechts	entspricht: /2
a **<<** n	Bit-Verschiebung um n Stellen nach links	entspricht: *2
not a	Negation von a	
a **and** b	UND-Verknüpfung	
a **or** b	ODER-Verknüpfung	
a **+** b		
a **-** b		
a ***** b		
a ****** b		
a **==** b	Gleichheit	
a **!=** b	Ungleichheit	

8.2.2. Strings und Co als Datentypen

Die Zeihenketten haben wir schon wegen vieler Besonderheiten und der großen Bedeutung weiter vorne besprochen (→ 8.1. Strings – Zeichenketten).
Hier gehen wir nur noch einmal in Bezug auf Typ-Umwandlungen auf sie ein. Einige Aspekte kommen hier wiederholend vor.

8.2.2.1. einzelne Zeichen

ord()
gibt für ein Zeichen / Charakter den ASCII-Code zurück

chr()
wandelt einen ASCII-Code (Ganzzahl!) in ein Symbol / Charakter um

8.2.2.2. Sequenzen von Zeichen - Zeichenketten

str()
produziert vorrangig für Menschen lesbare Strings

repr()
erstellt Strings, die optimal für den Interpreter sind

float()
wandelt eine Zeichenkette in eine Fließkomma-Zahl um
Achtung! bei Fehler-behafteten Zeichenketten ergibt sich ein Laufzeitfehler
Abfangen von Laufzeitfehlern über Exception's möglich (→ 8.14. Behandlung von Laufzeit-fehlern – Exception's)

int()
wandelt eine Zeichenkette in eine Festkomma-Zahl um
Achtung! bei Fehler-behafteten Zeichenketten ergibt sich ein Laufzeitfehler
Abfangen von Laufzeitfehlern über Exception's möglich (→ 8.14. Behandlung von Laufzeit-fehlern – Exception's)

Aufgaben:

1. ***Schreiben Sie ein Programm, dass einen einzugebenen Text in eine Liste von ASCII-Code's zerlegt und diese Liste dann ausgibt!***
2. ***Erweitern Sie das Programm von der Aufgabe 1 dahingehend, dass die Liste der ASCII-Code's in eine weitere Liste aus Symbolen umgewandelt wird! Lassen Sie diese Liste dann auf dem Bildschirm erscheinen!***
3. ***Im letzten Schritt soll das Programm nochmals um die Rekonstruktion eines String's aus der ASCII-Code-Liste (!!!) erweitert werden!***
4. ***Ein (neues) Programm soll eine einzugebene positive Ganzzahl (z.B.: 134) in "gesprochene" Ziffern-Folgen (z.B.: "eins drei vier") zerlegen!***

für die gehobene Anspruchsebene:

5. ***Erweitern Sie das Programm um die Ausgabe eines Vorzeichens für negative Ganzzahlen!***
6. ***Erstellen Sie ein Programm, dass eine einzugebene Fließkommazahl (z.B.: 183.45) in eine "deutsche", "gesprochene" Ziffernfolge zerlegt! (hier: "eins acht drei Komma vier fünf")!***

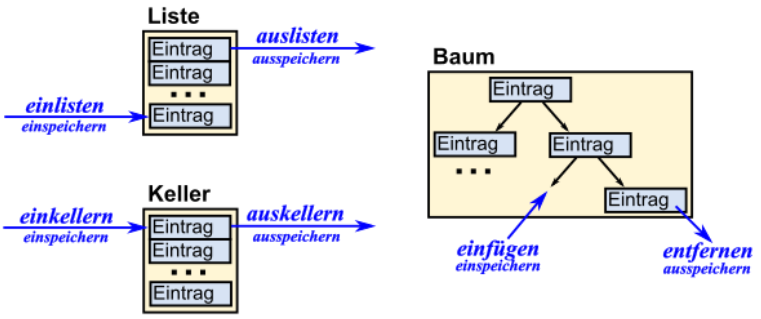
8.2.3. Listen, die I. – einfache Listen

In den vorderen Kapiteln haben wir schon das eine oder andere Mal unterschwellig Listen verwendet, ohne genau auf diese Daten-Struktur einzugehen. Hier sollen nun verschiedene einfache Zugriffs-Möglichkeiten usw. genauer besprochen werden.
Später betrachten wir die Listen auch noch mal aus der Sicht der Objekt-orientierten Programmierung (→ 9.7. Listen, die II. – objektorientierte Listen). Listen sind nämlich auch einfach nur Objekte, für die es dann vorgefertigte Attribute und Methoden gibt. Hier bieten sich dann viele genial-einfache Listen-Nutzungs-Möglichkeiten an, hinter deren Prinzip man aber erst einmal steigen muss. Hier helfen uns die Kenntnisse über die grundsätzliche Art und Weise der Objekt-orientierten Programmierung (→ 8.11. Objekt-orientierten Programmierung).

Definition(en): Datenstruktur
Eine Datenstruktur ist der Informatik eine Vereinbarung zur Organisation und Speicherung von Daten.

wichtige Daten-Strukturen:

- lineare Strukturen:
  - Liste
  - Keller
  - Ring
- verzweigte Strukturen:
  - Baum
  - Netz



ausgewählte (bedeutsame) Datenstrukturen

Speicher-Typ: **FIFO** - First In - First Out, selten auch FCFS (für: First come, first served.)

Daten, die zuerst in die Struktur aufgenommen / gespeichert werden, werden auch zuerst wieder aus ihr entfernt / entnommen

Wer zuerst kommt, mahlt zuerst. First come, first served.

bekannteste und auch häufigste Struktur in informatischen Systemen ist die (Warte-)Schlange
hier auch vielfach Queue oder Pipe genannt

alternativer Speicher-Typ für lineare Daten-Strukturen ist LIFO für "Last IN - First out"
z.B. bei Keller (Stapel-Speicher, Stack) realisiert
im englischen auch LCFS für "Last come - first served."

LIFO und FIFO sind ausschließlich Zeit-bedingt. Prioritäten spielen keine Rolle. Dafür werden dann spezielle Versionen dieser Speicher-Typen realisert.

8.2.3.0. theortische Vorbetrachtungen

8.2.3.0. 1. Listen – eine Form der Datensammlung

Wahrscheinlich sind Listen wohl die älteste Form der strukturierten Daten-Sammlung. Es gibt Höhlenmalerein, in denen die Jagd-Erfolge aufgelistet sind.
Abei auch in unserem heutigen Leben spielen Listen immer noch eine große Rolle. Vielzitierte Beispiele sind:

- Einkaufs-Listen
- Stück-Listen (z.B. am Anfang von Bau-Anleitungen (→ LEGO ®-Bausätze, IKEA ®-Möbel, …)
- Inventar-Listen
- Arbeits-Aufträge (To do-Listen)
- Check-Listen (für Flugzeugstart‘s od.ä.; Reinigungs-Arbeiten, …)
- Vokabel-Listen
- Anrufer-Liste im Telefon
- Mail-Box
- empfangene oder gesendete eMail’s
- Schüler-Liste im Klassenbuch
- Strafakte
- Adressliste / Telefon-Liste
- Messwerte
- Playlist im MP3-Player / Smartphone
- Dateien in einem Ordner
- Inhalts-Verzeichnis
- Handels-Register
- Warte-Liste
- Chroniken
- Wähler-Verzeichnis
- Speisekarte / Menü-Liste
- Robinson-Liste
- Mitglieder-Liste eines Vereins
- Lieferschein
-

Neben den einzelnen Elementen interessieren in Listen oft auch die Anzahl der Einträge. Quantitative Aspekte spielen bei Listen also eine wichtige Rolle. Listen haben einen Inventar-Charakter bezogen auf Objekte, die unter einem Aspekt zusammengestellt wurden.

Definition(en): Listen
Eine Liste ist eine Sammlung von thematisch zusammengehördnen Informationen / Daten in einer sich ständig wiederholenden Form.

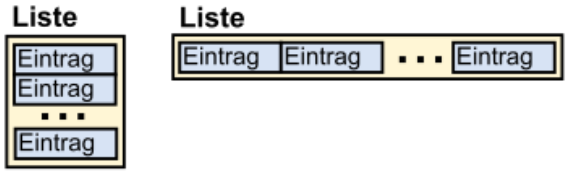
Selbst soetwas wie Spick-Zettel könnte man in die Kategorie Listen einordnen.
Viele der oben erwähnten Anwendungen von Listen gibt es sicher auch schon für unsere modernen Smartphone's als App. Hier ist dann der Bezug zur informatischen Datenstruktur Liste meist besonders leicht herzustellen. Da hier die Daten (z.B. eMail's, Kontakte usw. usf.) als Listen bereitgestellt werden.

Aufgaben:

1. ***Wählen Sie eine Gruppe von 10 App's auf Ihrem Smartphone aus (z.B. die auf einer Seite zusammenstehen) und notieren Sie deren Namen und Zugehörigkeit zu einem App-Typ! Prüfen Sie nun, ob und welche Daten in einer oder mehrerer Listen verwaltet werden!***
2. ***Was sind eigentlich Blacklist, Whitelist und Graylist?***
3.

Listen sind entweder leer oder bestehen aus einem (Kopf-)Element und einem Verweis auf eine (Rest-)Liste
die Rest-Liste kann natürlich auch leer sein

8.2.3.0.2. Daten-Struktur: Liste



Listen sind in Python wohl die Daten-Struktur. Ob wir sie dabei als vertikal oder horizontal angeordnet betrachten, ist für uns völlig egal. Meist wird aber eher eine horizontale betrachtung vorgezogen.
Vielfach sind die Elemente / Einträge in einer Liste gleichartig.
In Python muss dies nicht so sein. Man kann in einer Liste Obejekte ganz unterschiedlicher Typen sammeln. Das können auch wieder Listen sein. Gerade dies macht Listen in Python so unwahrscheinlich flexibel.
Im Vordergrund stehen die folgenden Gründe für den Einsatz von Listen:

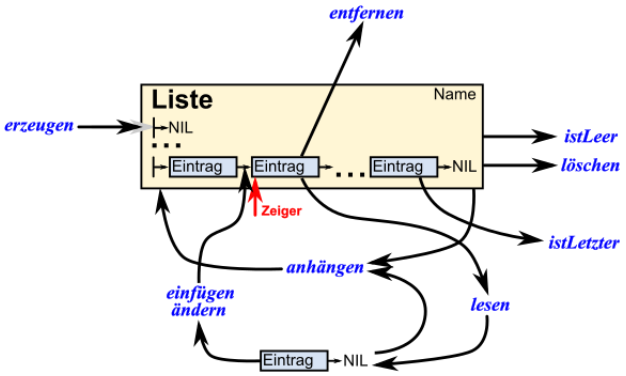
- gleichartige zu bearbeitende Elemente
- einfache Aufzählung von Objekten
- unbestimmte Anzahl von Elementen, Anzahl ist mehr oder weniger stark veränderlich
- Reihen-Folge der Listen-Einträge kann, muss aber keine Rolle spielen

Definition(en): Liste
Im informatischen Sinn versteht man unter einer Liste eine Sammlung von (gleichartig zu bearbeiteten) Elementen in einer Aneinander-Reihung.
Eine Liste ist eine dynamische Datenstruktur von Elementen / Objekten in einer verketteten Form.
Eine Liste ist eine lineare Datenstruktur,

Einlisten – Eintragen eines Elementes / Listen-Eintrag's in eine Liste
Auslisten – Entfernen eines Elementes / Listen-Eintrag's aus einer Liste

Größe der Liste wird maßgeblich vom verfügbaren / bereitgestellten / (dafür) reservierten Speicher bestimmt
die reine Anzahl an Einträgen kann sehr groß werden

Attribute von Listen:
Name
(aktuelle Zeiger-Position)



Operationen / Methoden zu / auf Listen:
initialisieren (init) → erstellen einer Liste
?leer / istLeer (isEmpty) → ist die Liste leer?
einspeichern / anhängen (append) → anhängen eines Element's an eine bestehende Liste

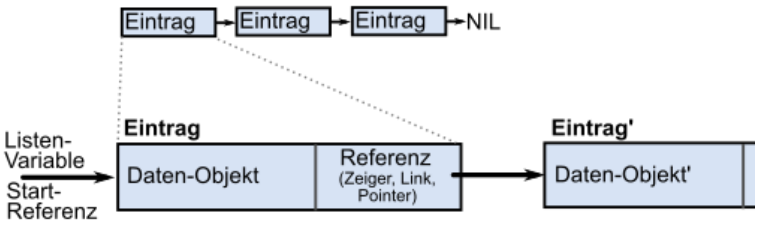
lesen / ausspeichern (read) → Lesen eines Eintrags, ohne ihn zu löschen
geheZumAnfang / (toFirst) → gehe zum ersten bzw. Kopf-Element
tauscheElement / (remove) → ersetze das aktuelle / ausgewählte Listen-Element durch ein anderes
einfügen / (insert) → ein Element an der aktuellen Stelle (Zeiger-Position) einfügen und den Rest der Liste anhängen
vorne einfügen / ()
entfernen / löschen () → entfernen / löschen des aktuellen Element's (Rest-Liste wird an den Vorgänger angehängt)

istEintrag / suche (search /) → ist ein bestimmter Eintrag in der Liste vorhanden?
finde / gibPosition (find / getPosition) → gibt die Listen-Position eines Eintrag's an / zurück
anzahl / länge (length) → gibt die Anzahl an Einträgen in der Liste zurück

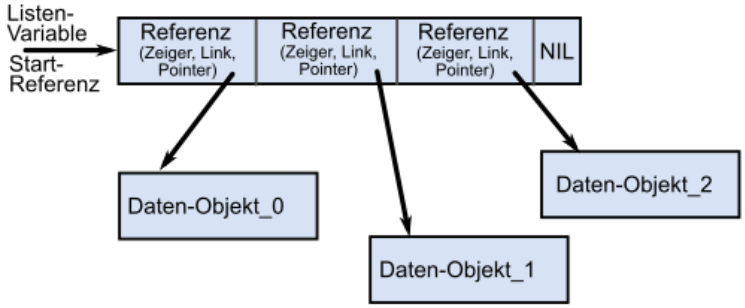
Kopf-bezogenes Arbeiten:
insert bzw. push zum Einspeichern (neue Liste ::= neuer Eintrag, alte Liste)
pop(0) zum Ausspeichern (liefert das Kopf-Element zurück und entfernt es vom Kopf der Liste)
Liste wird als Konstrukt aus Kopf und Rest verstanden

Schwanz- / Ende-bezogenes Arbeiten:
append() anhängen eines Eintrags an die Liste (einspeichern)
pop() zum Ausspeichern (liefert das letzte Element zurück und entfernt es vom Ende der Liste) (!!! Fehler bei leerer Liste)
eine Liste wird als Konstrukt aus einer (kleineren) Liste und einem (letzten) Element - dem Ende - verstanden

Vielfach sind Listen als zweigeteilte Objekte in Programmiersprachen angelegt. Neben dem eigentlichen Daten-Objekt enthält ein Eintrag auch noch eine Referenz auf den nächsten Eintrag. Eine Referenz ist praktisch eine Adresse im Hauptspeicher.



Beim letzten Eintrag wird die Referenz auf NIL gesetzt. NIL steht dabei für "Not in List" bzw. "Not in line". Es repräsentiert einen Null-Wert.
In Python wurde eine abweichende Organisations-Form gewählt. Die Liste besteht nur aus Referenzen. Diese verweisen auf irgendwo gespeicherte Daten-Objekte.
Ganz genau informatisch betrachtet sind Listen in Python also Felder (Array's) von Zeigern (Pointern).
Der NIL-Wert kann auch als Speicher-Adresse verstanden werden.



In dieser ist praktisch - wie in einem Mülleimer- ganz viel Platz.

in Python haben wir keinen Zugriff auf den Link-Teil eines Eintrages

lediglich das soundsovielte Element einer Liste kann angesteuert werden, was dem temporären Setzen des Zeiger's (auf ebendiesen Eintrag) entspricht

Listen lassen sich mithilfe von verschiedenen Schleifen durchlaufen
z.B. mit Zählschleife bei bekannter Anzahl von Einträgen (Index-Zugriffe)
z.B. mit einer bedingten Schleife und einer Erkennung des letzten Eintrag's (→ NIL)
z.B. mit Sammlungs-bedingten Schleifen / (verdeckten) Interatoren

praktische Nutzungen / Abwandlungen in …:

- Warteschlange
  - Drucker-Warteschlange
  - Übertragungs-Puffer (z.B. TCP/IP; Tastatur-Eingaben)
  -
-

Operationen auf Listen (Attribut-Schreibweise)

Bezeichnung	Operation	Resultat
	Liste **= []**	erzeugen einer leeren Liste
	Liste**[i] =** Wert	i. Eintrag in der Liste (er)setzen / einspeichern
	Liste**[i : j] =** Teilliste	in der Liste wird der Abschnitt von Eintrag i bis j durch die angegebene Teilliste ersetzt
	del Liste**[i : j]**	löst in der Liste den Abschnitt von Eintrag i bis j (→ Verkürzung der Liste) äquivalent zu: Liste[i : j] = []
	Liste**.append(**Wert**)**	hängt Wert als neuen Eintrag an die Liste an äquivalent zu: Liste[len(Liste) : len(Liste)] = [Wert]
	Liste**.extend(**Wert**)**	äquivalent zu: Liste[len(Liste) : len(Liste)] = Wert
	Liste**.count(**Wert**)**	liefert die Anzahl der Einträge zurück, die Wert entsprechen return: Anzahl der i mit Liste[i] == Wert
	Liste**.index(**Wert**)**	liefert den ersten Index zurück, bei dem der Eintrag dem Wert entspricht return: Erstes i mit Liste[i] == Wert
	Liste**.insert(i,** Wert**)**	fügt einen neuen Eintrag mit dem Wert an Position i ein äquivalent zu: Liste[i : i] = Wert , wenn i >= 0
	Liste**.remove(**Wert**)**	entfernt das erste Auftreten eines Wertes ohne Kenntnis des Index äquivalent zu: del Liste[Liste.index(Wert)]
	Liste**.peek()**	liest den obersten Wert der Liste äquivalent zu: Liste[0]
	Liste**.pop()**	liest und entfernt das oberste Objekt aus der Liste äquivalent zu: del Liste[0]
	Liste**.reverse()**	Liste wird intern (in place) umgedreht

		(originale Reihenfolge durch erneutes reverse() wiederherstellbar)
	Liste**.sort()**	Liste wird intern (in place) sortiert (originale Reihenfolge der Einträge geht verloren!)
	Liste**.sort(**VergleichsFunktion**)**	return -1, 0, +1 ; wenn x<y, x=y, x>y
	Liste**.( )**	

Operationen auf Listen (Präfix-Schreibweise)

Bezeichnung	Operation	Resultat
	map(Funktion, Liste**)**	neue (temporäre) Liste: [Funktion(Liste[0], Funktion(Liste[1], …, Funktion(Liste[n]]
	filter(Bedingung, Liste**)**	neue (temporäre) Liste mit allen Einträgen aus der Liste, die die Bedingung erfüllen
	Liste1 **+** Liste2	neue (temporäre) Liste mit allen Einträgen aus Liste 1 und 2
	reduce(Funktion, Liste**)**	
	reduce(Funktion, Liste, init**)**	
	zip(Liste1, Liste2**)**	erzeugen einer (temporären) Paar-Liste mit zusammengehörenden Tupeln aus der Einträgen der beiden Listen [[Liste1[0], Liste2[0], [Liste1[1], Liste2[1], …, [Liste1[n], Liste2[n]]

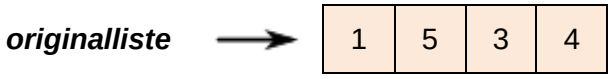
(c,p) 2015 - 2022 lsp: dre

8.2.3.1. Definition und Zuweisung von Listen in Python

Definieren wir uns zuerst einmal eine Liste mit dem Namen ***originalliste***. Sie soll bei den nächsten Versuchen immer wieder die Ausgangsbasis sein.

```
>>> originalliste=[1,5,3,4]
>>> print(originalliste)
[1, 5, 3, 4]
```

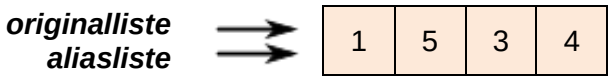
Der Name originalliste ist im Prinzip ein Verweis (bzw. ein Zeiger) auf die Speicher-Zellen, wo sich die Daten befinden.



Die nachfolgende Shell-Eingabe liest sich so, als würde man eine neue Liste – sozusagen eine Kopie vom Original erstellen.

```
>>> aliasliste=originalliste
>>> print(aliasliste)
[1, 5, 3, 4]
>>>
```

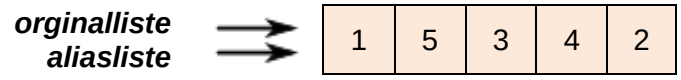
Praktisch wird aber nur ein zweiter Verweis auf die Originalliste gelegt.
Das merken wir spätestens, wenn wir eine der Listen verändern.



Einen Nebenverweis nennt man einen Alias bzw. einen Aliasnamen.

```
>>> aliasliste=aliasliste+[2]
>>> print(aliasliste)
[1, 5, 3, 4, 2]
>>> print(originalliste)
[1, 5, 3, 4, 2]
>>>
```

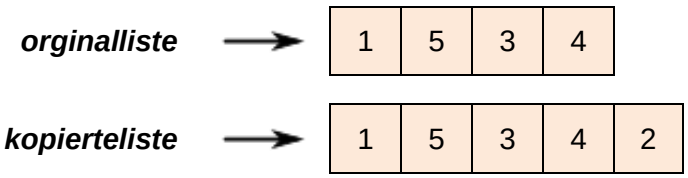
Beide Listen sind länger geworden. Ganz exakt müsste man eigentlich sagen, die (eine) Liste ist länger geworden



Das Aliasieren erzeugt nur eine sogenannte **flache Kopie** der Liste. Eine **tiefe Kopie** – wir würden wohl eher **echte Kopie** sagen – erhält man z.B. durch das Slicing (→ 8.2.3.5. Listen-Abschnitte (Slicing)).

```
>>> kopierteliste=originalliste[:]
>>> kopierteliste=kopierteliste+[7]
>>> print(originalliste)
[1, 5, 3, 4]
>>> print(kopierteliste)
[1, 5, 3, 4, 7]
>>>
```

Die Original-Liste bleibt bei Operationen auf die kopierte Liste unverändert.
Das Löschen der Original-Liste hat auch keine Auswirkungen, da beide Listen völlig eigenständig sind.



So zumindestens lautet die Theorie bzw. lauten die Aussagen vieler Buchautoren und der Python-Guru's aus dem Internet.
In meinen Test's mit einem Python-System (V. 3.4.3; 32 bit) sah das alles anders aus. Hier wird ganz offensichtlich eine echte (**tiefe**) **Kopie** der Liste erzeugt und auch durchgehend verwaltet. Beide Listen (originalliste und aliasliste) lassen sich unabhängig manipulieren und löschen.
Die zwei nachfolgenden Shell-Dialoge zeigen dieses ganz klar.

```
>>> originalliste=[1,5,3,4]
>>> print(originalliste)
[1, 5, 3, 4]
>>> aliasliste=originalliste
>>> print(originalliste)
[1, 5, 3, 4]
>>> print(aliasliste)
[1, 5, 3, 4]
>>> aliasliste=aliasliste+[3]
>>> print(originalliste)
[1, 5, 3, 4]
>>> print(aliasliste)
[1, 5, 3, 4, 3]
>>>
```

Erweiterung der Aliasliste

… und es wurde auch nur die Aliasliste erweitert

```
>>> originalliste=[1,5,3,4]
>>> print(originalliste)
[1, 5, 3, 4]
>>> aliasliste=originalliste
>>> print(aliasliste)
[1, 5, 3, 4]
>>> originalliste=originalliste+[3]
>>> print(originalliste)
[1, 5, 3, 4, 3]
>>> print(aliasliste)
[1, 5, 3, 4]
>>> del originalliste[2]
>>> print(originalliste)
[1, 5, 4, 3]
>>> print(aliasliste)
[1, 5, 3, 4]
>>> del originalliste
>>> print(originalliste)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    print(originalliste)
NameError: name 'originalliste' is not defined
>>> print(aliasliste)
[1, 5, 3, 4]
>>>
```

Erweiterung der Originalliste ...
… und auch nur diese wurde erweitert

Löschen des 3. Wertes in der Originalliste …
was offensichtlich klappt
… aber die Aliasliste davon unberührt lässt.
… genau so wie das Löschen der Originalliste

… die wirklich weg ist …

… aber die Aliasliste noch völlig im "Original"-Zustand vorhanden ist

Erst einmal bleibt uns jeweils wohl nur der Test am eigenem System!

Kehren wir zur offiziellen Version zurück und nehmen mein System als Ausrutscher!

Wie bekommen wir nun aber eine echte / eigenständige Kopie einer Liste? Eine kleine Variante haben wir oben schon gezeigt. Eine weitere Möglichkeit sind fertige Objekt-orientierte Funktionen zu Listen. Diese besprechen wir etwas später (→ 9.8. Listen, die II. – objektorientierte Listen). Eine Möglichkeit werden wir gleich bei der Listen-Bearbeitung, eine weitere nochmals genauer bei den Listen-Abschnitten (Slicing), besprechen. Python bietet mehrere (gut und weniger gut leserliche) Möglichkeiten – der Programmierer hat hier die freie Wahl. Man nennt dies auch Klonen von Listen.

8.2.3.2. Listen-Operationen (Built-in-Operatoren)

Für Listen funktionieren einige "Rechen"-Operationen im intuitiven Sinne. So ergibt sich bei Verwendung des Multiplikations-Zeichens ***** eine x-mal erweiterte / verlängerte Liste.

```
>>> liste=[2]
>>> liste=liste*5
>>> print(liste)
[2, 2, 2, 2, 2]
>>>
```

Mit **+** lassen sich Listen addieren, aneinander anhängen bzw. verbinden.

```
>>> liste=liste + liste
>>> print(liste)
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
>>>
```

Nicht wirklich eine Rechen-Operation steckt hinter dem **in**-Operator. Mit ihm können wir schnell testen, ob Etwas ein Element einer Liste ist. Als Ergebnis erhält man entweder **1** – für ist ***in*** der Liste – oder eben **0** (nicht ***in*** der Liste).
Der in-Operator lässt sich mit **not** in Verbindung nutzen

```
>>> liste=['gelb','grün','rot','weiß','gelb','braun','blau']
>>> print(liste)
['gelb', 'grün', 'rot', 'weiß', 'gelb', 'braun', 'blau']
>>> 'grün' in liste
True
>>> 'schwarz' in liste
False
>>> 2 in liste
False
>>>
```

Der in-Operator lässt sich mit **not** in Verbindung nutzen

```
>>> 'blau' not in liste
False
>>> 3 not in liste
True
>>>
```

Sehr praktisch sind die Funktionen **min()** und **max()**. Mit ihnen kann man ohne (eigenen) Programmieraufwand das kleinste bzw. größte Listenelement heraussuchen. Bei Texten gilt die alphabetische Ordnung, bei Zahlen die normale Rangfolge.

```
>>> liste=['rot', 'grün', 'gelb']
>>> print(min(liste1))
gelb
>>> print(max(liste1))
rot
>>>
```

Bei gemischten Listen gibt es eine Typ-Fehlermeldung.
An dieser Stelle hat die nachfolgende Funktion eigentlich keine Bedeutung. Ich erwähne sie hier wegen der Vollständigkeit und einem vielleicht benötigtem Hinweis. Mit der Funktion list() lassen sich Tupel (→) in Listen umwandeln. Das spielt immer dann eine Rolle, wenn die Daten in Tupeln vorliegen und bearbeitet werden sollen, was bei Tupeln eigentlich nicht geht.

```
>>> tupel1=(23,'gelb',['rot',2])
>>> print(tupel1)
(23, 'gelb', ['rot', 2])
>>> tupel1[0]=24
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    tupel1[0]=24
TypeError: 'tuple' object does not support item assignment
>>> liste1=list(tupel1)
>>> print(liste1)
[23, 'gelb', ['rot', 2]]
>>> liste1[0]=24
>>> print(liste1)
[24, 'gelb', ['rot', 2]]
>>>
```

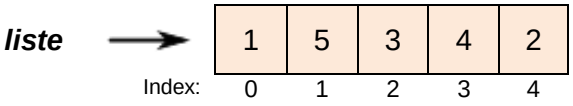
cmp(liste1, liste2**)**
vergleicht Listen (wahrscheinlich nur bis Python2)

Anhängen eines Elemntes bzw. einer Liste an eine andere ist auch über den Plus-Operator (**+**) möglich. Für interierende Aufgaben ist auch **+=** möglich. Bei der Verwendung sollte man aber beachten, dass diese Opratoren über 1'000x langsamer sind als z.B. die append()-Funktion.

8.2.3.3. Listen-Indexierung

Auf die einzelnen Elemente einer Liste kann man natürlich auch direkt zugreifen. Dazu muss man aber wissen, dass die Liste mit mit 0 beginnend gezählt wird. Die Zählung wird deshalb auch Indexierung genannt und der Zugriffswert Index. Das erste Listen-Element hat also den Index 0.

Der Zugriff wird in Python denkbar einfach ermöglicht. Wir müssen den Index nur an den Listennamen in eckigen Klammern angeben. Der Index kann beliebig berechnet werden.



Wichtig ist dabei nur, dass das Ergebnis immer ganzzahlig sein muss.

```
>>> liste=[1,5,3,4,2]
>>> print(liste)
[1, 5, 3, 4, 2]
>>> print(liste[2])
3
>>> print(liste[2*2-1])
4
>>> print(liste[5/2])
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    print(liste[5/2])
TypeError: list indices must be integers, not float
>>>
```

Aber das erscheint wohl logisch, denn z.B. ein 2,5tes Element gibt es eben nicht. Der Zugriff auf einen fehlerhaften oder zu großen Index ergibt eine Fehler-Meldung:

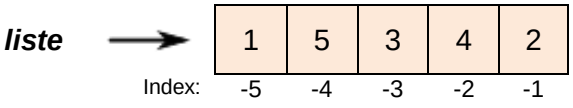
```
>>> print(liste[13])
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print(liste[13])
IndexError: list index out of range
>>>
```

Um hier fehlerfrei agieren zu können müssen wir natürlich wissen, wieviele Elemente in der Liste sind. Dafür gibt es die Funktion **len()**, die genau diese Aufgabe erfüllt.

```
>>> len(liste)
5
>>>
```

Wird ein negativer Index angegeben, dann wird in der Liste zurückgezählt:

Natürlich darf auch hier der Index-Wert nicht größer (gemeint ist natürlich: kleiner) als die Listenlänge werden.



Das Löschen einer ganzen Liste oder eines Elementes aus der Liste ist mit **del** möglich.

```
>>> originalliste=[1,5,3,4]
>>> del originalliste[2]
>>> print(originalliste)
[1, 5, 4]
>>>
```

where um Indizes anhand einer Bedingung auszuwählen

```
>>> i=0
>>> while i<len(farbenliste):
      print(farbenliste[i])
      i+=1


gelb
grün
rot
blau
braun
>>>
```

??? feld=array([[0,1,2],[3,4,5]])
??? feld[where(feld > 3)] where(feld > 3)

An die reine Indexierung schließt das sogenannte Slicing von Listen (→ 8.2.3.5. Listen-Abschnitte (Slicing)) an.

Zuweisen von Werten aus einer Liste auf Positionen, die in einer anderen Liste verwaltet werden

```
liste=[1,2,3,4,5,6,7,8,9,10]
indizies=[3,4,6]
neueWerte=[100,200,300]
```

Interieren über die Tupel (Paare) von Index und Wert

```
for i in zip(indizies, neueWerte):
    liste[i[0]]=i[1]

print(liste)
```

2. Möglichkeit
mit der eingebauten Funktion (Build In Funktion) enumerate über Listen interieren

```
for index,wert in enumerate(indizies):
    liste[wert]=neueWerte[index]

print(liste)
```

3. Möglichkeit
die Liste in ein (NumPy-)Array umwandeln und dann eine Zuweisung af der Basis der Indizies durchführen

```
import numpy as np

feld=np.array(liste)
feld[indizies]=neueWert
liste=list(feld)

print(liste)
```

zur Nutzung der Bibliothek NumPy gibt es weitere Informationen in einem gesonderten Kapitel (→ 8.14.3. Python numerisch, Python für Big Data)

8.2.3.4. Listen-Bearbeitung

Das Gute zuerst, Listen lassen sich bearbeiten. Das überrascht vielleicht etwas, nachdem wir bei den Strings (→ 8.2. Strings – Zeichenketten) und den Feldern (→ 6.6. Vektoren, Felder und Tabellen) dahingehend enttäuscht worden sind. Listen sind konzeptionell einfach die priviligierten Daten-Objekt in Python.

An dieser Stelle sei kurz auf eine den Listen sehr ähnliche Daten-Struktur hingewiesen – die Tupel. In Abschnitt 9.1. Tupel werden sie noch genauer besprochen. Prinzipiell ähneln sie den Listen, man kann fast alle Operationen usw. auf sie anwenden. Allerdings sind Tupel feststehende Objekte. Nach ihrer Erzeugung lassen sie sich nicht mehr ändern!

Am Einfachsten lassen sich Listen mittels Schleifen bearbeiten. Dabei bieten sich die **for**-Schleifen sofort an.

Wollen wir die Listenelemente einzeln ausgeben, dann reicht schon der folgende – leicht verständliche – Konstrukt:

```
>>> farbenliste=['gelb','grün','rot','blau','braun']
>>> for farbe in farbenliste:
      print(farbe)


gelb
grün
rot
blau
braun
>>>
```

Es geht natürlich genauso mit einer **while**-Schleife, aber hier müssen wir die Indizierung selbst verwalten.
Für die gleiche Beispiel-Aufgabe – also die Element-weise einer Liste – würde der Quelltext dann z.B. so aussehen:

```
>>> i=0
>>> while i<len(farbenliste):
      print(farbenliste[i])
      i+=1


gelb
grün
rot
blau
braun
>>>
```

Natürlich hätte man hier auch z.B. die Lauf-Variable farbe benutzen können, wie in der for-Schleife z.B. ein einfaches i.
Insgesamt ist der Programmier-Aufwand etwas größer. Welchen Schleifen-Konstrukt man dann in seinem Programm später verwendet, entscheidet sich wahrscheinlich neben der Vorliebe auch aus praktischen Abwägungen.
Um nun eine Liste zu klonen – also eine echte Kopie zu erzeugen – bietet sich der folgende Algorithmus an:

```
>>> neueListe=[]
>>> for f in farbenliste:
      neueListe=neueListe+[f]


>>> print(neueListe)
['gelb', 'grün', 'rot', 'blau', 'braun']
>>>
```

Es wird praktisch jedes einzelne Element aus der Originalliste ausgelesen und in die Kopie geschrieben.
Nachteilig ist es in vielen Programmiersprachen, wenn man lange Listen oder Felder (→) mittels Schleifen-Konstrukt durchlaufen muss, um z.B. eine Summe zu berechnen oder das Minimum herauszusuchen. Da bietet uns Python die map-Funktion, um beliebige mathematische Funktionen auf die Elemente einer Liste anzuwenden.

map(funktion, liste**)**

Die Funktion kann eine von Python vordefinierte oder eine selbst-definierte Funktion sein. Im map-Funktions-Aufruf wird die Funktion nur mit ihrem Namen (also ohne Klammern und Argumenten) notiert. Die Liste muss eine Sequenz sein, auf deren Elemente die Funktion angewendet werden kann.
Definieren wir zuerst eine Funktion, die den übergebenen Wert mit 10 multipliziert und dann noch inkrementiert. Die Operationen sind mit Absicht in einzelnen Schritten aufgeschrieben worden. Natürlich können sie in einer Zeile oder gar hinter **return** zusammengefasst werden.

```
>>> def mult10add1(x):
        x*=10
        x+=1
        return x
>>>
```

Die klassische Struktur einer Element-weisen Anwendung einer Funktion kann z.B. wie nebenstehend aussehen.
Wir definieren eine nutzbare Funktion. Alternativ kann auch jede interne Funktion genutzt werden.
In einer Schleife wird diese Funktion dann Element-weise angewendet und das Ergebnis an die ursprüngliche Listen-Position gespeichert.

```
def mult10add1(x):
    x*=10
    x+=1
    return x
…
liste=[2,3,5,8,11]
for i in range(len(liste)):
    liste[i]=mult10add1(liste[i])
print(liste)
```

```
[21, 31, 51, 81, 111]
```

```
liste = map(mult10add1, liste)
print(liste)
```

Der **map**-Operator ergibt im Allgemeinen kompaktere Quell-Code's. Auch die Ausführung ist effektiver. Aus meiner Sicht sind die Quell-Texte aber nicht gut lesbar, da geschachtelte Strukturen entstehen, die immer wieder mit zurück-denken oder eine Ebene höher-denken zu tun haben. Map-Konstrukte sollten immer gut kommentiert werden! Sie sind etwas für fortgeschrittene Programmierer / Programmier-Team's. Für kompexere Funktionen können vor allem einzeilig notierte map-Funktionen praktisch unlesbar werden und sind deshalb unbedingt zu vermeiden.

List-Comprehension

In der letzten Zeit werden auch sogenannte List-Comprehension's immer mehr in Programmiersprachen umgesetzt. Das gilt auch für Python.
Comprehension's kann man als Bedeutungs- oder Anwendungs-Objekte oder -Strukturen verstehen. Ziel ist eine bessere Lesbarkeit von Quelltexten, verbunden mit einer hohen Kompaktheit des Quelltextes.
Soll z.B. aus einer Daten-Liste eine Liste mit Quadraten erzeugt werden, dann würde man eine Sammlungs-orientierte FOR-Schleife benutzen. Dies ist im oberen Quellcode-Block notiert.
Eine Comprehension-Struktur könnte z.B. so aussehen, wie es im 2. Block zu sehen ist.
Die Struktur besteht aus der Listen-Beginn-Klammer ([) gefolgt von der Operation (auch Expression genannt) und der FOR-Schleife. Die Expression ist in Beispiel die Multiplikation von Wert mit sich selbst. Das Ende wird durch die schließende Listen-Klammer (])gekennzeichnet.

```
daten=[1,2,3,4,5]


quadrate=[]
for wert in daten:
    quadrate.append(wert*wert)
```

```
quadrate=[wert*wert for wert in daten]
```

```
quadrate=[wert*wert for wert in daten ↵
          if wert > 0]
```

```
quadrate=[wert*wert if wert > 0 else 0 ↵
          for wert in daten]
```

Die Ausführung der Expression / Operation lässt sich noch durch Bedingungen steuern. Eine ausschließliche IF-Bedingung (einfache Bedingung) wird hinter den Schleifen-Konstrukt geschrieben. Der 3. Quellcode-Block zeigt ein Beispiel, dass hier zum gleichen Ergebnis führt, wie die beiden oberen Quellcode-Abschnitte.
Benötigt man auch einen ELSE-Zweig, dann folgt der angepasste IF-ELSE-Konstrukt direkt hinter der Expression und noch vor dem Schleifen-Teil.

Mit dem List Comprehension lassen sich auch Listen filtern.

```
[wert for wert in liste if wert>20]
```

```
[wert for wert in liste if (wert>20) and (wert<100)]
```

```
[True if wert==1 else False for wert in daten]
```

8.2.3.5. Listen-Abschnitte (Slicing)

Der Zugriff auf Abschnitte einer Liste wird durch die sogenannte Slice-Notation (Doppelpunkt-Notation) erheblich erleichtert. Für Python-Einsteiger oder Umsteiger aus anderen "normalen" Programmiersprachen werden diese Konstrukte aber erst einmal schwer zu verstehen sein. Zur Verdeutlichung nehmen wir eine etwas längere Liste mit etwas größeren Zahlen:

liste →	111	222	333	444	555	666	777	888	999
Index:	0	1	2	3	4	5	6	7	8
Item:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Benötigt man nur die Liste bis zu einem bestimmten Index, dann wird der Abschnitt mit [: endeindex] notiert.

```
>>> liste=[111,222,333,444,555,666,777,888,999]
>>> sliceliste=liste[:3]
>>> print(sliceliste)
[111, 222, 333]
>>>
```

Es sind mit **[:3]** also die ersten 3 Listen-Elemente (Items) gemeint, der Doppelpunkt steht also hier für "bis" zum dritten (3.) Element. Da die Indizierung bei Null startet sind es also die Index-Elemente 0, 1 und 2.

	111	222	333	444	555	666	777	888	999
Index:	0	1	2	**3**	4	5	6	7	8
Item:	**1**	**2**	**3**	4	5	6	7	8	9

Wird dagegen nur ein Index vor dem Doppelpunkt (Slice) eingegeben, dann meint man den Abschnitt nach diesem indizierten Element bis zum Ende der Liste.

```
>>> sliceliste=liste[5:]
>>> print(sliceliste)
[666, 777, 888, 999]
>>>
```

Mittels **[5:]** wird also es sind die Listen-Elemente nach Item 5 (bzw. ab Index = 5) bearbeitet.

liste →	111	222	333	444	555	666	777	888	999
Index:	0	1	2	3	4	**5**	6	7	8
Item:	1	2	3	4	**5**	**6**	**7**	**8**	**9**

Natürlich dürfen zur Auswahl eines Mittelstücks aus einer Liste auch vordere und hintere Grenze angegeben werden.

```
>>> print(liste[2:7])
[333, 444, 555, 666, 777]
>>>
```

Mit **[2:7]** meint man dann den Abschnitt ab Index = 2 bis an den 7. Index ran.
es sind die Elemente nach dem 3. (also ab dem zweiten (2.)) bis zum 5. gemeint

liste →	111	222	333	444	555	666	777	888	999
Index:	0	1	**2**	3	4	5	6	**7**	8
Item:	1	**2**	3	4	5	6	**7**	8	9

Somit gilt also allgemein: abschnitt = liste[anfangsindex : endeindex].
Interessanterweise funktioniert das Slicen auch zum Einfügen eines Listen-Abschnitts:
Die Notierung wäre also: liste[anfangsindex : endeindex] = abschnitt

```
>>> print(sliceliste)
[666, 777, 888, 999]
>>> liste[6:7]=sliceliste
>>> print(liste)
[111, 222, 333, 444, 555, 666, 666, 777, 888, 999, 888, 999]
>>>
```

Die eingeslicte Liste wird zuerst noch einmal angezeigt (geprintet) und ist dann in der Ergebnis-Liste farblich unterlegt.

8.2.3.6. Listen-Erzeugung – fast automatisch

mit **range()** werden Listen automatisch erzeugt

range(ende**)**
erzeugt Liste von 0 bis (ausschließlich) ende

range(anfang, ende**)**
erzeugt Liste von anfang bis (ausschließlich) ende

range(anfang, ende, schrittweite**)**
erzeugt Liste von anfang bis (ausschließlich) ende mit der schrittweite (also: anfang + n * schrittweite)

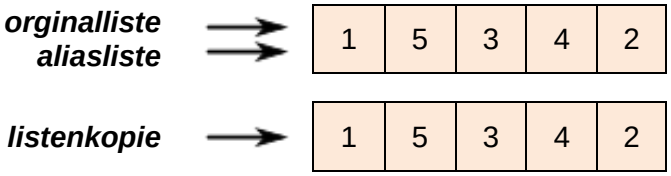
Listen können auch wieder Listen enthalten (Verschachtelung, Nesting)
so lassen sich Matrizen darstellen und bearbeiten. da das aber eher für mahematisch Fortgeschrittene interessant wird, folgen dazu in Kapitel → 8.13.2. Matrizen (Matrixes) mehr Informationen

Zugriff für aneinander-gereihte Index-Operatoren
Hierbei ist besonders auf die Gültigkeit der Indexes zu achten

Eine andere Möglichkeit zum Listen-Klonen (Kopieren einer Liste) ergibt sich aus der Slice-Notierung. Die neue Liste soll den Namen ***listenkopie*** bekommen.

```
>>> listenkopie=originalliste[:]
>>> print(listenkopie)
[1, 5, 3, 4, 2]
>>>
```

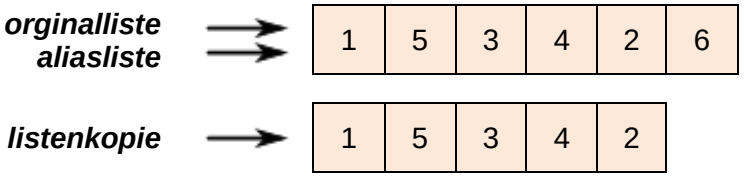
Der Doppelpunkt in der Slice-Notierung zieht eine Grenze. Da aber weder davor eine Anzahl Elemente, noch danach eine Anzahl angegeben wurde, handelt es sich um die gesamte Liste.



Zum Überprüfen, dass es sich bei der Kopie wirklich um einen neue / eigenständige Liste handelt, verändern wir sie durch Hinzufügen eines weiteren Elementes:

```
>>> aliasliste=aliasliste+[6]
>>> print(aliasliste)
[1, 5, 3, 4, 2, 6]
>>> print(listenkopie)
[1, 5, 3, 4, 2]
>>>
```

Die Situation im Speicher kann man sich etwa so vorstellen:



.

8.2.3.7. Listen - extravagant

Der Umgang mit Listen hält noch weitere Besonderheiten / Überraschungen bereit.

erweitertes Listen-Generieren

liste = [x for x in range(20) if x % 2]
erzeugt eine Liste der ungeraden Zahlen (von 0) bis an 20 ran

liste = [(x,y) for x in range(10) if not x % 3 for y in range(6) if y % 2]
erzeugt eine Liste aus Tupel, bei denen x die durch 3 teilbaren Zahlen (von 0) bis an 10 ran und y die ungeraden Zahlen (von 0) bis an 6 ran verwendet werden

gemeinsame Elemente zweier Liste in eine neue Liste
liste1=[1,2,3,4]
liste2=[3,4,5,6]
liste=[i for i in liste1 if i in list2]

aus zwei Datenlisten eine Liste aus Tupeln zusammenstellen
liste1=[1,2,3,4]
liste2=['a','b','c','d']
tupelliste=[(i,j) for i in liste1 for j in liste2]

erweitertes Slicing

text = 'abcdefg'
print(text[1:6:2]) ➔ 'bdf'
vom ersten bis zum sechsten Element (Achtung es geht immer noch bei 0 los!) jedes zweite Element

print(text[::-1]) ➔ 'gfedcba'

print(range(10)[::2]) ➔ [0, 2, 4, 6, 8] (entspricht: range(0,10,2) (besser verständlich)
print(range(10)[::-1]) ➔ [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] (entspricht: range(9,-1,-1))

Aufgaben:

x. Die chemischen Elemente lassen sich in verschiedene Gruppen einteilen. Dazu gehören die Hauptgruppen und die Metalle und Nichtmetalle. Schreiben Sie ein Programm, dass ein einzugebenes chemisches Symbol – z.B. Na – einer Hauptgruppe (mit Namen) zuordnet!
Erweitern Sie das Programm dann um die Zuordnung in die Perioden sowie zu den Metallen, Nichtmetallen bzw. Halbmetallen!
Zusatz:
Unterscheiden Sie die Elemente auch nach ihrem Aggregatzustand! Versuchen Sie möglichst kleine Listen (Datenbasen) zu verwenden!

Gruppe	Vertreter
I. Hauptgruppe, Alkalimetalle	H, Li, Na, K, Rb, Cs, Fr
II. Hauptgruppe, Erdalkalimetalle	Be, Mg, Ca, Sr, Ba, Ra
III. Hauptgruppe, Bor-Gruppe, Erdmetalle	B, Al, Ga, In, Tl
IV. Hauptgruppe, Cohlenstoff-(Silicium-)Gruppe, Tetrele	C, Si, Ge, Sn, Pb
V. Hauptgruppe, Stickstoff-(Phosphor-)Gruppe, Pnictogene	N, P, As, Sb, Bi
VI. Hauptgruppe, Chalkogene, Erzbildner, Sauerstoff-Gruppe	O, S, Se, Te, Po
VII. Hauptgruppe, Halogene, Fluor-Gruppe, Salzbildner	F, Cl, Br, I, At
VIII. Hauptgruppe, Edelgase, Helium-Gruppe	He, Ne, Ar, Kr, Xe, Rn

(c,p) 2015 - 2022 lsp: dre

kleine Zusammenfassung (Indexierung / Slicing)

allgemeine Struktur	**Funktion / Leistung** ***Umschreibung***	**Beispiel-Liste: liste=[1,2,3,4,5,10,11,12,13,14,15]**	
		Beispiel	**Ergebnis**
liste[i]	liefert den Eintrag von Index ***i*** **(Achtung! Zählung / Index beginnt bei 0)** *Welcher Wert steht an Index-Position i?*	liste[2] liste[0] liste[6]	3 1 11
liste[-1]	liefert das letzte Elem. zurück *Welcher Wert steht an der letzten Index-Position?* *Welcher Wert steht an 1. von hinten gezählten Index-Position?* **(Achtung! Zählung / Index beginnt hier mit -1,** **da es ein -0 natürlich nicht gibt)**	liste[-1]	15
liste[-i]	liefert den ***i***-ten Eintrag ***von hinten*** *Welcher Wert steht an der zurück gezählten Index-Position i?*	liste[-3] liste[-10]	13 2
liste[:]	liefert alle Elemente (ohne Begrenzungen) einer Liste als neue Liste *Welche Werte stehen in der Liste (ohne Begrenzungen durch Indizies)?*	liste[:]	[1,2,3,4,5,10,11,12,13,14,15]
liste[nach:]	liefert die Elemente ***nach*** Position bzw. ab Index ***nach*** als neue Liste *Welche Werte folgen ab der Index-Position?*	liste[2:] liste[6:] liste[0:]	[3,4,5,10,11,12,13,14,15] [12,13,14,15] [1,2,3,4,5,10,11,12,13,14,15]
liste[:bis]	liefert die Elemente ***bis*** Position bzw. kleiner dem Index ***bis*** als neue Liste *Welche Werte stehen vor der Index-Position?*	liste[:2] liste[:7] liste[:-1]	[1,2] [1,2,3,4,5,10,11,12] [1,2,3,4,5,10,11,12,13,14]
liste[nach:bis]	liefert als neue Liste die Elemente ***nach*** Position bzw. ab Index ***nach*** und ***bis*** Position bzw. kleiner dem Index ***bis*** *Welcher Wert stehen zwischen den Index-Positionen?*	liste[1:4] liste[4:8] list[3:-1]	[2,3,4] [5,10,11,12,13] [4,5,10,11,12,13,14]
liste[::sprung]	liefert eine neue Liste der Einträge einer originalen Liste, die nacheinander mit einem ***sprung*** erreicht werden (beginnend mit Index 0) *Welcher Wert steht an Index-Position i?*	liste[::3] liste[::-2] liste[::0]	[1,4,12,15] [15,13,11,4,2] → Fehler
liste[nach::]	wie liste[nach:] bzw. liste[nach::sprung]	liste[2::]	[3,4,5,10,11,12,13,14,15]
liste[nach::sprung]	liefert die Einträge, die ***nach*** dann nacheinander mit einem ***sprung*** erreicht werden *Welche Werte folgen ab der Index-Position in bestimmten Schritten?*	liste[4::2] liste[2::4] liste[0::1]	[5,12,14] [3,12] [1,2,3,4,5,10,11,12,13,14,15]

allgemeine Struktur	Funktion / Leistung *Umschreibung*	Beispiel-Liste: listeA=[1,2,3,4,5,10,11,12,13,14,15]	
		Beispiel	**Ergebnis**
listeA[nach:]= listeB[von:bis]	ersetzt in der A-Liste die Elemente ab ***nach*** durch die Elemente aus der B-Liste (hier ein Ausschnitt)	liste[2:]= liste[2:7] liste[4:]=	
listeA[:bis]= listeB[von:bis]	ersetzt in der A-Liste die Elemente vor ***bis*** durch die Elemente aus der B-Liste (hier ein Ausschnitt)	liste[:2]	
listeA[vor:nach]= listeB[von:bis]	fügt in die A-Liste in die Position zwischen ***vor*** und ***nach*** ersetzend die B-Liste (hier ein Ausschnitt) ein	liste[3:4]= liste[2:7] liste[4:7]= liste	[1,2,3,3,4,5,10,11,12,5,10, 11,12,13,14,15]

Aufgaben:

1. *Erstellen Sie sich eine Liste mit den ersten 9 Buchstaben (als Zeichen) unter dem Namen liste!*
2. *Überlegen Sie sich (ohne Python!)) für alle Beispiele aus der Zusammenfassungs-Tabelle oben die Ergebnisse!*
3. *Prüfen Sie nun mit Python!*
4. *Erstellen Sie sich eine Liste aller Groß-Buchstaben als Liste von Zeichen mit dem Namen buchstaben!*
5. *Geben Sie für die nachfolgenden Problemstellungen eine Listen-Formulierung an!*
   a)
6. *Prüfen Sie nun mit Python!*

8.2.3.8. Ringe – geschlossene Listen

Ringe sind in Python als solche nicht vorgesehen. Da man aber auch mit negativen Indizies arbeiten kann, hat man es praktisch bei Listen mit Ringen zu tun. Die Ring-Größe wird durch die Länge der Liste bestimmt. Die jeweilige Schreib- oder Lese-Position wird durch den aktuellen Index der innerhalb der Listen-Länge inkrementiert oder dekrementiert werden kann. Alternativ bietet sich auch eine Überwachung über die Modulo-Operation an.

Aufgaben:

1. *Erstellen Sie sich ein Programm, dass in einer einzugebenen Liste aus Elementen jeweils die Liste und die aktuelle Index-Postion (Zeiger) anzeigt (z.B. als "I" unter der Liste, s.a. folgendes Beispiel)!*

```
Liste/Ring       :    1    2    3    4    5    6    7    8
akt. Zeiger(=  2):              I
```

2. *Erweitern Sie nun das Programm von 1. so, dass der Nutzer (in einer Schleife) angeben kann, um wieviele Positionen sich der Zeiger verschieben soll! Die Anzeige soll wieder die Liste und die aktuelle Zeiger-Postionen sein!*
3.

8.2.3. Dictionarys - Wörterbücher

Wenn man es ganz genau nimmt, dann sind Dictionarys eigentlich eher Vokabel-Listen oder Daten-Paare. Ein Daten-Paar besteht immer aus einem "Schlüssel" – also dem beschreibenden Begriff – und einem zugeordneten Daten-Element. Als Daten dürfen die verschiedenen schon besprochenen Datentypen fungieren, sowie deren Verknüpfungen in Tupel (→), Mengen (→) und Vektoren (→).
Wir sehen hier schon, dass Dictonary's auch so einiges mit Listen gemeinsam haben. In der Informatik werden die Dictonary's aber eher als sehr lockere Liste besser Sammlung von Daten-Paaren gesehen.
Dictonary's werden einfach als eine spezielle Datenstruktur definiert. Wenn man etwas mit einfachen Listen machen will (→ 8.2.3. Listen, die I. – einfache Listen), dann nutzt man auch die Datenstruktur Liste. Stehen díe Daten-Paare im Vordergrund und die Auflistung ist zweitrangig, dann sind Dictonary's eine mögliche Wahl.
Eine etwas ausführlichere Besprechnung erfolgt im Kapitel → 9.3. Dictonary's - Wörterbücher. Hier gehen wir auf einfache Nutzungen ein.

Definition(en): Dictonary
Im informatischen Sinn versteht man unter der Datenstruktur Dictonary eine Listen-artige Sammlung von Daten-Paaren.

typische Nutzung z.B. Vokabel-Wörterbücher

allerdings keine gleichberechtigten Paare von Wörtern, sondern eine einseitig gerichtete Zuordnung von Daten-Elementen.
die linke Seite (quasi das erste Wort) ist der Schlüssel (engl. key), dieser muss im Wörterbuch eindeutig sein, d.h. er dar nur ein einziges Mal vorkommen
jedem Schlüssel wird dann noch ein Wert zugeordnet. Dieser darf mehrfach im Werte-Bereich vorkommen.
zusammen sprechen wir von Schlüssel-Wert-Paaren (key-value-Paare)

```
vokabel = {
      "Stadt": "City",
      "U-Bahn": "subway",
      "gelb": "yellow"
}
print(vokabel)
```

Reihenfolge ist nicht durch Notieren im Quell-Code oder nach einem Einlesen festgelegt
die Reihenfolge kann sich leicht ändern

```
len(vokabel)
```

gibt die Anzahl der Schlüssel-Wert-Paare zurück
Zugriff ähnlich wie Listen, nur dass hier statt einem Index der Schlüssel verwendet wird

```
print(vokabel["gelb"])
```

daraus abgeleitet erfolgt der ändernde Zugriff mit:

```
vokabel["U-Bahn"]="tube"
```

wird mit einem unbekannten Schlüssel gearbeitet, dann gibt es keinen Fehler, sondern es wird ein neuer Eintrag in das Wörterbuch aufgenommen

```
vokabel["orange"]="orange"
```

mit

```
del(vokabel["gelb"])
```

wird der gesamte Eintrag zum Schlüssel "gelb" gelöscht
zum Interieren über ein Wörterbuch kann man auf die Schlüssel-Liste zugreifen

```
for schluessel in vokabel.keys():
    print(schluessel)
```

genauso kann man auch über die Werte eines Wörterbuch's interieren:

```
for wert in vokabel.values():
    print(wert)
```

will man über die Schlüssel-Wert-Paare interieren, dann geht das über

```
for eintrag in vokabel.items():
    print(eintrag)
    print("deutsch: ",eintrag[0],"    heisst englisch" ",eintrag[1])
```

die Null steht dabei für den Schlüssel und die Eins für den Wert eines Eintrag's

auch gut geeignet um einfache Statistiken zu führen
z.B. Wort-Häufigkeiten

hier Beispiel zum Zählen von Farben in einer Liste

```
arbeitsListe = ["gelb","blau","blau","rot","blau","gelb","blau",
                "gelb","blau"]
haeufigkeit={
    "gelb": 0,
    "rot": 0,
    "blau": 0}

for elem in arbeitsListe:
    haeufigkeit(elem) += 1

print("aktuelle Häufigkeiten:")
print(haeufigkeit)
```

Aufgaben:

1. *Bauen Sie das Programm zur Analyse von Farb-Häufigkeiten in einer Liste so um, dass es auch Farben zählt, die nicht im Häufigkeits-Dictonary enthalten sind erfasst! Hierfür soll es im Ditonary ein Schlüsselwort "Reste" geben.*
2. *Bauen Sie das Programm zur Analyse von Farb-Häufigkeiten in einer Liste so um, dass es auch Farben zählt, die nicht im Häufigkeits-Dictonary enthalten sind erfasst! Neue Farben sollen als neue Einträge in das Dictonary hinzugefügt werden.*
3. *Erstellen Sie ein Programm, dass einen Text, der in Listen-Form vorliegt, hinsichtlich der enthaltenen Wörter statistisch analysiert! Neben der Wort-Häufigkeit, soll auch die Anzahl der Wörter insgesamt sowie die Anzahl unterschiedlicher Worte im Dictonary gespeichert werden!*

(c,p) 2015 - 2022 lsp: dre

8.4. Interation oder Rekursion? – das ist hier die Frage!

Die Frage, die wir uns hier stellen müssen, ist die nach dem besten Vorgehen beim Lösen eines Problems. Eine Variante wäre es ein Problem zuerst einmal auf ein oder mehrere einfachere Probleme zurückzuführen. Das macht man solange, bis es kein einfacheres Problem mehr gibt oder die Lösung offensichtlich ist. Auf dem Rückweg zum ehemaligen aufrufenden (großen) Problem ergänzt man die primitive Lösung immer ein Stück weiter.
Glaubt man der Literatur, dann ist dieses Lösungs-Verfahren, welches von Menschen und Programmierern (auch das sollen Menschen sein?!), am häufigsten / vorrangig genutzt wird.
Die andere Variante ist das gleichartige Wiederholen einer bekannten / einfachen Lösung bzw. einer Teil-Tätigkeit, bis die Aufgabe gelöst ist.
Meiner Meinung nutzen Menschen eher diese Methode. Bei Personen, die Programmieren lernten ist es ebenfalls die zuerst gewählte Lösungs-Strategie.
Praktisch ist es wohl eine nicht-entscheidbare Frage – wie die, was denn nun zuerst da war, das Huhn oder das Ei. Zum Einen lassen sich Probleme fast immer mit beiden Strategien lösen. Dabei ist meist die eine Strategie eleganter / effektiver / cleverer / schöner / …, aber das steht nicht Disposition.
Zum Anderen gibt es sie nicht – die universell beste Strategie, sonst könnten wir sie ja einfach ansagen / lehren / predigen. Vielfach hängt das beste Vorgehen von den Rahmen-Bedingungen ab, die zur Verfügung stehen. Im Computer-Bereich sind dies z.B. Speicherplatz oder die Rechen-Zeit.
Meist geht es bei Interation und Rekursion auch begrifflich etwas hin und her. Den die Interation oder die Rekursion gibt es nicht. Es sind verallgemeinerte Strategien.
Praktisch müsste man zwischen der interativen und / oder rekursiven Defintion einer Funktion und der programmiertechnischen Implementierung unterscheiden.
i.A. lassen sich die – wie auch immer definierten – Funktionen auf beide Arten implementieren; allerdings gibt es ohne weiteres Programmier-System, die bestimmte Strategien bevorzugen bzw. manche andere gar ausschließen.
Vielfach entscheidet der Programmierer, was günstiger ist.
Da beide Umsetzungen Vor- und Nachteile haben, müssen die System-Bedingungen aber mit beachtet werden

Die Suchmaschine google zeigt nach der Eingabe eines Suchbegriffes gleich unter der Trefferzahl und er Bearbeitungszeit oft auch ein "Meinst du: *XYZ*". Dabei werden vorrangig kleine Schreibfehler "korrigiert" oder alternative Begriffe angeboten. Sucht man nun auf der deutschen google-Seite nach Rekursion, dann ist das Ergebnis schon etwas überraschend. Auch auf der englischsprachigen Seite passiert mit dem Begriff "recursion" das Gleiche. Ist google hier ein Fehler unterlaufen? Wie unterscheiden sich die – und weitere alternative - Antwortseiten?

8.4.1. Interation

Denken wir z.B. an die Aufgabe eine 10 Kisten mit Wasserflaschen in der dritten Stock zu transportieren. Für einen echten Body-Builder kein Problem. Er weiss bloß nicht, was er in die andere Hand nehmen soll (;-).
Jeder würde diese Aufgabe sicher dadurch lösen, dass er kleinere Mengen (wahrscheinlich immer 2 Kästen) nach oben bringt. Die komplizierte (schwer zu lösende) Aufgabe wurde in mehrere gleiche Teil-Aufgaben zerlegt.

Teilaufgabe
Teilaufgabe
Teilaufgabe
Teilaufgabe
Teilaufgabe

Ein solches Problem-Lösen nennen wir **interatives Vorgehen**.
Aus informatischer Sicht ist das die Wiederholung strukturgleicher Blöcke mit Teilaufgaben. Wenn wir irgendwelche Dinge – z.B. eine Ausgabe x-mal wiederholen wollten, dann haben wir das in einer Schleife erledigt. Das ist eine klassische interative Lösung. Wir hätten auch ein Programm schreiben können, dass zumindestens für eine bekannte Anzahl von Wiederholungen, genau die gleiche Ausgabe in einem Stück erzeugt hätte. Da würden wir uns entweder die Finger wund tippen oder x-mal die Copy-und-Paste-Strategie anwenden müssen.
Alle Schleifen stellen typische Interationen dar. Der Wortstamm kommt auch vom latainischen *interare* für wiederholen.

Definition(en): Interation
Unter Interation versteht man das mehrfache (abzählbare / gezählte) Wiederholen einer Aktion / Handlung / Anweisung.
Interation ist die Anwendung immer gleicher Prozesse auf bereits gewonnene Zwischen-Ergebnisse.

Vorteile einer / der Interation

- **weniger Speicher-Bedarf**
- **intuitiv verständlich**
- **im direkten Vergleich meist schneller** meist sogar deutlich schneller
-

Nachteile einer / der Interation

- **kompliziertere Umsetzung**
- **längere Programmtexte**
-

8.4.1.1. typische Interations-Anwendungen

Eigentlich könnte ich mir diesen Abschnitt sparen, da die bisher besprochenen Wiederholungen fast ausnahmslos Interationen waren.
Da aber Summen und Produkte und vor allem deren Entwicklung in Schleifen zu den klassischen Programmier-Aufgaben gehören, seien sie hier noch mal aufgeführt, wiederholt und zum systematischen Verständnis dargestellt.
Wem die Summen- und Produkt-Bildung schon zur Nase raushängt und die Schwierigkeit damit nicht verstehen kann, der sollte gleich zu den Rekursionen (→ 8.4.2. Rekursion) übergehen. Da erwartet ihn vielleicht Neueres und Spannendes.

8.4.1.1.1. Summen-Bildung

```
def summe(endzahl):
    sum=0
    for i in range(1,endzahl+1):
        sum=sum+i
    return sum

# main
endzahl=eval(input("Bis zu welcher Zahl soll summiert werden?: "))
print("Die Summe lautet: ",summe(endzahl))
```

Wie sieht die Speicher-Belegung zum Zeitpunkt des Eintritts in die Zählschleife aus?
Ein Speicherzelle "endzahl" wurde mit der Eingabezeile angelegt und mit der Nutzer-Eingabe (hier: 10) gefüllt.
Beim Aufruf der Funktion summe wird nun eine Kopie dieser Speicherzelle angelegt, die aber nur innerhalb der summe-Funktion gültig ist. Gleiches gilt für die anderen Variablen.

summe()	i	1
summe()	sum	0
summe()	endzahl	10
	endzahl	10
	Name	***Speicher***

Man kann die Unabhängigkeit von endzahl gut testen, indem man z.B. innerhalb der Funktion die endzahl (vielleicht direkt vor dem return) ändert. Eine Ausgabe von endzahl im Hauptprogramm liefert die eingegebene Zahl. Mit dem return werden alle Variablen der Funktion summe gelöscht.
Auch davon kann man sich durch eine versuchte Ausgabe der summe-Funktions-Variablen im Hauptprogramm überzeugen. Es gibt eine Fehlermeldung.
Beim ersten Schleifen-Durchlauf ist i gleich 1 und wird in der Summierungszeile zuerst einmal (rechte Seite des Terms) auf den (alten) Inhalt von sum aufaddiert. Das Berechnungs-Ergebnis wird dann in der Speicherzelle sum (quasi als neue Belegung) gespeichert.

summe()	i	1
summe()	sum	1
summe()	endzahl	10
	endzahl	10
	Name	***Speicher***

Eigentlich würden wir in Python die Aufsummierung ja eher so schreiben: sum+=i. Das macht den Ablauf der inneren Speicher-Abläufe aber nicht nachvollziehbar.

summe()	i	1

summe()	sum	1
summe()	endzahl	10
	endzahl	10
	Name	***Speicher***

Mit dem Erreichen der letzten Schleifen-Anweisung (hier haben wir ja nur eine) wird i um Eins erhöht und geprüft, ob die Schleife ein nächstes Mal durchlaufen werden muss (i ist jetzt noch kleiner als endzahl+1).
Am Ende aller Schleifendurchläufe ist sum mit 55 belegt. Dieser Wert wird nun an die print-Anweisung übergeben. Natürlich hätte man auch eine andere Variable zur Übernahme des Funktionswertes nutzen können.

summe()	i	11
summe()	sum	55
summe()	endzahl	10
	endzahl	10
	Name	***Speicher***

Die gesamte Variablen-Struktur der Funktion wird nach dem return gelöscht und ist nicht wieder erreichbar. Nur bei speziellen Generator- Funktionen (→ 6.5.3. Generator-Funktionen – Funktionswerte schrittweise) bleibt die Variablen-Struktur für einen erneutetn Funktionsaufruf erhalten.

8.4.1.1.2. Produkt-Bildung

Die algorithmischen Änderungen zur Summe-Funktion sind minimal. Natürlich sollten die Bezeichner usw. angepasst werden. Aber für ein schnelles Test-Programm würde es auch ohne gehen.

```
def produkt(endzahl):
    prod=1
    for i in range(1,endzahl+1):
        prod=prod*i
    return prod

# main
endzahl=eval(input("Bis zu welcher Zahl soll multipliziert werden?: "))
ergebnis= produkt(endzahl)
print("Das Produkt lautet: ",ergebnis)
```

Aufgaben:

1. ***Erstellen Sie ein Speicher-Schema für das Produkt-Programm!***
2. ***Überlegen Sie sich, was passieren würde, wenn man innerhalb der Schleife ergebnis immer auf 13 setzt! Diskutieren Sie Ihre Voraussage mit anderen Kursteilnehmern! Probieren Sie es dann aus!***
3. ***Schreiben Sie eine Summe- und eine Produkt-Funktion in einem Programm, welche immer die Zahlen von einer Start- bis zu einer Endzahl (über Eingaben festzulegen) verarbeiten!***

8.4.2. Rekursion

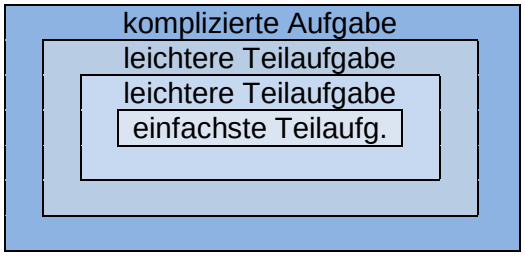
Kommen wir noch mal auf unser 10-Wasser-Kisten-Beispiel zurück. Um sie in den drutten Stock zu bekommen, können wir selbst mit jeweils 2 Kisten fünfmal Treppen steigen und die Kisten hochschleppen.
Eine andere Strategie wäre es, die Aufgabe einfach zu zerlegen. Ich übergebe das Kisten-Problem an den nächsten Party-Gast / Wassertrinker, indem ich ihn für den Transport von 8 Kisten verantwortlich mache. Ich selbst nehme 2 Kisten und bringe sie hoch. Der andere hat ein deutlich einfacheres Problem, als ich vorher mit 10 Kisten. Der Zweite kann nun genauso vorgehen. Sich einen "Dummen" suchen, der 6 Kisten als Auftrag bekommt und er selbst auch 2 Kisten nach oben transportiert. Der "Dumme" wird so weiterverfahren. Wenn es dann irgendwann nur noch 4 Kisten sind, übergibt der vorletzte Transporteur die (leichteste / letzte) Aufgabe an den letzten Party-Gast / Wassertrinker. Jeder der beiden löst nun seine Transport-Aufgabe und bringt jeweils 2 Kisten nach oben. In der Wohnung wird dann alles wieder zu einem 10-Kisten-Stapel zusammengesetzt.

in der Informatik versteht man darunter die Rückführung einer schwierigeren / aufwändigeren / komplizierteren / allgemeinen Aufgabe in eine leichtere / weniger aufwändigen / einfacheren / speziellen.
vom lat.: recurrere (zurücklaufen, zurückkehren)

besonders gern benutzt und besonders eindrucksvoll sind Rekursionen in der Grapfik-Programmierung
nutzt man dann noch die Turtle-Graphik (→ 8.8. Turtle-Graphik – ein Bild sagt mehr als tausend Worte), dann kann man Rekursion praktisch erleben (→ 8.8.6. Rekursion)

– theoretisch unendlich oft – in sich selbst geschachtelte Schleife
wobei hier nicht die Schleife das Struktur-Objekt ist sondern eine sich selbst-aufrufende Funktion

Ein Problem haben wir allerdings. Man braucht immer eine leichteste / letzte Teilaufgabe. Diese nennen wir Rekursions-Abbruch oder aus der anderen Richtung betrachtet Rekursions-Anfang. Die anderen – delegierenden / vereinfachenden – Schritte werden Rekursions-Schritt genannt.



In der Mathematik ist die Rekursion ein gängiges Mittel zur Definition von Funktionen

z.B.: Bildung einer Summe

sum(0) = 0 *Rekursions-Anfang*

jede andere Summe lässt sich dann so berechnen:

sum(n) = sum(n-1) + n *Rekursions-Schritt*

die gesamte Definition lautet dann

$$\text{sum}(n) = \begin{cases} 0, & \text{falls } n = 0 \quad \textit{Rekursions-Anfang} \\ \text{sum}(n-1) + n, & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

Auch wenn es ein bisschen wie eine interative Lösung aussieht, hier ist der entscheidende Unterschied, dass die Funktion sich selbst wiederaufruft. Bei der Interation wird nur wiederholt.

damit ein Problem rekursiv zu lösen geht, muss es die folgenden Bedingungen erfüllen:

- das Problem muss sich in eine einfachere Variante von sich selbst zerlegen lassen
- bei der Zerlegung in eine einfachere Variante muss irgendwann eine Variante erreicht werden, die sich ohne weitere Zerlegung lösen lässt
- wenn die Teilprobleme gelöst sind, dann müssen sich die Teil-Lösungen zu einer Lösung des Ausgangs-Problems zusammensetzen lassen

Definition(en): Rekursion
Unter Rekursion versteht man das nicht voraussehbare Wiederholen einer Aktion / Funktion durch Aufruf von sich selbst.
Rekursion ist das Problemlösungs-Konzept, bei dem eine (komplexe) Aufgabe in (kleinere, leichter lösbare) Teil-Aufgaben (der gleichen Klassen) zerlegt wird, diese gelöst werden und dann zur Gesamt-Lösung zusammengesetzt werden. Rekusionen bedürfen einer trivialen Teil-Aufgaben-Lösung, ab der eine weitere Aufgaben-Zerlegung nicht mehr durchgeführt werden kann.

Vorteile einer / der Rekursion

- **relativ einfache Defintion**
- **dem menschlichen Denken ähnlich**
- **Korrekheit ist i.A. leichter zu prüfen**
- **kürzere Formulierung**
- **kürze Implementierungen**
- **spart Variablen**
- **(i.A.) sehr effektiv**

Ob rekursives Arbeiten wirklich dem menschlichen Denken sehr nahe kommt, wage ich zu bezweifeln. Meine Erfahrungen sagen eher, dass rekursive Prinzipien / Funktionen zumindestens sehr einfach erscheinen, beim Umsetzen in ein Programm wird es deutlich schwieriger und problematisch wird es, wenn selbst neuartige Sachverhalte / Probleme rekursiv gelöst werden sollen
Meist erscheint dann irgendwie die interative Lösung logischer oder eingängiger. Kommt man später auf eine rekursive Lösung, ist sie zwar meist deutlich eleganter, aber auch schwerer zu verstehen und zu warten.

Nachteile einer / der Rekursion

- **unübersichtlicher Programmablauf**
- **schlechtes Laufzeit-Verhalten** — meist deutlich langsamer
- **größerer Speicher-Bedarf (großer Overhead von Funktions-Aufrufen)** — z.B. für Rücksprung-Adressen von noch nicht gelösten übergeordneten Funktions-Aufrufen

-

einige Programmiersprachen kennen nur Rekursionen, bei ihnen fehlen andere Wiederholungs-Strukturen (z.B. Scheme)
Computer arbeiten intern aber immer interativ, aber das ist nicht unsere Ebene

Wir unterscheiden direkte und indirekte Rekursion. Die direkte ist dadurch gekennzeichnet, dass die Funktion sich immer wieder selbst aufruft. Bei der indirekten Rekursion rufen sich mehrere Funktionen gegenseitig auf. Sind es z.B. zwei, dann ruft Funktion1 die Funktion2 auf und diese dann wieder Funktion1.

8.4.2.1. Rekursions-Beispiele: Summen- und Produkt-Bildung

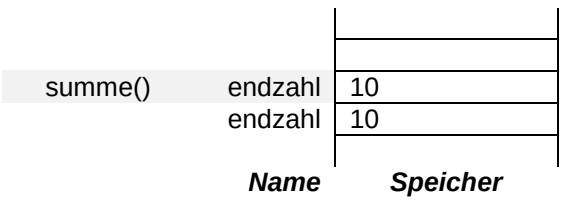
```
def summe(endzahl):
    sum=0
    for i in range(1,endzahl+1):
        sum=sum+i
    return sum

# main
endzahl=eval(input("Bis zu welcher Zahl soll summiert werden?: "))
print("Die Summe lautet: ",summe(endzahl))
```

Betrachten wir hier auch die beiden Funktionen (summe und produkt), die oben bei den Interationen nochmals besprochen worden.
Über die Rekursion beschreiben wir die Funktion summe wie oben besprochen:

$$\text{summe}(n) = \begin{cases} 0, & \text{falls } n = 0 \quad \textit{Rekursions-Anfang} \\ \text{summe}(n-1) + n, & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

```
def summe(endzahl):
    if endzahl==0:
       return 0
    else:
       return summe(endzahl-1)+endzahl

# main
endzahl=eval(input("Bis zu welcher Zahl soll summiert werden?: "))
print("Die Summe lautet: ",summe(endzahl))
```

Typisch ist die umgekehrte Abarbeitung zur kleinsten Zahl / zum Abbruch-Kriterium hin.
Die Speicher-Belegung ist aber letztendlich deutlich verschieden.
Beim Aufruf der Funktion summe wird wieder eine Kopie von endzahl angelegt. Nun wird die Verzeigung passiert und bevor irgenwas getan



wird, wird die Funktion schon wieder verlassen allerdings mit einem erneuten Aufruf von summe. Das Argument wurde aber um Eins verringert.
Zur Kennzeichnung eines untergeordneten Aufrufs verwende ich unterschiedlich dunkle Grautöne.
Dieser Vorgang wiederholt sich jetzt einige Male bis der Aufruf mit dem Argument 0 (für die endzahl) erfolgt.
Es erfolgt ein Return mit 0 und nun wird der Speicherstapel abgebaut, indem der Rückgabe-Wert des untergeordneten Funktions-Aufrufs mit der – auf der jeweiligen Ebene gültigen – endzahl addiert wird.
Letztendlich kommen wir so zum 1. Funktionsaufruf zurück und der gibt nun das Ergebnis (aus der Berechnung summe(9)+endwert) an den aufrufenden Programmschritt zurück (hier die Ausgabe).

summe()	endzahl	9
summe()	endzahl	10
	endzahl	10

summe()	**endzahl**	0
summe()	endzahl	1
		…
summe()	endzahl	7
summe()	endzahl	8
summe()	endzahl	9
summe()	endzahl	10
	endzahl	10

Schon bei nur 10 Rekursionen wird also deutlich mehr Speicher gebraucht, als in der interativen Version.
Typische Rekursionen haben meist eine deutlich größere Rekursionstiefen und häufig auch noch interne Variablen. Auch diese benötigen Platz im sogenannten Kellerspeicher, LIFO-Speicher oder Stack. Der zuletzt gespeicherte Inhalt wird zuerst wieder herausgeholt (Last In First Out). Anders herum kann man sich das Speicher-Prinzip auch als Stapel (engl.: stack) verstellen. Man muss Neues oben auflegen und auch von oben der Stapel wieder abbauen.
Die informatische Datenstruktur "Keller" wird später nochmals ausführlich (Objekt-orientiert) besprochen (→ 9.8. Keller).

Aufgaben:

1. *Erstellen Sie die Definition für ein Produkt!*
2. *Erstellen Sie ein Programm mit einer rekursiven Produkt-Funktion!*
3. *Zeigen Sie an einem Speicher-Schema, welche Variablen wann angelegt werden und welche Werte sie beinhalten!*

8.4.2.2. weitere typische Anwendungen für Rekursionen

8.4.2.2.1. Überführung einer Dezimal-Zahl in eine Dual-Zahl

```
def dualzahl(dezimalzahl):
    ganzzahlteiler=dezimalzahl/2
    rest=dezimalzahl%2
    if rest==0:
        stellensymbol="0"
    else:
        stellensymbol="1"
    if ganzzahlteiler==0
        return stellensymbol
    else:
        return dualzahl(ganzzahlrest) + stellensymbol
```

8.4.2.2.2. die Fakultät

faktorielle Funktion
für die Wahrscheinlichkeitsrechnung / Stochastik häufig gebraucht
in der Mathematik durch das Ausrufe-Zeichen nach der Zahl ausgedrückt:

6! = 1 * 2 * 3 * 4 * 5 * 6 = 720

oder eben allgemein:

n! = 1 * … * (n-1) * n = $\prod_{i=1}^{n} i$

die meisten Programmierer würden wohl auch eher interativ an die Implementierung herangegen (→ 8.4.1.1.2. Produkt-Bildung)
hier schauen wir uns aber auch mal die rekursive Lösung an:

$$\text{fakultät}(n) = \begin{cases} 1, & \text{falls } n = 1 \quad \textit{Rekursions-Anfang} \\ \text{fakultät}(n-1) + n, & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

```
def fakultaet(x):
    if x==1: return 1
    else:
        return fakultaet(x-1)*x
```

8.4.2.2.3. die FIBONACCHI-Folge

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

$$fib(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 1, & \text{falls } n = 1 \\ fib(n-1) + fib(n-2), & \text{sonst} \end{cases}$$

Rekursions-Anfang

Rekursions-Schritt

Exkurs: FIBONACCHI ohne die Vorglieder?

Das Berechnen eines bestimmten Gliedes der FIBONACCHI-Folge ist durch Rekursion und Interation möglich. Beide Lösungswege – also die interative bzw. die rekursive – haben durch die vielen Wiederholungen bzw. Funktionsaufrufe einen recht großen Rechenaufwand. Schließlich müssen alle Vorglieder berechnet werden, um dann die letzten beiden Vorglieder zum Ergebnis zu addieren.
Besonders für höhergliedrige Werte in der Folge ist der Rechenaufwand dann enorm.
Der französische Mathematiker J.-Ph.-M. BINET schlug (1843) eine andere Funktion zur Berechnung der Einzelglieder vor:

$$fib(n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

Einen solchen Lösungsweg nennen wir **explizit**. Explizite Lösungen sind meist extrem schnell – vor allem im Vergleich zu den anderen beiden Lösungs-Strategien. Der Aufwand für die Implementierung liegt im Bereich der Interation – also etwas aufwändiger, als für eine Rekursion.
Explizte Lösungen von Problemen, die im "normalen" Leben einen großen Rechnenaufwand haben stellen z.B. häufig Sicherheits-Probleme dar. Wenn z.B. eine Sicherheits-Lösung darauf aufbaut, dass sie erst mit einem riesigen Rechenaufwand geknackt werden kann, und es exisiert auf einmal eine explizite Lösung, dann stürzt das Sicherheits-Konzept in sich zusammen.

Aufgaben:

1. ***Programmieren Sie die nachfolgende "Super"-FIBONACCHI-Folge als rekursive Funktion mit kleinem Rahmen-Programm zur Anzeige mehrerer Folge-Glieder!***

$$sfib(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 1, & \text{falls } n = 1 \\ 2, & \text{falls } n = 2 \\ sfib(n-1) + sfib(n-2) + sfib(n-3), & \text{sonst} \end{cases}$$

Rekursions-Anfang

Rekursions-Schritt

zur Kontrolle: erwartete erste Glieder der Folge:
0, 1, 2, 3, 6, 11, 20, 37, 68, 125, 230, 423, …

Aufgaben (für Fortgeschrittene):

2. *Erstellen Sie ein Programm, dass für die ersten 20 Glieder der FIBONACCHI-Folge die Werte jeweils klassisch interativ und rekursiv und dann noch einmal mit der BINET-Funktion berechnet. Prüfen Sie, ob es Differenzen gibt (Anzeigen lassen!)!*
3. *Die sogenannte PADOVAN-Folge (auch: kleine Schwester der FIBONACCHI-Folge) versucht die verzögerte Fortpflanzungs-Fähigkeit der Nachkommen nachzubilden. Statt mit den beiden unmittelbaren Vorgängern (n-1 und n-2) zu rechnen, werden die Vorgänger n-2 und n-3 addiert. Gestartet wird mit dem Wert 1 für die ersten drei Glieder. Ertellen Sie ein Programm, dass die PADOVAN-Folge für die Glieder 1 bis 20 simuliert!*
4. *Stellen Sie die Glieder der FIBONACCHI- und der PADOVAN-Folge in einer tabellarischen Form gegenüber (Glieder 1 bis 30)!*

$$\text{pad}(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 1, & \text{falls } n = 1 \quad \textit{Rekursions-Anfang} \\ 1, & \text{falls } n = 2 \\ \text{pad}(n-2) + \text{pad}(n-3), & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

zur Kontrolle: erwartete erste Glieder der Folge:
0, 1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, …

Aufgaben für das gehobene Anspruchsniveau:

5. *Untersuchen Sie, ob es zwischen den Gliedern der FIBONACCHI-Folge einen Wachstums-Faktor (Quotient des aktuellen und dem vorlaufenden Glied) gibt! Wie verhält sich dieser Quotient im Verlauf der Folge?*
6. *Untersuchen Sie gleiches für die PADOVAN-Folge!*

8.4.2.2.4. das ggT – der Größte gemeinsame Teiler

Natürlich müsste es der ggT (GGT; eng.: gcd (greatest common divisor)) heißen, aber wer spricht schon so?

beim ggT mehrerer (mehr als 2) Zahlen muss allerdings auf die Primfaktoren-Zerlegung zurückgegriffen werden

bei zwei Zahlen

$$10584 = \mathbf{2^3} * \mathbf{3^3} * 7^2$$
$$40500 = \mathbf{2^2} * \mathbf{3^4} * 5^3$$

$$\text{ggT}: \mathbf{2^2} * \mathbf{3^3} = 108$$

wird für mehr Zahlen

1400 =	$\mathbf{2^3}$			*	$\mathbf{5^2}$	*	7^2	
283500 =	$\mathbf{2^2}$	*	3^4	*	$\mathbf{5^3}$	*	7^1	
20250 =	$\mathbf{2^1}$	*	3^4	*	$\mathbf{5^3}$			
ggT:	$\mathbf{2^1}$			*	$\mathbf{5^2}$			= 50

Primfaktoren-Zerlegung sehr rechen-aufwändig

EUKLIDischer und STEINscher Algoritmus

Grundidee von EUKLID und dann durch STEIN verbessert

40500	:	**10584**	=	3	Rest:	**8748**
10584	:	**8748**	=	1	Rest:	1836
8748	:	1836	=	4	Rest:	1404
1836	:	1404	=	1	Rest:	432
1404	:	432	=	3	Rest:	108
432	:	**108**	=	4	Rest:	**0**

ggT(x, y, z) = ggT(ggT(x, y), z) = ggT(x, ggT(y, z))

8.4.2.2.5. Erkennung von Palindromen

rekursiv:

```
def ist_palindrom(zeichenkette):
    if len(zeichenkette)<=1:
        return 1
    if zeichenkette[0]!=zeichenkette[-1]:
        return 0
    return ist_palindrom(zeichenkette(s[1:-1])
```

interativ:

```
def ist_palindrom(zeichenkette):
    links=0
    while links<rechts:
        if zeichenkette[links]!=zeichenkette[rechts]:
            return 0
        links+=1
        rechts-=1
    return 1
```

mit speziellen Python-Funktionen für Strings und Listen:

```
def ist_palindrom(zeichenkette):
    buchstabenliste=list(zeichenkette)
    buchstabenliste.reverse()
    return ("".join(1))
```

ist bei Zeitvergleichen die schnelleste Variante, weil die Listen- und String-Funktionen in Maschinensprache realisiert sind

8.4.2.2.x. weitere klassische Rekursions-Probleme

Türme von Hanoi

rekursive Zerlegung des aktuellen Turm in die größte / untereste Scheibe und einen kleineren (Rest-)Turm

ACKERMANN-Funktion

1926 von Wilhelm ACKERMANN beschrieben
wird zur Austestung von Speicher- und Computer-Modellen benutzt, da die Funktion extrem schnell wächst

$$
\begin{aligned}
ack(a, b, 0) &= a + b \\
ack(a, 0, n+1) &= ack2(a, n) \\
ack(a, b+1, n+1) &= ack(a, ack(a, b, n+1), n)
\end{aligned}
$$

$$
ack2(a, n) = \begin{cases} 0, & \text{wenn } n=0 \\ 1, & \text{wenn } n=1 \\ a, & \text{wenn } n>1 \end{cases}
$$

durch PÈTER 1935 etwas einfacher definiert:

$$
\begin{aligned}
ack(0, m) &= m+1 \\
ack(n+1, 0) &= ack(n, 1) \\
ack(n+1, m+1) &= ack(n, ack(n+1, m))
\end{aligned}
$$

rekursiv:

```
def ackermann(n, m):
    if n==0:
        return m+1
    elseif  m==0:
        return ackermann(n-1, 1)
    else:
        return ackermann(n-1, ackermann(n, m-1)
```

teilweise interativ:

```
def ackermann(n, m):
    while n!=0:
       if m==0:
           m=1
       else:
           m=ackermann(n, m-1)
       n+=1
    return m+1
```

Quicksort

Beim Quicksort-Verfahren wird eine Liste von Zahlen od.ä. Objekten dadurch sortiert, das die originale Liste in immer kleiner werdende Liste aufgeteilt wird. Dabei wird einfach nur nach Größe in die eine oder andere Liste eingeordnet. Als Entscheidungs-Element (Grenzwert) wird ein zufälliger Wert oder z.B. einfach das erste Element der Liste benutzt. Das Entscheidungs-Element wird auch Pivot-Element genannt. Das Wörtchen *pivot* bezeichnet im Französischen den Dreh- und Angel-Punkt.
Optimalerweise sollten die Teil-Listen immer die halben Listen der Vorgänger-Liste sein, dann sortiert dieses Verfahren sehr schnell.
Genaueres später bei der Besprechung verschiedener Sortier-Algorithmen (→ 8.15. Sortieren – eine Wissenschaft für sich).

Mergesort

Eine ähnliche Strategie verfolgt der Sortier-Algorithmus Mergesort. Auch hier wird in kleine(re) Listen zerlegt, die dann für sich sortiert werden. Am Schluss werden die sortierten Teil-Listen durch Mischen (*merge* = engl.: verschmelzen) vereint.
Mergesort folgt dem Teile-und-herrsche-Prinzip (divide and conquer), welches erstmals von J. VON NEUMANN (1945) beschrieben wurde und praktisch auch in seinen Rotor-Maschinen zum Knacken des Enigma-Code's verwendet wurde.
Genaueres später bei der Besprechung verschiedener Sortier-Algorithmen (→ 8.15. Sortieren – eine Wissenschaft für sich).

Potenzierung von Zahlen

interativ

```
def potenz(basis, exponent):
    pot=1
    for i in range(exponent+1):
        pot*=basis
    return pot
```

rekursiv

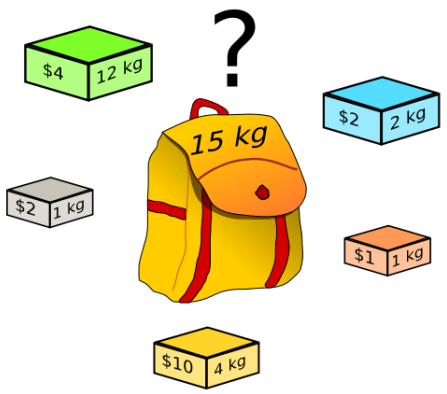
```
def potenz(basis, exponent):
    if exponent==0:
        return 1
    else:
        return basis*potenz(basis, exponent-1)
```

Rucksack-Problem

eng.: knapsack problem
Optmirungs-Problem aus der Kombinatorik

gegeben ist eine Menge von Objekten, die einen Nutzwert und ein Gewicht (Kostenfaktor) besitzen
gesucht ist eine Teilmenge, deren Gewicht eine bestimmte Grenze nicht überschreitet und der Nutzen aber maximiert sein soll

gehört zu den klassischen NP-vollständigen Problemen (Richard KARP (1972))



Veranschalichung des Rucksack-Problems
Q: de.wikipedia.org (Dake)

Zahlen-Beispiel von http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo15.php

Objekt	1	2	3	4	5	6	7	8		
Gewicht	153	54	191	66	239	137	148	249		
Profit	232	73	201	50	141	79	48	38		
Profit-Dichte	1,52	1,35	1,05	0,76	0,59	0,58	0,32	0,15		

Gewichts-Schranke soll z.B. bei 645 liegen

1. intuitiver Lösungs-Ansatz:
nehme die Objekte mit der höchsten Profit-Dichte
also → 1, 2, 3, 4 → Gewicht=464 → Profit=556

2. Lösung, wie 1. und Auffüllen mit weiteren passenden Objekten (entsprechend der Rangfolge)
also → 1, 2, 3, 4, 6 → Gewicht=601 → Profit=647

→ aber nicht optimal! ?????
es muss jede Kombination ausprobiert werden!

2^n Möglichkeiten (einpacken oder nichteinpacken / 1 oder 0)
Problem ist hier die expotentielle Steigerung des Rechen-Aufwandes
es gibt scheinbar mehrere Lösungen !?

besser ist der Algorithmus von NEMHAUSER und ULLMANN (1969)
basiert auf PARETO-Prinzip

Alpha-Beta-Suche für Spielzüge bei Brettspielen (Computer-Stategie)

Volumen-Berechnung einer n-dimensionalen Hyperkugel

Suche in einem Baum

Weg aus einem Labyrint

Rechte-Hand-Regel

geht natürlich auch als Linke-Hand-Regel

Permutationen

```
def permutation():

    return
```

effektive Speicherung von Daten (z.B. Bilder)

Wollten wir das nebenstehende Bit-Muster / Bild über eine Liste abspeichern, dann würde diese mit 64 Elementen doch recht lang werden. Nehmen wir an, es geht oben links los und es wird Zeilen-weise gearbeitet, dann ergibt sich die folgende Liste:

1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	1	1
1	1	1	1	0	0	1	1
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	1	1	0	0	0	0

muster=[1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 0]

Das Bild / Daten-Muster wird zuerst als das größtmögliche Quadrat (gleicher Elemente) betrachtet. Wäre es z.B. homogen nur mit Einsen gefüllt, dann würde sich als Muster die Liste **muster=[1]** ergeben. Statt der 64 Speicher-Elemente hätten wir es nur noch mit einem zu tun. Wir bräuchten also grob 1/64 des Speicherbedarfs – wenn das keine Kompression ist?!
Das Muster ist aber heterogen, also müssen wir es verkleinern.
Die verkleinerten Quadrate sind umrandet hervorgehoben. Für jedes der kleineren Quadrate müssen wir nun in unserer muster-Liste ein Listen-Element verwenden. Da es mehr als eins ist, wird klar, dass das gesamte 8x8-Quadrat strukturiert ist. Die Grundstruktur sieht dann so aus:

muster=[, , ,]

Wären jetzt die kleineren (4x4)-Quadrate einheitlich gefärbt (mit 1 oder 0 belegt), dann würden wir mit 4 Listen-Elementen hinkommen, was immer noch einer Effektivität der Kompression von 4/64 entsprechen würde. Aber leider ist es im Beispiel nicht so, also müssen wir genauer weiter differenzieren.
Das oberste linke Quadrat ist vollständig mit Einsen gefüllt, also speichern wir uns in die Teil-Liste eine Eins:

muster=[1, , ,]

Nun wechseln wir zum rechten oberen 4x4-Quadrat. Es ist nicht homogen und muss deshalb wieder unterteilt werden. Es entsteht also eine Liste (**rot gekennzeichnet**) in der Liste (Das Listen-Element ist selbst wieder eine Liste).

muster=[1, **[, , ,]**, ,]

Die sich ergebenden 2x2-Quadrate sind homogen, also kann die Muster-Liste nun so geschrieben werden:

muster=[1, **[0, 0, 0, 1]**, ,]

Das untere, linke Quadrat ist nicht homogen, also muss es zerlegt werden. Auch das oberste linke 2x2-Quadrat ist nicht homogen, also muss es als Bit-Muster in die Liste geschrieben werden.

muster=[1, [0, 0, 0, 1], [[0, 1, 1, 0], , ,],]

Das zweite 2x2-Quadrat ist homogen mit Nullen belegt, also speichern wir ein 0 in die Liste.

muster=[1, [0, 0, 0, 1], [[0, 1, 1, 0], 0, ,],]

Bei den unterern beiden 2x2-Quadraten verfahren wir in der gleichen Weise und erhalten dann:

muster=[1, [0, 0, 0, 1], [[0, 1, 1, 0], 0, [1, 0, 1, 1], 1],]

Bleibt das letzte (untere, rechte) 4x4-Quadrat. Es ist homogen, so dass die Liste nur die darin enthaltene Null repräsentieren muss:

muster=[1, [0, 0, 0, 1], [[0, 1, 1, 0], 0, [1, 0, 1, 1], 1], 0]

Im vergleich zur obigen Voll-Liste kommen wir nun mit 26 Speicher-Elementen aus. Das bedeutet eine Verbesserung fast um den Faktor 2,5 (grob: 16/64).
Die Kompressionsraten sind sehr theoretisch berechnet. Es muss beachtet werden, dass noch Strukturierungs-Elemente (zur Unterscheidung von über- und unter-geordneten Listen) mit abgespeichert werden müssen.
So ähnlich – wie hier besprochen – laufen z.B. Kompressions-Verfahren, wie das JPEG oder MP4 ab. Neben dem Vergleich der Bild-Elemente werden auch noch die vorlaufenden Bilder mit verglichen. Dabei nutzt man den Effekt aus, dass sich in einer Bildfolge meist nur wenige – isolierte – Teile verändern.

Aufgaben:

1. *Übernehmen Sie das Muster und Muster-Liste! Kennzeichen Sie durch unterschiedlich farbige Umrandungen im Muster und durch entsprechend farbige Klammern, welche Bitmuster zu welchen Listen-Elementen gehören!*

Eine Rekursion bietet sich immer dann an, wenn das Problem / die Funktion schrittweise auf ein kleineres / leichteres Problem // eine einfachere Funktion reduziert werden kann.

McCARTHYs "91-Funktion"

$$f(n) = \begin{cases} n-10 & \text{falls } n > 100 \\ f(f(n+11)) & \text{sonst} \end{cases}$$

PELL-Folge

$$P(n) = \begin{cases} 0, & \text{falls } n = 0 & \textit{Rekursions-Anfang} \\ 1, & \text{falls } n = 1 & \textit{Rekursions-Anfang} \\ 2P(n-1) + P(n-2) & \text{sonst} & \textit{Rekursions-Schritt} \end{cases}$$

erste Elemente: 0, 1, 2, 5, 12, 29, 70, 169, 408, …

die ersten beiden Elemente sind mit 0 und 1 definiert
die nachfolgenden Elemente ergeben sich als Summe aus dem verdoppelten Vorgänger und dem (einfachen) Vorvorgänger

PELL-Folge 2. Art

$$Q(n) = \begin{cases} 2, & \text{falls } n = 0 & \textit{Rekursions-Anfang} \\ 2, & \text{falls } n = 1 & \textit{Rekursions-Anfang} \\ 2Q(n-1)+Q(n-2) & \text{sonst} & \textit{Rekursions-Schritt} \end{cases}$$

erste Elemente: 2, 2, 6, 14, 34, 82, 198, 478, 1154, …

die ersten beiden Elemente sind mit 2 definiert
die nachfolgenden Elemente ergeben sich als Summe aus dem verdoppelten Vorgänger und dem (einfachen) Vorvorgänger

LUCAS-Folge(n)

$$L(n) = \begin{cases} x, & \text{falls } n = 0 \quad \textit{Rekursions-Anfang} \\ y, & \text{falls } n = 1 \quad \textit{Rekursions-Anfang} \\ L(n-1)+L(n-2) & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

erste Elemente: immer abhängig von x und y
bei x=2 und y=1 → 2, 1, 3, 4, 7, 11, 18, 29, 47, …

die ersten beiden Elemente sind mit x und y definiert
die nachfolgenden Elemente ergeben sich als Summe aus dem Vorgänger und dem Vorvorgänger

JACOBSTHAL-Folge

$$J(n) = \begin{cases} 0, & \text{falls } n = 0 \quad \textit{Rekursions-Anfang} \\ 1, & \text{falls } n = 1 \quad \textit{Rekursions-Anfang} \\ J(n-1)+2J(n-2) & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

erste Elemente: 0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, 683, …

die ersten beiden Elemente sind mit 0 und 1 definiert
die nachfolgenden Elemente ergeben sich als Summe aus dem Vorgänger und dem verdoppelten Vorvorgänger

RECAMÀNs-Folge
(OEIS → A005132)

$$R(n) = \begin{cases} 0, & \text{falls } n = 0 \quad \textit{Rekursions-Anfang} \\ R(n-1)-n & \text{falls } R(n) >= 0 \text{ und nicht in Sequenze} \quad \textit{Rekursions-Schritt} \\ R(n-1)+n & \text{sonst} \quad \textit{Rekursions-Schritt} \end{cases}$$

erste Elemente: 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8, 25, 43, …

die ersten beiden Elemente sind mit 0 und 1 definiert
die nachfolgenden Elemente ergeben sich als Summe aus dem Vorgänger und dem verdoppelten Vorvorgänger

interessante Links:
https://oeis.org/wiki/Welcome (On-Line Encyklopedia of Integer Sequences ® OEIS ®)
https://oeis.org/A?????? (Informationen zur Folge mit der Nummer ??????)

Aufgaben für die gehobene Anspruchsebene:

1. *Informieren Sie sich zur Biographie von N.J.A. SLOANE!*
2. *Was verbirgt sich hinter der Folge A000108?*

NUR!!! zum Üben: die DREWS-Folgen

Nicht wundern, natürlich gibt es diese Folge (wahrschein) nicht wirklich – und wenn, dann unter einem anderen Namen! Sie sind praktisch abgewandelte FIBONACCHI-Folgen. Bei der ersten Folge wird immer eine 1 dazuzählt. Also typisch DREWS – immer noch Einen drauf setzen. Die Folgen haben keinen tieferen Zweck, außer dem Programmieren zu dienen.

0, 1, 2, 4, 7, 12, 20, 33, 54, 88, 143, 232, …

$$dre(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 1, & \text{falls } n = 1 \\ dre(n-1) + dre(n-2) + 1, & \text{sonst} \end{cases}$$

Rekursions-Anfang

Rekursions-Schritt

Die zweite Folge ist etwas komplexer. Hier unterscheidet sich das Zuzählen danach, ob die Gliednummer gerade oder ungerade ist.

0, 1, 2, 5, 8, 15, 24, 41, 66, 109, 176, 287, …

$$dre2(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 2, & \text{falls } n = 1 \\ dre2(n-1) + dre2(n-2) + 1, & \text{sonst, falls n gerade} \\ dre2(n-1) + dre2(n-2) + 2 & \text{sonst, falls n ungerade} \end{cases}$$

Rekursions-Anfang

Rekursions-Schritte

Aufgaben:

1. *Programmieren Sie die 1. DREWS-Folge als rekursive Funktion mit einem kleinen Rahmen-Programm!*
2. *Erstellen Sie ein Programm, mit dem die DREWS-Folge sowohl rekursiv als auch interativ berechnet wird!*
3. *Entwickeln Sie nun ein Programm, das die DREWS2-Folge rekursiv berechnet!*

für die gehobene Anspruchsebene:

4. *Erstellen Sie ein Programm, mit dem die DREWS2-Folge sowohl rekursiv als auch interativ berechnet wird!*
5. *Vergleichen Sie den Implementier-Aufwand für die Berechnung der DREWS2-Folge beim interativen und rekursiven Vorgehen!*

8.4.2.2. direkte Gegenüberstellung von interativen und rekursiven Algorithmen

In der Literatur und in der täglichen Programmierarbeit finden sich bzw. entstehen die unterschiedlichsten Umsetzungen von bestimmten Problem. Einige sind hier gesammelt und jeder programmierer wird nach und nach den einen oder anderen programm-Text hinzutuen können. Ob die einzelnen Lösungen immer optimal (gut lesbar, schnell, wenig Speicherbedarf, …) sind, wird hier nicht bewertet. Sollten Algorithmen entscheidend für ein Programm sein, dann müssen spezielle Test (Abfrage Speicherbedarf, Zeitmessungen, …) erfolgen. Auf einige Möglichkeiten gehen wir noch ein.
In einigen Algorithmen sind **blaue** **print**-Anweisungen eingebaut. Diese dienen als optionale Ausgabe, um das Arbeiten des Algorithmus zu verfolgen. Für echte Anwendungen sollten sie dann raus genommen werden. Ev. lassen sich weitere sinnvolle Ausgaben erzeugen, z.B. um die Anzahl der Schleifendurchläufe bzw. die Rekursion-Aufrufe zu zählen. Dafür müssen dann aber eigene Variablen und Zähl-Anweisungen eingebaut werden.
Bei einigen ausgewählen Algorithmen-Umsetzungen notieren wir das in **rot**er Farbe. Auch diese Quelltext-Teile sollten vor dem echten Einsatz entfernt werden.

8.4.2.2.1. GGT – größter gemeinsamer Teiler

Lösung	interativ	rekursiv

1	`def ggt(a,b):`
	`  i = 0`
	`  while b > 0:`
	`  i += 1`
	`  print("Durchlauf: ",i)`
	`  print("a= ",a,"b= ",b)`
	`  r = a % b`
	`  a = b`
	`  b = r`
	`  return a`
		`def ggt(a,b,i):`
		`  i += 1`
		`  print("Aufruf: ",i)`
		`  print("a= ",a,"b= ",b)`
		`  if b == 0:`
		`  return a`
		`  return ggt(b, a % b,i)`
		
		`# Aufruf der Funktion:`
		`i = 0`
		`ggt(a,b,i)`

2	`def ggt(a,b):`
	`  while b != 0:`
	`  a,b = b, a % b`
	`  return a`
		`def ggt(a,b):`
		`  return if a == b:`
		`  a`
		`  elif a > b:`
		`  ggt(a-b,b)`
		`  else:`
		`  ggt(a, b-a)`

3	`def ggt(a,b):`
	`  while a != b:`
	`  if a > b:`
	`  a=a-b`
	`  else:`
	`  b=b-a`
	`  return a`	

		`def ggt(a,b):` `  return if a > b:`

8.4.2.2.2. Palindrom-Prüfung

Lösung	interativ	rekursiv

1	`def ist_palim(s):`
	`  links = 0`
	`  rechts = len(s)-1`
	`  while links < rechts:`
	`  if s[links] != s[rechts]:`
	`  return 0`
	`  links += 1`
	`  rechts -= 1`
	`  return 1`
		`def ist_palim(s):`
		`  if len(s) <= 1:`
		`  return 1`
		`  if s[0] != s[-1]:`
		`  return 0`
		`  return ist_palim(s[1:-1])`

2		

außer Konkur-renz	`def ist_palim(s):`
	`  liste = list[s]`
	`  liste.reverse()`
	`  return ("".join(liste))`	

8.4.2.2.2. Potenz-Prüfung

ist p eine ganzzahlige Potenz von x

Lösung	interativ	rekursiv

1	`def istpotenz(p,x):`
	`  return if p == 1 or p == x`
	`  or p % x != 0:`
	`  p == 1 or p == x`
	`  else:`
	`  istpotenz(p/x,x)`
		`def istpotenz(p,x):`
		`  return if p == 1 or p == x:`
		`  True`
		`  elif p % x !=0:`
		`  False`
		`  else:`
		`  istpotenz(p/x,x)`

2	`def istpotenz(p,x):`
	`  while p != 1 and p != x`
	`  and p % x == 0:`
	`  p = p / x`
	`  return p == 1 or p == x`	

8.4.3. komplexe Programmier-Aufgaben:

Wählen Sie eine geeignete oder Ihre präferierte Programmiersprache zur Lösung der nachfolgenden Aufgaben aus!
Überlegen Sie sich bzw. vergleichen mit anderen, ob die von Ihnen präferierte programmiersprache gut geeignet ist das gewählte Problem zu lösen!
Zahlen-Eigenschaften nach: www.zahlen.mathematic.de

Aufgaben:

1. ***Berechnen Sie die Summe und das Produkt einer Reihe von einzugebener Zahlen sowie Summe und Produkt der reziproken Werte!***
2. ***Erstellen Sie ein Programm, dass im Zahlen-Raum bis zur einer einzugebenen (größeren) natürlichen Zahl, die Kombination von drei aufeinanderfolgenden Primzahlen findet, deren Produkt möglichst dicht an der Zahlen-Grenze liegt!***
3. ***Prüfen Sie ob eine als Zeichen-String vorgegebene Zahl (ohne Leer- und Vorzeichen bzw. Nachkommastellen) im auszuwählenden Zahlensystem gültig ist! (Die Ziffern werden als ASCII-Zeichen notiert. Gültige und unterscheidbare Zeichen sind: 0 .. 1 A .. Z a .. z → das sollte auch bis zum Sexagesimal-System reichen! Doppeldeutung A = a muss nicht beachtet werden!)***
4. ***Lassen Sie durch eine Erweiterung des Programms von 3. prüfen, ob es sich bei der eingegeben Zahl um eine normale Zahl handelt! Normale Zahlen enthalten alle Ziffern ihres Alphabetes mit der gleichen Häufigkeit.***
5. ***Erstellen Sie das Programm "Zahlen-Charakterisierer"! Das Programm soll eine einzugebene ganze Zahl (ev. zuerst nur für natürliche Zahlen) Charakter-Eigenschaften prüfen bzw. bestimmen und ausgeben, ob die Zahl die Eigenschaft hat oder nicht bzw. den berechneten Wert. Das Programm sollte später um weitere Zahlen-Eigenschaften ergänzt werden können und passend kommentiert sein! Auf die (spätere) Nutzbarkeit von Unterprogrammen ist zu achten! Wählen Sie sich mindestens 12 Eigenschaften aus! Die Reihenfolge kann frei geändert werden!***
   - ***a) männliche Zahl (Zahl ist ungerade und größer als 1)***
   - ***b) Quersumme (ist die Summe der einzelnen Ziffern der Zahl (ohne deren Potenzwert))***
   - ***c) titanische Zahl (ist eine Primzahl mit mindestens 1000 Stellen***
   - ***d) weibliche Zahl (Zahl ist eine positive gerade Zahl)***
   - ***e) Totient od. Indikator (ist die Anzahl der Primzahlen, die kleiner als die (gegebene) Zahl ist)***
   - ***f) zusammengesetzte (od. zerlegbare od. teilbare) Zahl (ist eine Zahl, die mehr als zwei positive Teiler hat ODER eine gerade Zahl, die größer als 1 ist)***
   - ***g) abundante Zahl (wenn echte Teilersumme (Summe aller Teiler (ohne Rest), außer die Zahl selbst) größer als die Zahl selbst ist)***
   - ***h) arme od. defizierte od. mangelhafte Zahl (wenn echte Teilersumme kleiner als das doppelte der Zahl ist)***

i) vollkommene od. perfekte Zahl (wenn die echte Teilersumme gleich der Zahl selbst ist)

j) Sophie-GERMAIN-Primzahl (sind Primzahlen, bei denen der Term 2 p +1 wieder eine Primzahl ist)

k) reiche od. überschießende od. übervollständige Zahl (wenn die echte Teilersumme größer als das Doppelte der Zahl selbst ist)

l) SMITH-Zahl (wenn die Quersumme der Zahl gleich der Quersummen ihrer Primfaktoren ist; außer Primzahlen!)

m) erhabene Zahl (wenn Zahl und deren echte Teilersumme vollkommene Zahlen sind)

n) palindrome Zahl (wenn die Zahl und die umgedrehte Zifferfolge gleich (groß) sind)

o) palindrome Primzahl (wenn Zahl eine Primzahl ist und die Zahl und deren umgedrehte Zifferfolge gleich sind)

p) SIERPINSKI-Zahl (ist eine ungerade, natürliche Zahl n, bei der der Term $n\ 2^x + 1$ immer eine zusammengesetzte Zahl ergibt (x ist eine beliebige natürliche Zahl))

q) RIESEL-Zahl (sind ungerade, natürliche Zahlen, bei denen der Term $n\ 2^x - 1$ immer eine zusammengesetzte Zahl ergibt (x ist eine beliebige natürliche Zahl))

r) strobogrammatische Zahl (ist eine Zahl, die um 180° gedreht wieder die gleiche Zahl ergibt (hier gelten 1, 2 mit 5, 6 mit 9, 8 und 0 als drehbare oder strobogrammatische Ziffern))

s) strobogrammatische Primzahl (ist eine Primzahl, die auch strobogrammatisch ist)

6. Gesucht wird ein modulares Programm, dass für zwei natürliche Zahlen prüft, ob es sich um ein Paar mit den folgenden Eigenschaften handelt!

a) befreundete Zahlen (wenn die echten Teilersummen beider Zahlen gleich sind))

b) Primzahlen-Zwilling (wenn zwei aufeinanderfolgende Primzahlen eine Differenz von 2 aufweisen)

c) Teiler-fremde (od. inkommensurable) Zahlen (ganze Zahlen, die außer -1 und 1 keine gemeinsamen Teiler besitzen)

7. Gesucht wird ein modulares Programm, dass für drei natürliche Zahlen prüft, ob es sich um ein Tripel mit den folgenden Eigenschaften handelt!

a) pythagoreische Zahlen (Tripel erfüllt die diophantische Gleichung 2. Grades ($a^2 + b^2 = c^2$))

b) Primzahlen-Drilling (wenn drei aufeinanderfolgende Zahlen die Reihe p, p+2, p+6 bilden ODER wenn innerhalb einer Dekade (also 10 aufeinanderfolgenden Zahlen) drei Primzahlen vorkommen)

8.

8.5. Umgang mit Dateien

8.5.0. Dateien und Ordner

Text-Dateien
relativ leicht zu erzeugen, immer selbst durch Programmierer möglich, meist aber Module zum effektiveren Umgang verfügbar, sowohl vom Computer, als auch von Menschen lesbar
relativ Fehler-tolerant

praktisch ein Umleiten der Bildschirmausgabe in eine Datei

Binär-Dateien
Daten sind sehr kompakt gespeichert, praktisch nur noch Maschinen-lesbar
Fehler können für völlige Unlesbarkeit der Daten sorgen

Arbeiten mit Dateien bestehen immer aus drei Abschnitten, die unbedingt eingehalten bzw. erledigt werden müssen

- **Eröffnung, Initialisierung** — Festlegen der Datei über den Dateinamen und den Datentyp
  Festlegung der Zugriffsart auf Datei
- **eigentliches Schreiben bzw. Lesen** — eben genau das
  (praktisch könnte dieser Abschnitt auch entfallen, aber wozu dann der andere – unbedingt notwendige! - Aufwand)
- **Datei-Freigabe** — Beenden des Zugriffs auf die Datei, damit wird die Datei für andere Programme, Programmteile etc. benutzbar
  an dieser Stelle erfolgt vielfach erst das physikalische Schreiben

8.5.1. Dateien lesen

8.5.1.1. Lesen von Text-Dateien

Dateivariable = **open(**Dateiname,"r"**)**

Zeilenlistenvariable = Dateivariable.readlines()

Zeilenvariable = Dateivariable.readline()

Dateivariable**.close()**

Dateivariable**.seek(**Position**)**

Will man den gesamten Datei-Inhalt in einen String schreiben, dann lässt sich das folgendermaßen realisieren:

```
Dateivariable = open(Dateiname,"r")
Inhalt = Dateivariable.read()
print("Datei-Typ: ",type(Inhalt)
print("Datei-Inhalt:")
print(Inhalt)
Dateivariable.close()
```

8.5.1.1.1. Lesen von CSV- bzw. strukturierten TXT-Dateien

Einlesen einer Text-Datei und Speichern als CSV

```
zielDatei = open("daten.CSV", "w")

for zeile in open("Text.TXT"):
    zeile = zeile.strip()
    if zeilestartswith("#"):
        continue
    elemente = zeile.split()
    print(elemente)
print(";".join(elemente), file = zielDatei)
```

8.3.1.1.2. Lesen von XML-Dateien

spezielle Module verfügbar

8.1.1.1.3. Lesen von JSON-Dateien

spezielle Module zum decodieren verfügbar

```
import sys, json

dateinmae = "test.JSON"

print(json.dumps(json.load(dateinmae)), indent =2))
```

8.5.1.2. Lesen von Binär-Dateien

8.5.2. Dateien schreiben

8.5.2.1. Schreiben von Text-Dateien

8.5.2.1.1. Schreiben einer neuen Datei

Dateivariable = **open(**Dateiname,"w"**)** zum Neuschreiben

Dateivariable**.write(**"\n"+Zeile | "\n"+Zeilenvariable**)**
die write-Funktion liefert übrigens die Anzahl der geschriebenen Zeichen wieder zurück

alternativ auch Umleitung der Bildschirmausgabe möglich
print >> Dateivariable, Text | Textvariable
etwas einfacher, weil Ausgaben immer in String- / Text-Format umgewandelt werden

Dateivariable**.close()**
hier extrem wichtig, weil erst jetzt das echte Speichern erfolgt!

mit writelines(StringListe) kann eine Liste von Strings in einen Text-Datei geschrieben werden

8.5.2.1.2. anhängendes Schreiben

zum Anhängen weiterer Daten an eine existierende Datei
Dateivariable = **open(**Dateiname,"a"**)**
Dateivariable**.write(**Zeile+"\n" | Zeilenvariable+"\n"**)**

letzte Zeile ohne "\n"
Dateivariable**.write(**Zeile | Zeilenvariable**)**

Dateivariable**.close()**
nicht vergessen!

8.5.2.1.3. Schreiben von CSV- bzw. strukturierten TXT-Dateien

8.5.2.1.4. Schreiben von XML-Dateien

8.5.2.1.5. Schreiben von JSON-Dateien

8.5.2.2. Schreiben von Binär-Dateien

8.5.3. gepickelte Dateien – Dateien mit gemischten Daten

8.5.3.1. Schreiben von Dateien mit gemischten Daten

8.5.3.2. Lesen von Dateien mit gemischten Daten

8.6. Module

= Bibliothek
Sammlung vorgefertigter Programm-Teile (meist Funktionen)
praktisch Objekte (→ Objekt-orientierte Programmierung)

Probleme dann möglich, wenn im aktuellen Programm-Ordner schon eine Datei mit dem Namen des Moduls vorhanden ist, dann muss mit Fehler-Meldungen gerechnet werden
genau wenn man versucht seine eigene Datei mit dem Namen eines Moduls abzuspeichern, das geht zwar, aber der Aufruf der Module / Modul-Funktionen geht nicht → Fehler-Meldungen

vollständiger Import eines Moduls

```
import modul
wert = modul.funktion(10)
print(modul.funktion(20))
```

Funktionen müssen mit vorgesetztem Modul-Namen aufgerufen werden
Vorteile:
man kann eigene Funktionen und (globale) Variablen mit dem gleichen Namen im Programm händeln

Nachteile:
lästiges Mitschreiben des Modul-Namens

Import einzelner Funktionen eines Moduls

```
from modul import funktion
wert = funktion(10)
print(funktion(20))
```

Vorteile:
Funktion kann ohne Modul-Namen aufgerufen werden

Nachteile:
Aufruf **from … import** für jede einzelne Funktion oder für Gruppen notwendig

vollständiger Import eines Moduls als integraler Programmteil

```
from modul import *
wert = funktion(10)
print(funktion(20))
```

Vorteile:
keine selektiven Importe mehr

Nachteile:
es werden viele unnötige Funktionen importiert

Modul-Import mit Vergabe eines internen Namens

```
import modul as mo
wert = mo.funktion(10)
print(mo.funktion(20))
```

Vorteile:
keine selektiven Importe mehr
kürzere Modulschreibung möglich
Module mit gleichen internen Funktionen / Attributen lassen sich sauber trennen

Nachteile:
es werden viele unnötige Funktionen importiert

Anzeige der verfügbaren Funktionen und (globalen) Variablen

```
import math
print(dir(math))
```

```
>>>
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gam-
ma', 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Um Module auch in der Konsole nutzen zu können, müssen diese am Ende des Quell-Code's folgenden Konstrukt enthalten:

```
if __name__ == "__main__":
    …
```

8.6.1. "built-in"-Funktionen

Funktionen, die schon direkt im klassischen Python verfügbar sind
häufig gebraucht; weiterhin sollen sie schnell und Fehler-frei bzw. Fehler-unanfällig sein
in vielen anderen Programmiersprachen gehören sie gleich zum Befehls-Umfang dazu
in Python extra Module; dadurch etwas langsamer aber auch veränderlich / überschreibbar, wenn's denn wirklich notwendig ist

z.B. **max()**, **min()**, **abs()**, **type()**

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

8.6.2. wichtige interne Module

8.6.2.1. die Bibliothek math

ausführlich unter: https://docs.python.org/3/library/math.html

ausgewählte Konstanten

math.**pi**

math.**e**

ausgewählte Funktionen

math.**ceil(*wert*)**
rundet auf die nächstgrößere Ganzzahl bzw. auf Wert, wenn Wert eine Ganzzahl ist
Gegenstück ist math.floor()

math.**fabs(*wert*)**
liefert den Absolut-Wert zurück

math.**factorial(*wert*)**
liefert die Fakultät von Wert zurück

math.**floor(*wert*)**
rundet auf die nächstkleinere Ganzzahl bzw. auf Wert, wenn Wert eine Ganzzahl ist
Gegenstück ist math.ceil()

math.**fmod(*wert*)**
Modulo-Funktion bevorzugt für Gleitkommazahlen (sonst besser x **%** y verwenden

math.**frexp(*wert*)**
liefert die Mantisse und den Exponenten als Paar zurück

math.**gcd(*wert {, wert}*)**
liefert den größten gemeinsamen Teiler (GGT) der Werte zurück

math.**lcm(*wert {, wert}*)**
liefert das kleinste gemeinsame Vielfache (KGV) zurück

math.**perm(*wert*)**

liefert die Anzahl der Permutationen (Kombinations-Möglichkeiten von k Elementen aus den n Elementen) zurück

math.**trunc(*n, k=keine*)**
gibt den Nachkomma-Teil einer Gleitkommazahl zurück

math.**exp(*wert*)**
liefert den Funktions-Wert der Expotential-Funktion zu Wert zurück

math.**log(*wert [, basis]*)**
liefert den Funktions-Wert der natürlichen Logarithmus-Funktion zu Wert zurück, bei Bedarf kann eine zu e abweichende Basis angegeben werden

math.**log10(*wert*)**
liefert den Lorarithmus zur Basis 10 zurück

math.**pow(*wert, exponent*)**
liefert die Exponente Potenz von Wert zurück

math.**sqrt(*wert*)**
liefert die Quadrat-Wurzel zurück

math.**sin(*wert*)**
liefert den Sinus zu Wert zurück

math.**cos(*wert*)**
liefert den Cosinus zu Wert zurück

math.**tan(*wert*)**
liefert den Tangens zu Wert zurück

math.**asin(*wert*)**
liefert den Sinus zu Wert (gegeben in Bogenmaß) zurück

math.**acos(*wert*)**
liefert den Cosinus zu Wert (gegeben in Bogenmaß) zurück

math.**atan(*wert*)**
liefert den Tangens zu Wert (gegeben in Bogenmaß) zurück

math.**dist(*punkt1, punkt2*)**
liefert den EUKLIDischen Abstand zwischen den Punkten (mit Koordinaten) zurück

math.**degrees(*wert*)**
liefert den Winkel in Grad zum Wert in Bogenmaß zurück

math.**radians(*wert*)**
liefert den Winkel in Bogenmaß zum Wert in Grad zurück

8.6.2.2. die Bibliothek random

ausführlich unter:

8.6.2.x. Verschiedenes zum Modul: sys

ausführlich unter:

8.6.2.x. Verschiedenes zum Modul: time

ausführlich unter:

clock()
liefert einen Programm-internen Zeitstempel (Programm-Laufzeit) zurück
für Laufzeit-Messungen vergleicht man einfach die Zeitstempel vor und nach dem zu prüfenden Programm-Teil / Algorithmus / Funktions-Aufruf

```
from time import *

…

t0 = clock()
# hier steht dann der zu testende Quelltext
t1 = clock()

print("Laufzeit: ",t1-t0,"s")
```

```
>>>
```

time()
liefert Zeitstempel als Fließkommazahl
gut für genauere Zeit-Differenzen auch im ms-Bereich geeignet

```
import time

…

t0 = time.time()
# hier steht dann der zu testende Quelltext
t1 = time.time()

print("Laufzeit: ",t1-t0,"s")
```

```
>>>
```

ctime()
Zeitstempel wird als Text ausgegeben (mit Datum und Uhrzeit)
Die Formatierung orientiert sich an der Zeit-Anzeige in der Programmiersprache C.

```
import time

zeitstempel = time.ctime()

print(zeitstempel)
```

```
>>>
Wed Dec 16 17:25:59 2020
>>>
```

8.6.2.x. Verschiedenes zum Modul:datetime

ausführlich unter:

```
import datetime
```

oder auch:

```
import datetime as dttm
```

Abfrage des aktuellen Zeit-Stempels mit

```
dttm.datetime.now()
```

hat man auf das `as dttm` verzichtet, dann würde der Funktions-Aufruf so aussehen:

```
datetime.datetime.now()
```

mit Hilfe der str()-Funktion lässt sich aus dem Ergebnis-String eine lesbare / verständliche Ausgabe erzeugen
zusammen z.B.:

```
str(dttm.datetime.now().date())
```

entsprechend für Zeit:

```
str(dttm.datetime.now().time())
```

oder als Selektion der einzelnen Zeit-/Datum-Elemente:

```
.year(), .month(), .day(), .hour(), .minute(), .second(), .microsecond()
```

Der Zeitstempel kann also mittels integrierter Funktionen / Attribute in die Anteile zerlegt werden. Das ist dann wichtig, wenn nur bestimmte Zeit-Informationen gebraucht werden. Ein klassisches Problem ist z.B. die Angabe eines Zeitstempels in einem Datei-Namen. Zuerst ist dies scheinbar kein Problem, da z.B. ctime() ja einen String zurückliefert. Aber wie immer steckt der Teufel im Detail. Der String enthält in der Zeitangabe Doppelpunkte. Diese sind aber nicht in Dateinamen zugelassen.

```
import datetime as dttm

zeitstempel = dttm.datetime.now()

print("Zeitstempel: ", zeitstempel)
print()
print("Stunden : ",zeitstempel.hour)
print("Minuten : ",zeitstempel.minute)
print("Sekunden: ",zeitstempel.second)
```

```
>>>
Zeitstempel:  2020-12-16 17:45:48.235872
```

```
Stunden :  17
Minuten :  45
Sekunden:  48
>>>
```

Wahrscheinlich ist das erneute Zusammensetzen eines Zeitstemples aus den Bestandteilen des datetime-Zeitstempels die flexibelste Variante. Hier kann man die gewünschten Bestandteile frei auswählen:

```
import datetime as dttm

zeitstempel = dttm.datetime.now()

print("Zeitstempel: ",zeitstempel)
print("Stunden : ",zeitstempel.hour)
print("Minuten : ",zeitstempel.minute)
print("Sekunden: ",zeitstempel.second)

zeitstempel_neu = ""
zeitstempel_neu += str(zeitstempel.hour)+"-"
zeitstempel_neu += str(zeitstempel.minute)+"-"
zeitstempel_neu += str(zeitstempel.second)

print()
print("Zeitstempel: ",zeitstempel_neu)
```

```
>>>
Zeitstempel:  2020-12-16 19:34:48.928490
Stunden :  19
Minuten :  34
Sekunden:  48

Zeitstempel:  19-34-48
>>>
```

Deutlich kürzer ist die Umwandlung des Zeitstempels in einen String und dann nachfolgendes Slicing und Zusammensetzen der Elemente unter Herausschneiden der Doppelpunkte.

```
zeitstempel = str(zeitstempel)
zeitstempel_neu = zeitstempel[:13]+"-"
                  +zeitstempel[14:16]+"-"+zeitstempel[17:19]
```

Aufgaben:

1. *Erstellen Sie einen neuen Zeitstempel für einen Dateinamen aus einer aktuellen Tageszeit einschließlich Millisekunden!*
2. *Erstellen Sie eine Funktion, die aus einem datetime-Zeitstempel einen Zeitstempel-String erzeugt, der ein beliebiges Trennzeichen zwischen den Datums- bzw. Zeit-Bestandteilen benutzt!*
3.

8.6.2.x. Verschiedenes zum Modul: os

ausführlich unter:

Ermitteln des aktuell verfügbaren / freien Speichers

für Linux-basierte Systeme (über Shell-Aufruf)

```
import os

…

speicher = os.popen('df –m | grep rootfs | cut –c33-42').readline()
print("freier Speicher: ", int(speicher)," Byte")
```

```
>>>
```

die intern verfügbaren Module der Standard-Python-Installation (V. 3.5.0.)

```
AutoComplete          _pickle               enum                  pydoc_data
AutoCompleteWindow    _pyio                 errno                 pyexpat
AutoExpand            _random               faulthandler          pygame
Bindings              _sha1                 filecmp               queue
CallTipWindow         _sha256               fileinput             quopri
CallTips              _sha512               fnmatch               random
ClassBrowser          _sitebuiltins         formatter             re
CodeContext           _socket               fractions             reprlib
ColorDelegator        _sqlite3              ftplib                rlcompleter
Debugger              _sre                  functools             rpc
Delegator             _ssl                  gc                    run
EditorWindow          _stat                 genericpath           runpy
FileList              _string               getopt                sched
FormatParagraph       _strptime             getpass               select
GrepDialog            _struct               gettext               selectors
HyperParser           _symtable             glob                  setuptools
IOBinding             _testbuffer           gzip                  shelve
IdleHistory           _testcapi             hashlib               shlex
MultiCall             _testimportmultiple   heapq                 shutil
MultiStatusBar        _thread               hmac                  signal
ObjectBrowser         _threading_local      html                  site
OutputWindow          _tkinter              http                  smtpd
ParenMatch            _tracemalloc          idle                  smtplib
PathBrowser           _warnings             idle_test             sndhdr
Percolator            _weakref              idlelib               socket
PyParse               _weakrefset           idlever               socketserver
PyShell               _winapi               imaplib               sqlite3
RemoteDebugger        abc                   imghdr                sre_compile
RemoteObjectBrowser   aboutDialog           imp                   sre_constants
ReplaceDialog         aifc                  importlib             sre_parse
RstripExtension       antigravity           inspect               ssl
ScriptBinding         argparse              io                    stat
ScrolledList          array                 ipaddress             statistics
SearchDialog          ast                   itertools             string
SearchDialogBase      asynchat              json                  stringprep
SearchEngine          asyncio               keybindingDialog      struct
StackViewer           asyncore              keyword               subprocess
ToolTip               atexit                lib2to3               sunau
TreeWidget            audioop               linecache             symbol
UndoDelegator         base64                locale                symtable
WidgetRedirector      bdb                   logging               sys
WindowList            binascii              lzma                  sysconfig
ZoomHeight            binhex                macosxSupport         tabbedpages
__future__            bisect                macpath               tabnanny
__main__              builtins              macurl2path           tarfile
_ast                  bz2                   mailbox               telnetlib
_bisect               cProfile              mailcap               tempfile
_bootlocale           calendar              marshal               test
_bz2                  cgi                   math                  textView
_codecs               cgitb                 mimetypes             textwrap
_codecs_cn            chunk                 mmap                  this
_codecs_hk            cmath                 modulefinder          threading
_codecs_iso2022       cmd                   msilib                time
_codecs_jp            code                  msvcrt                timeit
_codecs_kr            codecs                multiprocessing       tkinter
_codecs_tw            codeop                netrc                 token
_collections          collections           nntplib               tokenize
_collections_abc      colorsys              nt                    trace
_compat_pickle        compileall            ntpath                traceback
_csv                  concurrent            nturl2path            tracemalloc
_ctypes               configDialog          numbers               tty
_ctypes_test          configHandler         opcode                turtle
_datetime             configHelpSourceEdit  operator               turtledemo
_decimal              configSectionNameDialog optparse              types
_dummy_thread         configparser          os                    unicodedata
_elementtree          contextlib            parser                unittest
_functools            copy                  pathlib               urllib
_hashlib              copyreg               pdb                   uu
```

```
_heapq              crypt               pickle              uuid
_imp                csv                 pickletools         venv
_io                 ctypes              pip                 warnings
_json               curses              pipes               wave
_locale             datetime            pkg_resources       weakref
_lsprof             dbm                 pkgutil             webbrowser
_lzma               decimal             platform            winreg
_markerlib          difflib             plistlib            winsound
_markupbase         dis                 poplib              wsgiref
_md5                distutils           posixpath           xdrlib
_msi                doctest             pprint              xml
_multibytecodec     dummy_threading     profile             xmlrpc
_multiprocessing    dynOptionMenuWidget pstats              xxsubtype
_opcode             easy_install        pty                 zipfile
_operator           email               py_compile          zipimport
_osx_support        encodings           pyclbr              zlib
_overlapped         ensurepip           pydoc
```

Für eine schnelle Hilfe kann folgender Link benutzt werden. Dabei wird random durch den Namen der Bibliothek ersetzt:

https://docs.python.org/3/library/random.html

ansonsten als Einsprung-Punkt: https://docs.python.org/3/ nutzen

für die Bibliotheken ist der Index: https://docs.python.org/3/library/index.html

Wer mal richtig schmunzeln möchte, schaut sich die google-Übersetzung der Seiten an! Da werden die Grenzen einfacher Übersetzungs-Systeme sehr offensichtlich.

8.6.3. externe Module installieren und nutzen

8.6.3.x. Package-Installer PIP

Für das Installieren von Paketen (Bibliotheken, Modulen, …) gibt es Python das Tool PIP.
Es wird in der Eingabeaufforderung / Konsole bedient. Soll in ein aktuelles System ein zusätzliches Paket installiert werden, dann wechselt man in der Konsole ins Installations-Verzeichnis von Python.
Eine gute Hilfe für die nicht mehr DOS-mächtigen Konsolen-Benutzer ist ein verstecktes feature von Windows. Im Windows-Explorer klickt man bei gedrückter [⇧]-Taste auf die rechte Maus-Taste. Im nun angezeigten Kontext-Menü findet man auch den Menü-Punkt: "Eingabeaufforderung hier öffnen".
Sollte dieser Menü-Punkt nicht erscheinen, dann geht auch die folgende Schrittfolge:

1. Öffnen des Windows-Explorer's und Auswählen des übergeordneten Ordner's (bezogen auf den Ziel-Ordner)
2. Öffnen der Konsole
3. Eintippen des Befehls: cd
4. Ziehen des Ziel-Ordner-Symbols aus dem Windows-Explorer in die Konsole

→ der vollständige Pfad steht jetzt hinter dem cd-Befehl und kann ausgeführt werden

Ev. muss noch ein Laufwerks-Wechsel mittels Laufwerk-Buchstabe und angehängtem Doppelpunkt gemacht werden. In der Konsole wird dann mit:

```
pip install Paketname
```

das angegebene Paket installiert. Es werden diverse Verlaufs-Informationen angezeigt und beim ordnungs-gemäßen Durchlauf auch eine Erfolgs-Bestätigung.
Zum Deinstallieren verwendet man:

```
pip uninstall Paketname
```

Das ist aber eigentlich in speziellen Situationen notwendig.
Läuft das Python-System schon länger, dann sollte man das PIP-Programm zuerst einmal selbst updaten:

```
pip install –upgrade pip
```

Das ist auch eine Option, falls eine Paket-Installation nicht klappt. Oft geht es dann mit der aktuellen PIP-Version.
Die Anzeige der installierten Pakete erfolgt mit:

```
pip list
```

Sollen (ver)alte(ter) Paket angezeigt werden und die zugehörigen neuerenVersionen, dann hilft das Kommando:

```
pip list --outdated
```

Vielleicht ist der genaue Paketname nicht bekannt, oder die Versionen überschlagen sich, dann ist eine Suche nach bestimmten Paketen mit:

```
pip search "Anfrage"
```

möglich.

8.6.4. Modul / Bibliothek NumPy

ausführlich unter: https://numpy.org/doc/stable/

Die wichtigste Eigenschaft von NymPy ist wohl die bereitstellung von Feldern / Array's für mathematische Aufgaben in Python. Die üblichen Listen des originalen Python sind für aufwendige mathematische Anwendungen einfach zu sperrig und zu langsam.

NumPy bietet viele Möglichkeiten seine Array's intern zu verknüpfen oder Funktionen darüber laufen zu lassen. Diesen direkten Weg sollte man immer der eigenen Interation über die Array's vorziehen. Die NumPy-Umsetzungen sind deutlich schneller.

Importieren der Bibliothek

```
import numpy
```

oder etwas praktischer mit einem verkürzten Namen für die Bibliothek. Dabei hat sich in der Programmierer-Welt np eingebürgert.

```
import numpy as np
```

Im Folgenden gehen wir genau von diesem Import aus.

Erstellen von Array's

Ein einfache Array (Feld) lässt sich über:

```
datenSet = np.array([4,3,6,2,7,2,2,3,4,4])    # 10 Daten-Punkte
```

erzeugen. In der Array- oder Matrix-Sprache handelt es sich um eine ein-dimensionales Feld oder einen sogenannten Vektor. Auch der nächste Erstellungs-Befehl erzeugt einen solchen Vektor:

```
datenSet = np.arange(12) #liefert aufsteigend belegtes Feld (von 0 bis 11)
```

Viele Daten liegen praktisch oder im Modell als mehr-dimensionale Struktur vor. Gerade hierfür eigenet sich NumPy besonders. Ein mehr-dimensionales Array aus konkreten Daten(-Listen) erstellt man so:

```
datenSet = np.array([[4,3,6,2,7,2,2],    # 7 Beobachtungen
                     [7,2,6,5,4,3,2],
                     [3,2,6,4,3,2,1]])  # in 3 Beobachtung(s-Reih)en
print(datenset.shape)    # liefert Dimensionen des Array's
print(np.max(datenSet, axis=1 – np.min(datenSet,axis=1))  #zeigt Spanbreite
                                  #der Beobachtungen innerhalb einer Reihe
```

Initialisieren eines leeren Array's

```
datenSet = np.empty([3,4,7])   # leeres 3-dim. Feld 3x4x7
```

Der Standard-Datentyp ist bei NumPy float.
Werden aber andere Daten vorgegeben und die Erstellungs-Funktion asarray() benutzt, dann "errät" NumPy den passenden Datentyp.

Initialisieren eines Array's mit Nullen (Null-Matrix)

hier 2-Dim-Matrix

```
matrix = np.zeros((3,4))
```

Auch wenn es nicht scheint, die Nullen sind allesamt float-Werte! Das muss ev. bei Berechnungen usw. beachtet werden.

Initialisieren eines Array's mit Nullen (Null-Matrix)

hier 4-Dim-Matrix

```
matrix = np.ones((3,4,2,3))
```

Auch hier sind die Einsen vom Datentyp float.

Initialisieren eines Array's mit Zufalls-Zahlen

Vektor mit 10 Zufalls-Zahlen zwischen 0 und 1

```
matrix = np.random.rand(10)
```

Matrix mit deb Dimensionen von 4 Spalten und 3 Zeilen sowie ganzzahligen Werten zwischen 4 und 10 (praktisch also obere Grenze: 11):

```
matrix = np.random.randint(4,11,(3,4))
```

Daten aus Dateien einlesen

Nichts ist nerviger als Daten beim Testen eines Programm's ständig per Hand einzugen. Natürlich kann man sich ein Daten-Set direkt in den Quell-Text legen, aber von schöner und universeller ist das Laden von Daten aus Dateien.

```
dateiname = "eingabedaten.csv"
datenset = np.genfromtxt(dateiname)
```

Neben dem Dateinamen können Komma-getrennt auch noch merhrere Optionen angegeben werden. Besonders interessant sind dabei das Trennzeichen zwischen den Daten-Elementen in der Zeile. Dies wird mit delimiter festgelegt. Bei einer Semikolon-getrennten CSV-Datei würde man dann `delimiter=";"` verwenden. Soll die erste Daten-Zeile überlesen werden, weil sie – wie häufig vorkommend – Überschriften enthält, dann kann dies mit skip_header=1 eingestellt werden.
Der Dateiname ist vom Typ her offen. Man kann also auch gerne Datei-Typ-Kürzel wie .IN, .OUT oder .TXT benutzen. Selbst wenn man es .BMP nennen würde, ist dies ok. Nur sollte man damit rechnen, dass dann Objekt-bezogene Aufrufe durch das Betriebssystem oder eine BMP-verarbeitendes Programm schief gehen werden. Intern ist und bleibt die Datei eine Text-Datei.
Die Speicherung eines Array's in eine Text-Datei erfolgt über:

```
np.savetxt(dateiname)
```

Hier sind ebenfalls wieder diverse Optionen zulässig.
Mit Hilfe der Funktionen .tofile(dateiname) und .fromfile(dateiname) lassen sich NumpY-Array's auch in Binär-Form speichern bzw. laden.

Zugriff auf Daten-Elemente

Der Zugriff auf einzelne Elemnte erfolgt, wie üblich über die Indicies in eckigen Klammern:

```
elem = datenset[3,2]
```

Mittels Slicing lassen sich Bereiche aus einem größeren Array herausholen. Dabei wird für jede Dimension ein eigener Slicing-Ausdruck verwendet. Es gilt die übliche Notierung `start:ende:schrittweite`.

Operationen / Funktionen mit / zu Array's

Multiplikation eines Vektor's / einer Matrize mit einem Faktor

```
import numpy as np
vektor = np.array([12.3, 53.6, 31.5, 33.3])
vektor = vektor * faktor
print(vektor)
```

Multiplikation einer Matrize mit einem Vektor

```
matrix = np.array([[2,3,4,5],
                   [4,5,6,7],
                   [6,7,8,9]] )
vektor = np.array([2,6,4,3])
erg = np.dot(vektor,matrix)
print(erg)
```

Multiplikation einer Matrize mit einer anderen

```
matrixA = np.array([[2,3,4,5],
                    [4,5,6,7]] )
matrixB = np.array([[1,2],
                    [2,3],
```

```
                    [3,4],
                    [4,5]] )
erg = np.dot(matrixA,matrixB)
print(erg)
```

Vergleich von zwei NumPy-Array's und speichern des Ergebnis in einem Array mit BOOLE-schen Werten, z.B.:

```
vergl_erg = NumPyArray1 < NumPyArray2
```

Lineare Algebra (z.B. Lösen von Gleichungs-Systemen)

Ein lineares Gleichungs-System lässt sich in die Matrizen-Welt übersetzen. Dabei ergibt sich eine Matrix (praktisch ein Vektor) für die Ergebnisse (hier Schreibung links) und dem Variablen-Teil.

$$\begin{array}{rcrcrcrcr} -8 & = & 3 & x_1 & + & -1 & x_2 & + & 2 & x_3 \\ 2 & = & & & & 2 & x_2 & + & 2 & x_3 \\ 0 & = & 4 & x_1 & & & & + & 1 & x_3 \end{array}$$

Die Faktoren vor den Variablen x_1 bis x_3 bilden eine zwei-dimensionale Matrix. Dabei ist zu beachten, dass jedes Element eine numerisch verwertbare Zahl ist. Die nicht benutzten Variablen erhalten als Faktor eine 0 und die ohne Faktor den Faktor 1.

$$\begin{bmatrix} -8 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 & , & -1 & , & 2 \\ 0 & , & 2 & , & 2 \\ 4 & , & 0 & , & 1 \end{bmatrix}$$

```
faktoren = np.array([[3,-1,2],  # Matrix muss quadr. u. vollst. sein
                     [0,2,2]    # keine Vielfache anderer Zeilen zulässig
                     [4,0,1]])  # Determinante darf nicht 0 sein
ergebnisse = nparray([-8,2,0])  # quasi y-Werte

variablen = np.linalg.solve(faktoren,ergebnisse)
print(variablen)
```

interessante Links:
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf (→ Cheat Sheet zu NumPy)
https://riptutorial.com/de/numpy (Tutorial zu NumPy)

8.6.5. Modul / Bibliothek MatPlotLib

aktuelle Dokumentation unter https://matplotlib.org/

Was dem Standard-Python fehlt, sind Befehle, um graphische Elemente auf dem Bildschirm darzustellen. Für viele Berechnungen usw. würden wir uns aber gerne eine Ausgabe als Diagramm od.ä. wünschen. Zwar kann man sich für den Notfall auch mit Pseudographiken helfen, die auf den Konsolen-Symbolen basieren. Für Facharbeiten usw. reicht sowas natürlich nicht. Hier brauchen wir professionelle Diagramme.
Die Bibliothek MatPlotLib bietet viele Funktionen zum einfachen Erstellen und Gestalten von Diagrammen. Diese werden in einem separaten Graphik-Fenster angezeigt.
Praktisch ist das Modul MatPlotLib in reichlich Untermodule geteilt. Für unsere Zwecke – der einfachen Erstellung anspruchvoller 2D-Diagramme – brauchen wir das Untermodul **PyPlot**.
Hauptziel-Richtung von PyPlot ist die Präsentation von Daten in einer MATLAB-ähnlichen Umgebung. Dabei wird derzeit auf 2D-Diagramme orientiert.
Da es sich bei MatPlotLib um eine externe Bibliothek handelt, kann eine Installation notwendig sein.
Einige Entwicklungs-Umgebungen – wie z.B. Anaconda, WinPython und ActiveState – haben MatPlotLib gleich mitinstalliert. Bei der klassischen Python-Installation (mit dem Installations-Programm von python.org) ist die Bibliothek nicht dabei.
Ob **MatPlotLib** verfügbar ist, merkt man sofort nach einem Import der Bibliothek (s.a. sonst auch weiter unten). Dazu reicht es in der Python-Konsole ein:

```
import matplotlib
```

einzugeben. Kommt keine Fehlermeldung, dann ist MatPlotLib schon integriert. Ansonsten müssen wir es nachinstallieren.
Die Installation erfolgt auf der Konsole (Eingabeaufforderung, ms-dos-Fenster, PowerShell-Fenster).

Für viele Windows-Nutzer gestaltet sich innerhalb der Konsole ein Verzeichniswechsel schwierig, weil die alten ms-dos-Befehle (**cd *Pfad oder Verzeichnis*** und Laufwerkswechsel mit Doppelpunkt) nicht mehr recht geläufig sind. Hier bietet sich ein kleiner Trick an:

1. Wählen Sie im Arbeitsplatz / Windows-Explorer / Datei-Manager / … das Laufwerk und den Ordner aus, in dem Python installiert wurde.
2. Wenn Sie im rechten Fenster-Bereich den Ordner Python sehen, dann klicken Sie bei gedrückter Hochstell-Taste mit der rechten Maus-Taste drauf.
3. Im nun sichtbaren erweiterten Kontext-Menü sieht man – je nach Betriebssystem – eine Möglichkeit die Eingabeaufforderung oder die PowerShell zu öffnen.

In der geöffneten Konsole müssen wir nun Text-basiert einige Befehle eingeben, um unser Python-System auf den neuesten Stand zu bringen.

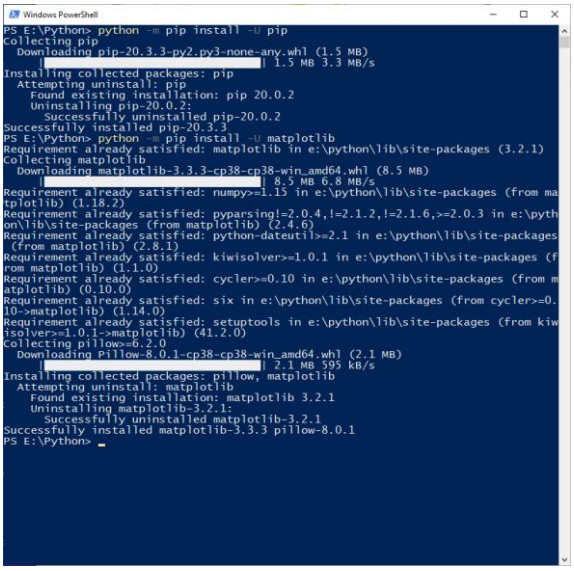
Zuerst aktualisieren wir das Update-Programm **pip** selbst:

```
python –m pip install –U pip
```

Es folgen u.U. mehrere Installations-Vorgänge (s.a. Abb. rechts). Wenn dann der Konsolen-Prompt wieder angezeigt wird, dann installieren / updaten wir MatPlotLib mittels:

```
python –m pip install –U matplotlib
```

und wieder folgt im Allgemeinen eine Reihe von Installationen.



8.6.5.1. allgemeines Vorgehen (Workflow)

Die Diagramme werden in Python Plot's oder Figuren genannt. Je nach dem werden entsprechend benannte Variablen benutzt. Diese können aber beliebig ausgetauscht werden. Nur für den Austausch mit anderen Programmierern usw. sind standardisierte Namen eher sinnvoll.

Arbeitsschrittfolge für die Diagramm-Erstellung

- **Vorbereiten der Daten**

  Daten in Listen oder NumPy-Array zusammenstellen / berechnen / …
  - 1D: aufgezählte y-Werte
  - 2D: x- und y-Werte oder Bilder

- **Importieren der Bibliothek (Modul + Untermodul)**

  nur einmalig (zu Beginn des Programms / Modul's)
  `import matplotlib.pyplot as plt`
  plt kann durch anderen Namen ersetzt werden, hat sich aber so eingebürgert

- **Erzeugen eines Plot's**

  Initialisierung eines Zeichen- / Diagramm- / Plot-Objektes (auch Figur genannt)
  `plt.plot( …)`
  z.B. möglich:
  - klassische X-Y-Diagramme
  - Histogramme
  - Balken-Diagramme
  - Kreis-Diagramme
  -

- **Formatieren / Anpassen eines Plot's**

  optional
  viele Anpassungen / Festlegungen werden klassischerweise schon gleich beim Erzeugen des Plot's gemacht
  zusätzlich z.B. Hinzufügen von:
  - Legende
  - Titel
  - Achsen
  -

- **Speichern des Plot's**

  optional
  Diagramm als Bild-Datei speichern

- **Anzeigen eines Plot's**

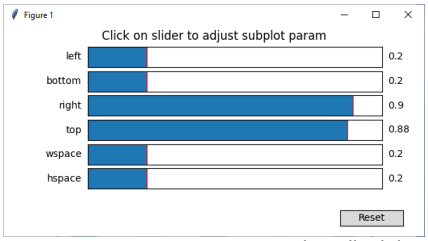
  erst mit
  `plt.show()`
  wird das Zeichen-Objekt / das Diagramm / der Plot / die Figur angezeigt (vorher nur Speicher-Objekt!)
  nach Änderungen ist ein erneuter Aufruf notwendig!

- **Schließen eines Plot's**

  innerhalb des Programm's mit:
  `plt.close()`
  ansonsten wird Plot am Ende des Programm's automatisch geschlossen

Die Plot's / Diagramme werden klassischerweise in qaudratischen Boxen (Figuren) angezeigt. In diesen stehen einige Bedien-Elemente für eine interaktive Anpassung zur Verfügung. Die Schaltflächen unten sind weitesgehend selbsterklärend.
Mit dem Haus gelangt mann nach irgendwelchen Manipulationen immer wieder zur ursprünglichen Anzeige zurück. Man kann die Diagramm-Fläche verschieben sich hineinzoomen und / oder auch die Grafik (mit all ihren Parametern) einstellen.
Das letzte Button lässt eine Speicherung der Figur in einer PNG-Datei zu. Gerade für Dokumentations-Zwecke ist das sehr praktisch.

Aktivitäts-Schaltflächen
in der Diagramm-Anzeige



Einstell-Dialog

weitere Möglichkeiten:
Die quadratische Grund-Figur lässt sich kleinere Unter-Figuren – also Unter-Diagramme – unterteilen.
Mit der Funktion **imshow(…)** lassen sich Bilder mit quadrat-förmiger Ausdehnung anzeigen. Dies wird z.B. gern für die Anzeige von Bilder aus Trainings-Set's für Programme zu Künstlichen Neuronalen Netzen (KNN, → (🕮 **Programmieren mit Python Teil 3: für Experten** → 10.8. Python und Data Science + 10.9. Python und Künstliche Intelligenz) genutzt.

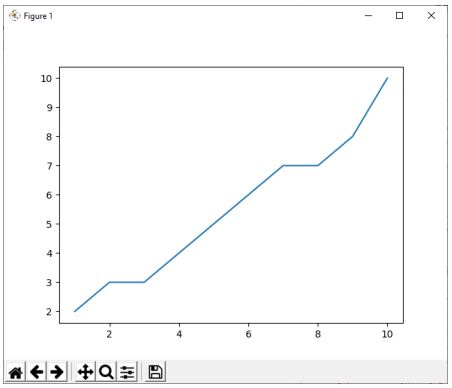
8.6.5.2. Erstellen und Manipulieren von Diagrammen

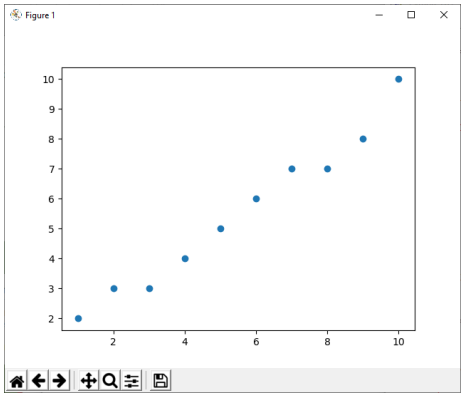
8.6.5.2.1. Entscheidung für einen Diagramm-Typ

Entsprechend der vorliegenden Daten muss in einem ersten Schritt festgelegt werden, welchen Diagramm-Typ man benutzen möchte. Hier sind natürlich die allgemeinen Regeln für die Auswahl zu beachten. Der Diagramm-Typ muss zu den Daten passen.

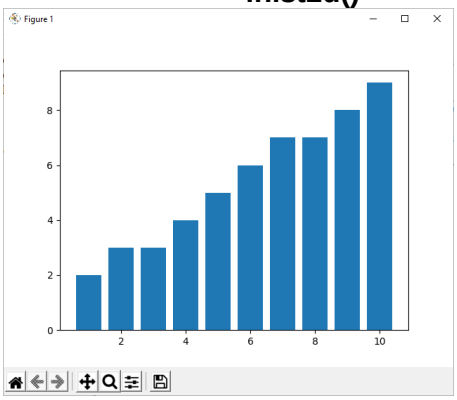
mögliche Diagramm-Typen (Auswahl klassischer Typen)

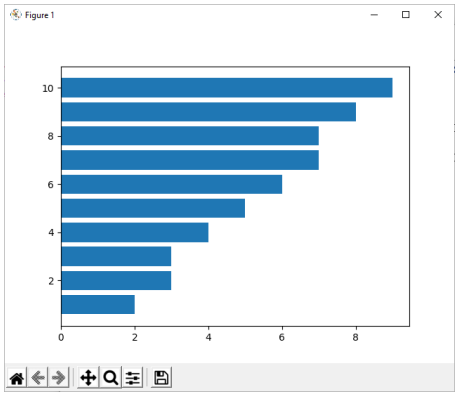
- **Linien-Diagramm** **.plot()**
- **X-Y-Punkt-Diagramm** **.plot(… , 'o')**
- **Zeit-Diagramme** **.plot_date()**

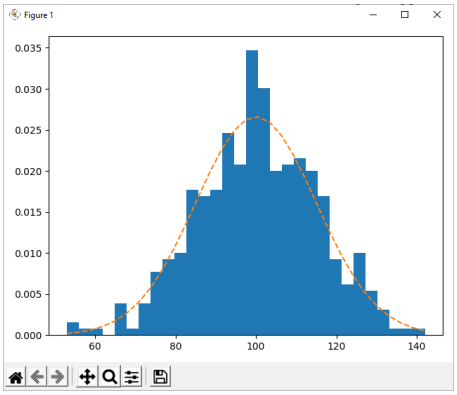




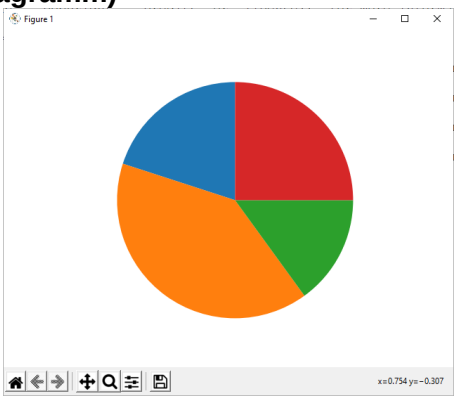
- **Säulen-Diagramm** **.bar()**
- **Balken-Diagramm** **.barh()**
- **Histogramm** **.hist()**
  .hist2d()

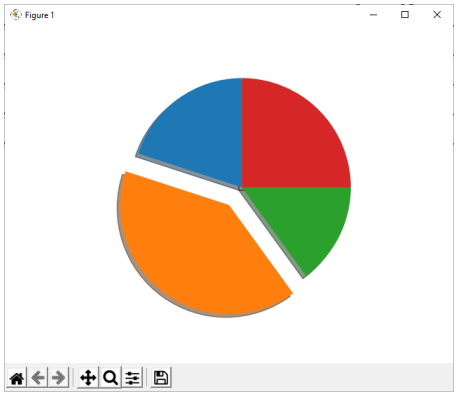




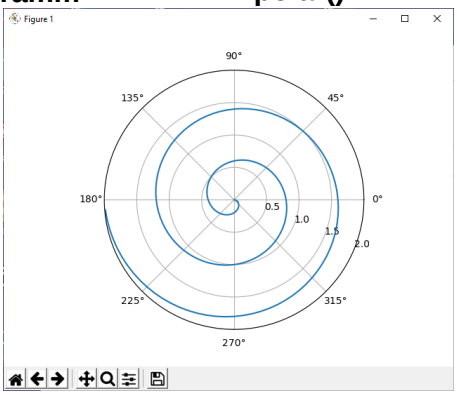


- **Kreis-Diagramm .pie()**
  (Torten-Diagramm)





- **Polar-Diagramm .polar()**



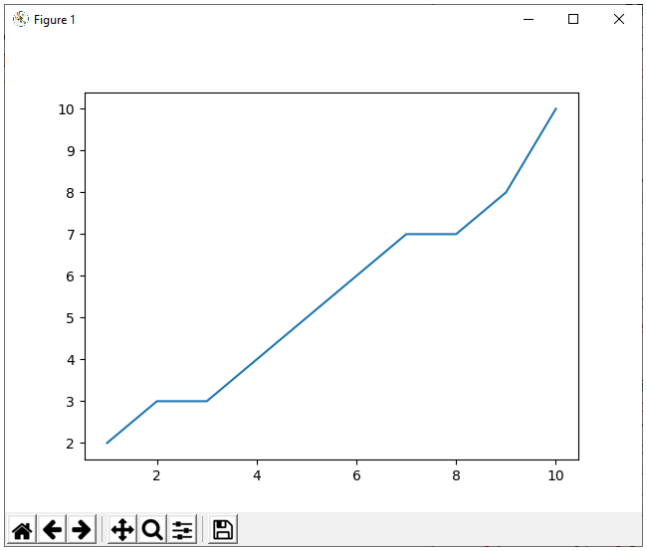
-

Um ev. später mit den Diagramm-Typen experimentieren zu können und vielleicht auch den Quell-Code wiederzuverwenden, empfiehlt es sich, die Daten in Variablen zu speichern.
Die Daten werden in den meisten Fällen als Listen bzw. NumPy-Array's erwartet.
Zuerst werden wir hier nur die Linien- bzw. die recht ähnlichen Punkt-Diagramme besprechen. An diesen zeigen wir dann auch die Formatierungs-Möglichkeiten auf (→ Diagramm gestalten / formatieren). Die anderen – oben erwähnten – Diagramm-Typen sowie einige außergewöhnliche Diagramme betrachten wir hinterher in einer komplexeren Form, ohne dabei betont Erstellung und Formatierung zu trennen (→ weitere Diagramm-Typen).
Vielfach erwähnen wir nur den Funktions-Namen. Die Optionen und Gestaltungs-Möglichkeiten sind für ein Skript, wie dieses, viel zu speziell. Der genötigte Anwender wird sich über die Hilfe-Seiten von MatPlotLib aber schnell einarbeiten.

Erstellen eines Linien-Plots (aus einfacher Daten-Reihe)

```
import matplotlib.pyplot as plt
werte = [2,3,3,4,5,6,7,7,8,10]
plt.plot(list(range(1,11)), werte)
plt.show()
```

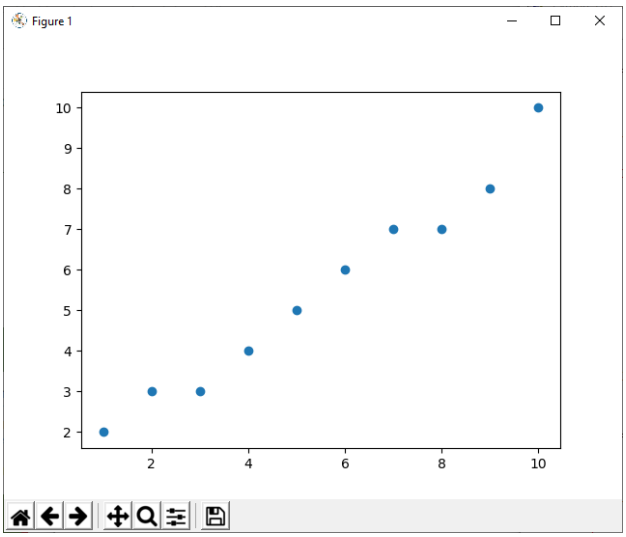
Der obige Quell-Text erezugt ein einfaches Linien-Diagramm (s.a. Abb. rechts), bei dem nur eine Werte-Liste genutzt wird. Die Werte bekommen über die range()-Funktion Nummern für die x-Achse.
Zu beachten ist hier, dass die Daten immer in Listen-Form bereitgestellt werden müssen.



Durch eine einfache Options-Angabe (hier: 'o'), kann aus dem Linien-Diagramm das elementare Punkt-Diagramm gemacht werden. Ob das Verbinden der Punkte über eine Linie überhaupt zulässig war, muss vorher geprüft werden.

```
plt.plot(list(range(1,11)), werte, 'o')
```

Spätestens, wenn mehrere Punkt-Reihen dargestellt werden sollen, muss man sich um die Formatierung der Daten-Punkte kümmern. Als Formatierungs-Möglichkeiten für Daten-Punkte sind unterschiedliche Symbole und auch Farben möglich.
Bei den oben vorgestellten Linien-Diagrammen kann bei der Formatierung auf unterschiedliche Strich-Stärken und Linien-Arten zurückgegriffen werden (→ 8.6.5.2.3. Diagramm gestalten / formatieren).



Aufgaben:

1. *Erstellen Sie für die nochfolgenden Daten einer Reihe ein Linien-Diagramm!*

   5, 4, 4, 3, 3, 4, 5, 6, 7, 8, 7, 8, 8, 6, 4, 3

2. *Verwenden Sie die nachfolgenden Daten zum Erstellen eines Punkt-Diagramm's!*

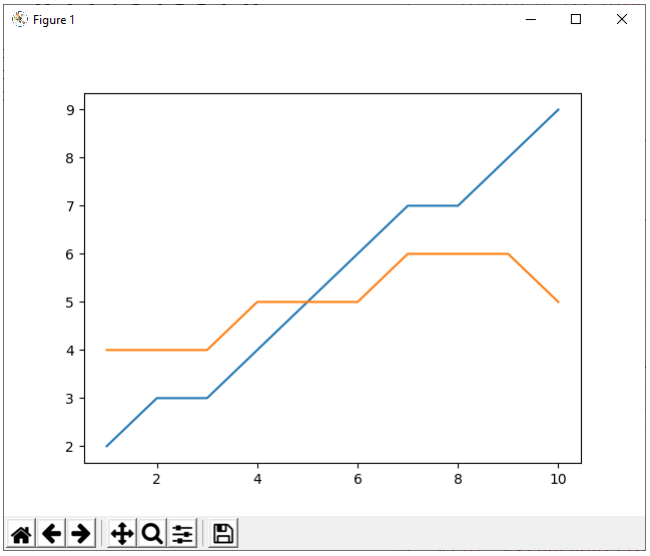
   2,3; 3,1; 2,8; 3,0; 4,0; 4,6; 3,9; 4,1

(c,p) 2015 - 2022 lsp: dre

Kombination von Daten-Reihen

```
import matplotlib.pyplot as plt
werte1 = [2,3,3,4,5,6,7,7,8,9]
werte2 = [4,4,4,5,5,5,6,6,6,5]
plt.plot(list(range(1,11)), werte1)
plt.plot(list(range(1,11)), werte2)
plt.show()
```

Linien- und Punkt-Diagramme lassen sich auch kombinieren. Für den entsprechend veränderten Graphen wird einfach die passende Option angegeben.

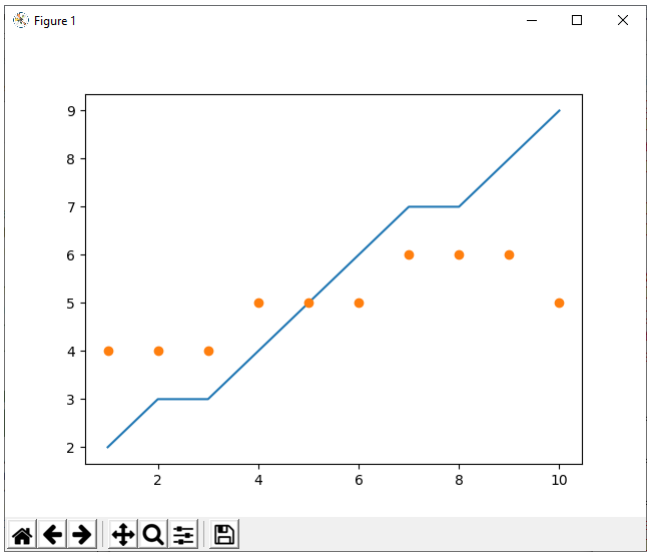



```
plt.plot(list(range(1,11)), werte2, 'o')
```

Desweiteren lassen sich die einzelnen Daten-Reihen auch in einem plot-Befehl vereinen:

```
x_werte = list(range(1,11))
plt.plot(x_werte, werte1,
         x_werte, werte2, 'o')
```

Die Anzahl der Paare von x- und y-Werten ist praktisch nur durch die Übersichtlichkeit des Plot's begrenzt.



Aufgaben:

1. *Erstellen Sie ein Punkt-Diagramm aus der folgenden Daten-Reihe!*
   2, 4, 6, 8, 10, 10, 8, 6, 4, 2, 4, 6, 8, 10, 10, 8, 6
2. *Erstellen Sie aus der gegebenen Daten-Reihe (von Aufgabe 1) ein kombiniertes Punkt- und Linien-Diagramm!*
3. *Lassen Sie sich ein Diagramm für die quadratische Funktion (für x nur natürliche Zahlen) anzeigen. Der Definitions-Bereich soll von 0 bis 10 gehen.*

8.6.5.2.2. Sichern der Diagramme

Das Abspeichern der Plot's gelingt ganz einfach aus der Anzeige heraus. Das Disketten-Symbol steht hier für die Möglichkeit eine Graphik-Datei in verschiedenen Formaten zu erzeugen. Der klassische Datei-Typ wird wohl PNG sein.
Man kann aber auch direkt aus dem eigenen Programm heraus eine Speicherung auslösen.

Plot Programm-gesteuert abspeichern

```
plt.savefig(dateiname, format='png')
```

Zu beachten sind mehrere Sachen. Der Aufruf der Speicher-Funktion muss vor der show()-Funktion erfolgen. Wahrscheinlich ist der Befehl als vorletzte Aktion (vor dem show()) am universellsten.
Zum Zweiten werden Dateien, die schon existieren, ohne Nachfrage überschrieben. Wenn also mehrere Diagramme in einem Programm gespeichert werden sollen, dann müssen die Dateinamen entsprechend angepasst werden.
Dies kann z.B. durch den Einbau eines Datums-Zeit-Stempels im Diagramm-Namen gelöst werden. Wie man hier vorgehen kann wurde schon beim Modul datetime besprochen (→ 8.6.2.x. Verschiedenes zum Modul:datetime).
Desweiteren ist die Format-Vorgabe entscheidend. Es geht zwar auch folgender Aufruf:

```
plt.savefig("Diagramm.jpg", format='png')
```

aber die angebliche JPG-Datei ist und bleibt eine PNG-Datei, auch wenn sie anders heißt.
Für viele Zwecke benötigt man Bilder / Bild-Dateien mit einem transparenten Hintergrund. Dann passen die besser zum Basis-Dokument. Der Transparenz-Modus wird über die Option transparent gesteuert.

```
plt.savefig("Diagramm.png", format='png', transparent=True)
```

Aufgaben:

1. *Passen Sie ein Diagramm-Programm so an, dass das erstellte Diagramm als "MeinDiagramm.png" gespeichert wird!*
2. *Überlegen Sie sich eine Möglichkeit, wie Sie in den Dateinamen einen Zeit-Stempel einbauen können! Realisieren ein passendes Programm!*
   Hinweis: Zeit-Stempel sind über das Modul time erreichbar.
3.

8.6.5.2.3. Diagramm gestalten / formatieren

Jeder Linie kann eine individuelle Strich-Art und Farbe zugewiesen werden. Die Daten-Punkte können ebenfalls ganz speziell festgelegt werden. Das erleichtert das Erkennen von Graphen in komplexeren Diagrammen. Die speziellen Merkmale werden als Zeichen-Folge im Options-String gesammelt, der Komma-getrennt an die Daten-Paare angehängt wird.

Linien-Arten formatieren

```
…
plt.plot(range1,11), werte2, ':')
…
```

Linienarten:

'-' durchgezogen	':' gepunktet
'--' gestrichelt	'-.' Strich-Punkt-Linie

Marker / Daten-Punkte formatieren

oft in Kombination mit Linien-Art

```
…
plt.plot(range1,11), werte1,'o—')
plt.plot(range1,11), werte2,'h:')
…
```

oder auch in Kombination mit einer Linien-Farbe

```
…
plt.plot(range1,11), werte1,'o—r')
…
```

Marker / Daten-Punkt-Arten

'.' Punkt	's' Quadrat
',' Pixel	'p' Fünfeck
'o' Kreis	'*' Stern
'+' Plus-Zeichen	'h' Sechseck 1
'x' Kreuz-Zeichen	'H' Sechseck 2
'd' Diamant-Zeichen, dünn	'_' horizontale Linie
'D' Diamant-Zeichen	'|' vertikale Linie
'v' Dreieck mit Spitze nach unten	'1' Dreiecksstern mit Spitze nach unten
'^' Dreieck mit Spitze nach oben	'2' Dreiecksstern mit Spitze nach oben
'<' Dreieck mit Spitze nach links	'3' Dreiecksstern mit Spitze nach links
'>' Dreieck mit Spitze nach rechts	'4' Dreiecksstern mit Spitze nach rechts

Linienfarben formatieren

```
…
plt.plot(range1,11), werte1,'r')
…
```

Linienfarben:

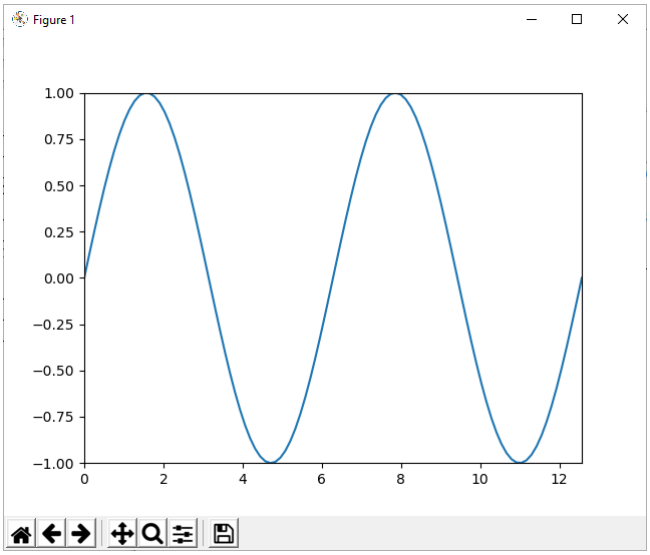
'k' schwarz	'g' grün	'y' gelb
'b' blau	'm' magenta	'w' weiß
'c' cyan	'r' rot	

feine Funktions-Diagramme

In der Praxis kommen Daten oft in NumPy-Array's daher. Diese sollen dann mittels MatPlotLib als Funktions-Diagramme dargestellt werden.
Im folgenden Programm wird zuerst einmal eine Sinus-Funktion über 2π-Intervalle erzeugt und angezeigt.

```
import numpy as np
import matplotlib.pyplot as plt
import math as mth

x_min = 0
x_max = 4 * np.pi
x_werte = np.linspace(x_min, x_max, 40, endpoint=True)
fkt = np.sin(x_werte)
y_min = mth.floor(np.min(fkt))
y_max = mth.ceil(np.max(fkt))

plt.plot(x_werte, fkt)
plt.axis([x_min, x_max, y_min, y_max])
plt.show()
```

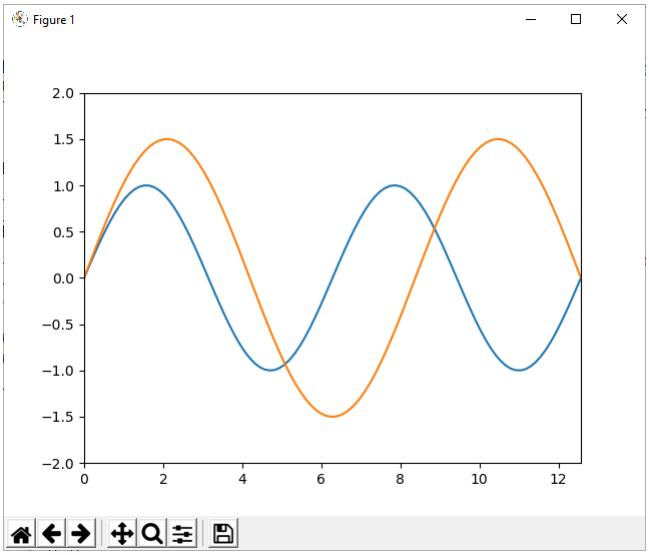
Auf der x-Achse haben wir hier – etwas unschön – dezimale Werte. Praktischer wäre hier eine Skalierung mit Pi-Vielfachen. Soetwas bekommt man durch Formatierung der Achsen eingestellt (→ Achsen gesondert definieren und formatieren).
Bei solchen Diagrammen kommt es oft auch dazu, dass die Kurven an die Achsen anstoßen. Wenn dies stört, dann kann man durch geschickte Wahl der Grenzen die Darstellung entsprechend anpassen (→ Diagramme ohne "anstößige" Kurven)



Bei Funktions-Diagrammen lassen sich natürlich auch mehrere Graphen darstellen:

```
fkt = np.sin(x_werte)
fkt2 = 1.5 * np.sin(0.75 * x_werte)
y_min = mth.floor(np.min(fkt))
y_max = mth.ceil(np.max(fkt))
y_min2 = mth.floor(np.min(fkt2))
if y_min2<y_min:
    y_min=y_min2
y_max2 = mth.ceil(np.max(fkt2))
if y_max2>y_max:
    y_max=y_max2


plt.plot(x_werte, fkt)
plt.plot(x_werte, fkt2)
plt.axis([x_min, x_max, y_min, y_max])
```

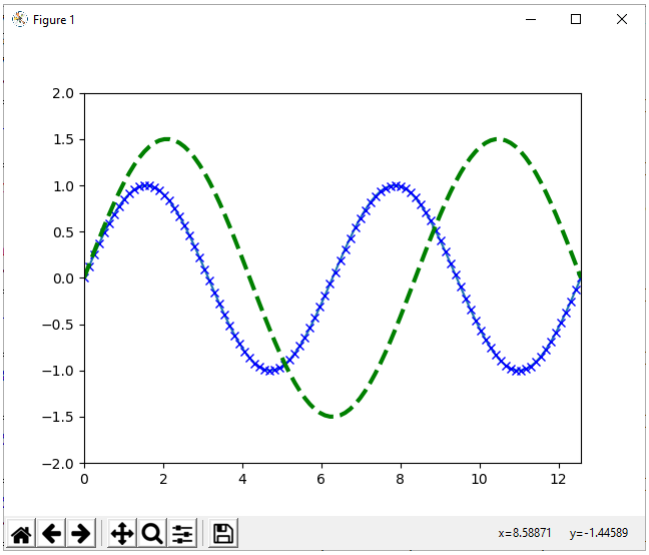
Im Programm wurden die Grenzen über die Maximum- und Minimum-Funktion ermittelt und durch Runden etwas erweitert. Da macht i.A. einen angenehmeren Eindruck. Allerdings muss man hier ev. noch Anpassungen vornehmen, wenn die Grenzen nicht im Einer-Bereich liegen. Ceil() und floor() runden eben nur auf den nächsten ganzzahligen Wert. Bei z.B. einem Maximum von 10'342,3 macht das keinen Sinn (Grenze wäre dann 10'343).

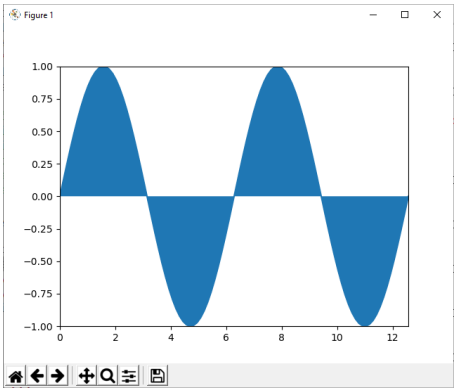


Die einzelnen Graphen lassen sich – äquivalent zu oben – gestalten:

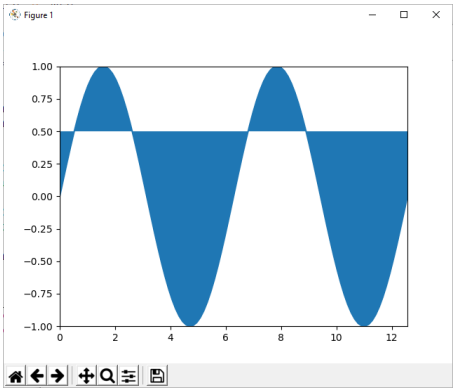
```
plt.plot(x_werte, fkt)
plt.plot(x_werte, fkt, 'bx')
plt.plot(x_werte, fkt2, color='green', linewidth=3, linestyle='--')
plt.axis([x_min, x_max, y_min, y_max])
```

Kehren wir zu unseren ursprünglichen Kurven zurück. In diesen wollen wir nun Flächen einfärben. Dazu stellt Mathplotlib die Funktion fill_between() bereit. Sie benötigt diverse Argument. Als Erstes wird das Array mit den x-Werten erwartet. Es folgen zwei Array's für y-Werte. Diese können aber auch auf 0 oder einen anderen Wert (quasi Parallele zur x-Achse) gesetzt werden.



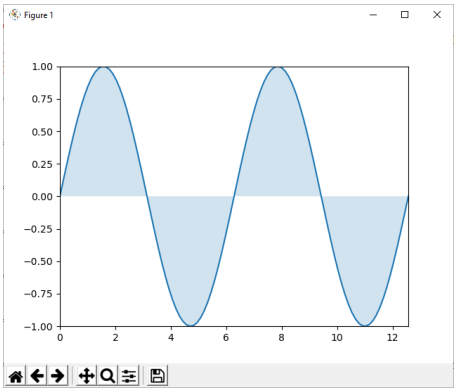



```
plt.fill_between(x_werte,0,fkt)
```




```
plt.fill_between(x_werte,0.5,fkt)
```

Über die Option alpha kann man die Farb-Intensität anpassen. Die normale Intensität ist auf 1,0 gesetzt. Durch eine kleinere Zahl läßt sich z.B. die Fläche aufhellen:

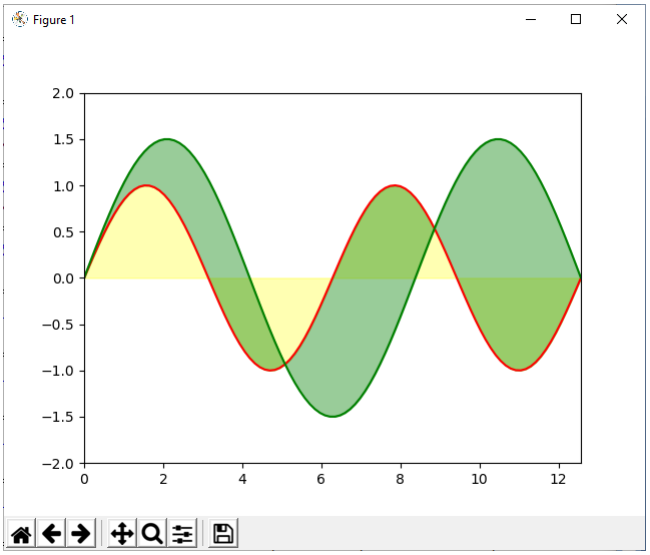



```
plt.fill_between(x_werte,0,fkt,alpha=0.2)
```

Das erzeugt für die normale Anwendung einen angenehmeren Eindruck.
Kommen wir aber zu den Argumenten von fill_between() zurück. Das zweite y-Argument kann auch ein Array sein. So lassen sich Flächen zwischen Graphen einfärben.
Hier nun noch einmal den gesamten Quelltext:

```
import numpy as np
import matplotlib.pyplot as plt
import math as mth

x_min = 0
x_max = 4 * np.pi
x_werte = np.linspace(x_min, x_max, 100, endpoint=True)
fkt = np.sin(x_werte)
fkt2 = 1.5 * np.sin(0.75 * x_werte)
y_min = mth.floor(np.min(fkt))
y_max = mth.ceil(np.max(fkt))

y_min2 = mth.floor(np.min(fkt2))
if y_min2<y_min:
    y_min=y_min2
y_max2 = mth.ceil(np.max(fkt2))
if y_max2>y_max:
    y_max=y_max2

print(y_min, y_max)

plt.plot(x_werte, fkt, 'r')
plt.fill_between(x_werte,0,fkt,color='yellow',alpha=0.3)
plt.plot(x_werte, fkt2, color='green')
plt.fill_between(x_werte,fkt,fkt2,color='green',alpha=0.4)
plt.axis([x_min, x_max, y_min, y_max])
plt.show()
```

Zu erwähnen sind noch zwei weitere möglich Argumente / Optionen. Da wäre zum Einen "where". Where ist per Default auf None gesetzt. Man kann aber auch eine Numpy-Array mit Boolean's übergeben, mit denen festgelegt wird, wo eingefärbt werden soll.
Weniger gebräuchlich ist die zweite Option "interpolate". Mit ihr wird per Boolean-Wert festgelegt, ob zwischen den beiden y-Kurven der Schnittpunkt interploliert werden soll.
So ein Boolean-Array lässt sich z.B. durch den Vergleich von zwei Numpy-Array's erzeugen.
Mit **fill_betweenh()** lassen sich die Flächen zwischen vertikalen Kurven einfärben.

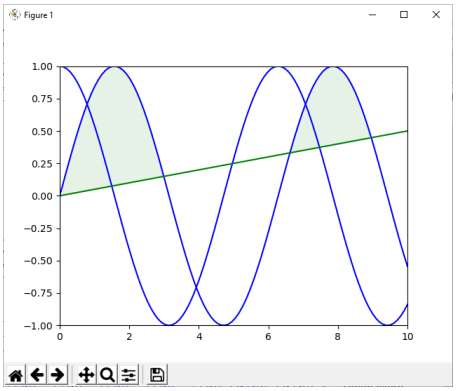


Aufgaben:

1. *Erstellen Sie ein Diagramm, in dem die Sin- und die Cos-Funktion im Intervall von 0 bis 10 angezeigt wird!*
2. *Lassen Sie sich die Fläche zwischen der x-Achse und der Cos-Funktion hell-bläulich einfärben!*
3. *Lassen Sie sich die Flächen zwischen der Sin- und der Cos-Funktion hell-rötlich einfärben!*

für die gehobene Anspruchsebene:

4. *Bei der Sin-Funktion soll nun die Fläche grünlich eingefärbt werden, die über einer Geraden (von {0,0} bis {10,0.5}) liegt! (s.a. Abb. rechts)*



Label / Achsen-Beschriftungen hinzufügen / formatieren

```
…
plt.xlabel('x-Achse')
plt.ylabel('y-Achse')
…
```

Achsen gesondert definieren und formatieren

```
…
achsen = plt.axes()
achsen.set_xlim([0,11])
achsen.set_ylim([-1,11])
achsen.xticks([1,2,3,4,5,6,7,8,9,10])
achsen.yticks([0,1,2,3,4,5,6,7,8,9,10])
plt.plot(range(1,11),werte)
plt.show()
```

Bereiche definieren und in einem Achsen-Aufruf unterbringen:

```
…
xmin=0
xmax=10
ymin=0
ymax=100
plt.axes([xmin, xmax, ymin, ymax])
…
```

Achsen bei einem bestimmten Wert schneiden lassen:

zuerst einmal die "alten" Achsen unsichtbar machen

```
achsen = plt.gca()
achsen.spines['right'].set_color('none')
achsen.spines['top'].set_color('none')
```

dann neue Achsen einstellen

```
achsen.xaxis.set_ticks_position('bottom')
achsen.spines['bottom'].set_position('data',0)
achsen.yaxis.set_ticks_position('left')
achsen.spines['left'].set_position('data',0)
```

Achsen im Programm-Ablauf anpassen (abfragen und neu festlegen)

```
…
print("Die aktuellen Minima und Maxima für die Achsen sind:")
print(plt.axes())   # Abfrage
print("die neue Grenzen werden nun festgelegt auf:")
xmin, xmax, ymin, ymax = 0, 12, 0, 90
print(xmin, xmax, ymin, ymax)
plt.axes([xmin, xmax, ymin, ymax])   # Setzen
print("Die aktuellen Minima und Maxima für die Achsen sind nun:")
print(plt.axes())   # Abfrage
```

An dieser Stelle ist z.B. auch ein automatisches Runterrunden der Minima und Aufrunden der Maxima sinnvoll einzusetzen (Runden: →). So lassen sich auch Diagramme erstellen, bei denen der Graph so skaliert wird, dass der Plot optimal ausgenutzt wird. Ev. nicht gebrauchte Zahlen-Bereiche werden so eleminiert.

Aufgaben:

1. *Lassen Sie sich ein Diagramm für die Quadrate der Zahlen von 5 bis 15 anzeigen! Die Achsen sollen dabei so skaliert werden, dass der Graph möglichst groß dargestellt wird!*
2.
3.

Diagramme ohne "anstößige" Kurven

```
…
plt.xlim(X.min()*1.1, X.max()*1.1)
plt.ylim(Y.min()*1.1, Y.max()*1.1)
…
```

Gitternetz-Linien hinzufügen

Zum besseren Orientieren und Ablesen von Werten dienen Gitternetzlinien.

```
…
achsen.grid()
…
```

für Polar-Diagramme:

```
plt.rgrids( … )
```

Aufgaben:

1.
2. *Erstellen Sie ein Diagramm für eine halbierte quadratische Funktion, in dem die Funktionswerte von 0 bis 5 eingezeichnet sind! Mit Hilfe eines Gitternetzes und der geeigneten Achsen-Skalierung sollen Schüler die Möglichkeit bekommen im Ausdruck (gespeichertes Bild), die Funktionswerte von 6 bis 10 zu ergänzen!*
3.

Daten-Punkte beschriften

Mit xy wird die Position bezüglich des Gitternetzes bestimmt und mit s der auszugebene Text

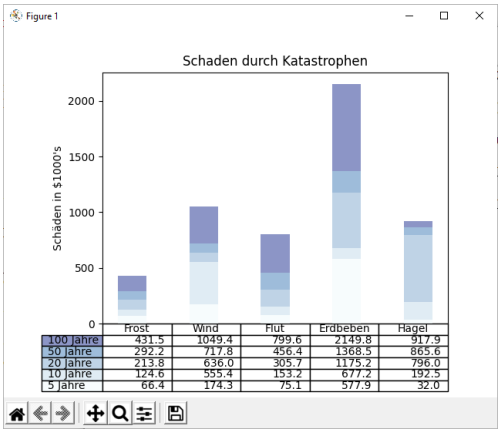
```
…
plt.annotate(xy=[1,1], s='Wert 1')
…
```

Legende hinzufügen / formatieren

```
…
plt.legend(['Datenreihe 1','Datenreihe 2'], loc = 4
…
```

einer Achse eine Daten-Tabelle hinzufügen / formatieren

```
plt.table( … )
```



(Bsp. aus MatPlotLib-Dokumentation; übersetzt: dre)

mehrere Diagramme in einem Plot

```
plot, ((diag1, diag2), (diag3, diag4)) = plt.subplots(2,2)
```

Nachfolgend müssen dann die Unter-Diagramme diag1 bis 4 wie gewöhnliche plot's initialisiert und formatiert werden.

alle offenen Plot's schließen und die Optionen wieder zurücksetzen

```
plt.switch_backend()
```

sollen nur die Achsen wieder zurückgesetzt werden, dann geht dies mit:

```
plt. cla()
```

und die gesamte Figur / den gesamten Plot zurücksetzen durch:

```
plt.clf()
```

8.6.5.2.4. weitere Diagramm-Typen

Kreis-Diagramm

```
werte = [20,30,20,15,15]
farben=['b','c','g','m','w']
beschrift=['blaue','hellblaue','grüne','lilane','weiße']
explodiert=[0,0.2,0,0,0]
plt.pie(werte, color=farben, labels=beschrift, explode=explodiert,
autopct=%1.1f%%, counterclock=False, shadow=True)
plt.title('Werte')
plt.show()
```

Balken-Diagramm

gemeint ist hier ein Säulen-Diagramm

```
…
weite = [0.7,0.7,0.7,0.7,0.7]
plt.bar(range(0,5), werte, width = weite, color = farben, align='center')
…
```

ein horizontales Balken-Diagramm (also ein echtes Balken-Diagramm erhält man mit:

```
plt.barh( …)
```

Aufgaben:

1. *Recherchieren Sie die letzten Wahlergebnisse zum Bundestag*
   a) für Deutschland (insgesamt)
   b) aus Ihrem Bundesland
2. *Stellen Sie die Deutschland-Daten in einem Torten-Diagramm dar! Stellen Sie eine Partei in explodierter Form heraus (z.B. Wahl-Sieger oder Ihre Präferenz)!*
3. *Die Daten aus Ihrem Bundesland sollen in einem Säulen-Diagramm angezeigt werden!*

für die gehobene Anspruchs-Ebene:

4. *Recherchieren Sie die vorletzten Wahlergebnisse zum Bundestag für Deutschland dazu und erstellen Sie ein geeignetes Diagramm, in dem die Zugewinne und Verluste sichtbar werden.*

Histogramm

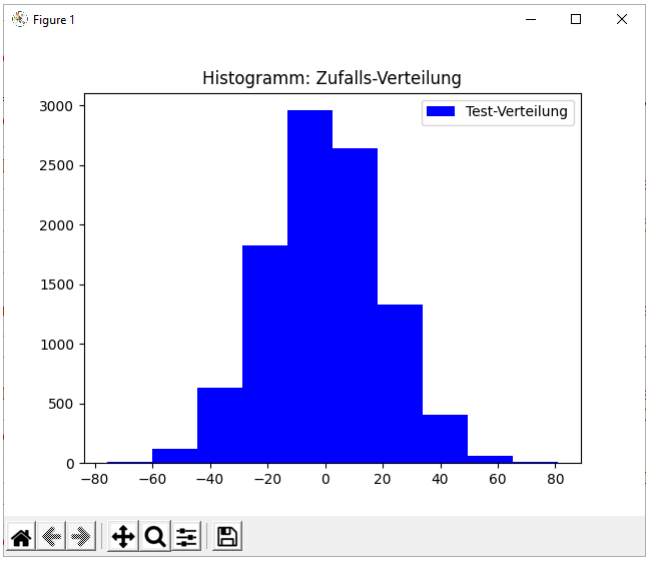
Entwickeln eines Histogramm's über eine Zufalls-bedingte Verteilung

```
import numpy as np
import matplotlib.pyplot as plt

werteX=20* np.random.randn(10000)
plt.hist(werteX, bins=10,
         #range(-50,50),
         histtype='bar', align='mid',
         color='b', label='Test-Verteilung')
plt.legend()
plt.title('Histogramm: Zufalls-Verteilung')
plt.show()
```

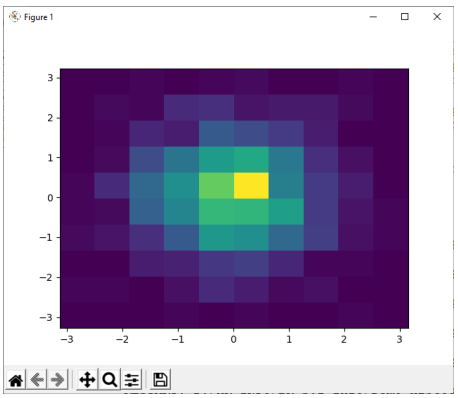
Auch möglich sind 2D-Histogramme. Die Betrachtungs-Richtung ist nun von oben und die Höhe der Säulen wird durch die Farben ausgedrückt. Die Funktion dafür heißt:

```
plt.hist2d( … )
```

Wir benötigen natürlich 2-dimensionale Daten. Diese erzeugen wir uns für ein Beispiel wieder per Zufalls-Funktion aus dem NumPy-Modul.




```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.randn(1000)
y = np.random.randn(1000)
plt.hist2d(x, y, bins=10)
plt.show()
```



Eine weitere interessante Darstellung ist mit z.B. **.hexbin(x,y, gridsize=25)** möglich. Allerdings wird hier das Interpretieren der Daten schon schwieriger, da ein exaktes Zuordnen der Daten unklarer bleibt.

Aufgaben:

1. *Erstellen Sie eine Histogramm für 10'000 Zufallszahlen im Bereich von 0 bis 1,0*
2. *Erzeugen Sie sich ein Histogramm mit 10 Gruppen (bins) aus 1'000 Zufalls-Zahlen aus dem Zahlen-Bereich von 0 bis 100 in der Farbe rot!*
3. *In einem Diagramm soll für 100'000 Würfe mit einem 6er Würfel die Verteilung der gewürfelten Punkte dargestellt werden. Erstellen Sie ein entsprechenes Programm dafür!*
4. *Erstellen Sie ein 2D-Histogramm für 100'000 Zufalls-Zahlen, die in einem Raster von 50x50 dargestellt werden sollen!*

Ein Linien- / Funktions-Diagramm mit Fehler-Indikator

```
plt.errorbar( … )
```

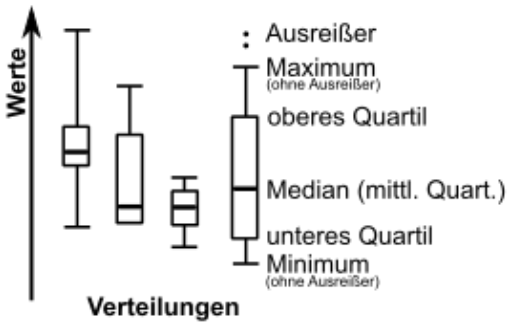
Streu-Diagramme / Haufen-Diagramme

```
plt.( … )
```

Box-Plot-Diagramme / Whisker-Diagramme

Box-Plot's eigenen sich besonders gut dazu, um umfangreiche Datensätze mittels mehrerer Kennwerte vergleichend darzustellen.
Die obere und untere Grenze (Maximum und Minimum) werden auch als Whisker (frei übersetzt: Haar-Linie) bezeichnet.

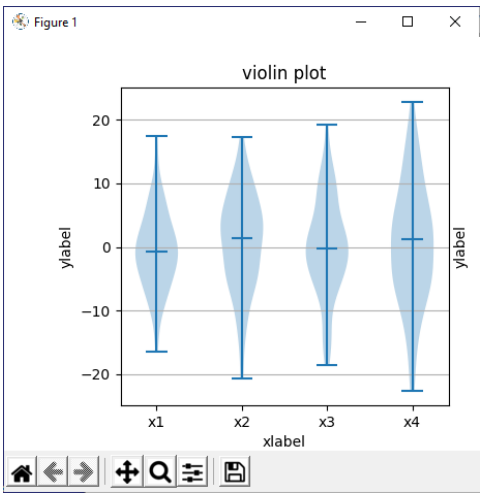
```
plt.boxplot( … )
```



???-Diagramme / Violin-Plot

vereint Histogramme in einer Box-Plot-ähnlichen Struktur
oder anders herum: In Boxplot's werden die Daten statt mit Boxen durch Verteilungen dargestellt
ermöglichen mehr Detail-Verständnis, dafür fallen die kategorisierenden und verallgemeinernden Kennwerte weg.

```
plt.violinplot( … )
```



Gruppen von Box-Plot's

Darstellung von Graphen (Knoten und Kanten)

hierfür muss aber eine andere (Unter-)Bibliothek (mit) eingebunden werden, die sich **matplotlib.path** nennt.

Kästchen-Diagramme

Folge von horizontal angeordneten Rechtecken (z.B. für differenzierte Balken)

```
plt.broken_barh( … )
```

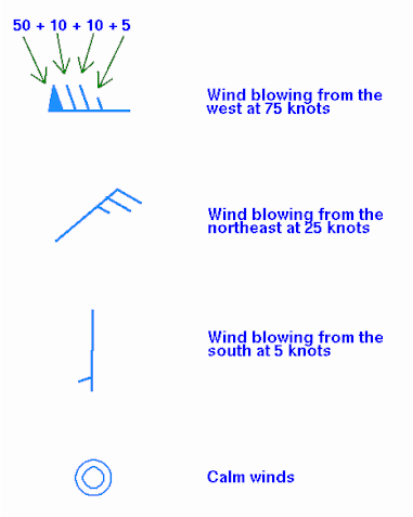
andere Varianten erreicht man durch:

```
plt.pcolormesh( … )
```

oder:

```
plt.contour( … )
```

Diagramme mit Wind-Fahnen (Barb's, Strömungs-Fahnen)



Q: en.wikipedia.org (Thegreatdr)

Für Wetterkarten oder metreologische Modelle sind u.a. Diagramme mit Wind-Fahnen gebräuchlich. Mit **barbs()** lassen sich solche Wind-Fahnen auf einer 2D-Fläche verteilen. Einige Grund-Informationen zur Codierung kann aus der nebenstehenden Abbildung entnommen werden.
Der Grund-Aufruf erfolgt über die Funktion:

```
plt.barbs( … )
```

Dabei sind Unmengen von Optionen und speziellen Darstellungen möglich. Wer hier einsteigen will oder muss, dem hilft die Hilfeseite zu dieser Methode / Funktion weiter.
Für die Zwecke dieses Skriptes geht das zu weit.

Sehr ähnlich sind Diagramme mit Pfeilen / Pfeilspitzen

```
plt.quiver( … )
```

Diagramme mit Strömungs-Linien (Stromlinien-Diagramme)

```
plt.streamplot()
```

Zeichnen eines Spektrogramm's

```
plt.specgram( … )
```

Zeichnen eine Phasen-Spektrum's

```
plt.phase_spectrum( … )
```

Magnituden-Spektren

```
plt.magnitude_spectrum( … )
```

Step-Diagramme (???)

```
plt.step( … )
```

interessante Links:
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf (Cheat Sheet zu MatPlotLib)
http://matplotlib.1069221.n5.nabble.com/Matplotlib-3-1-cheat-sheet-td49476.html (visuelles Cheat Sheet zu MatPlotLib)
https://matplotlib.org/gallery.html (Diagramm-Gallerie mit Code-Beispielen)

8.6.5.3. ein komplexes Diagramm-Projekt – Erdbeben-Anzeige

Q: https://github.com/rougier/matplotlib-tutorial

8.6.6. Modul / Bibliothek network

Mit der Bibliothek kann man sich die Arbeiten mit Graphen und Graph-Daten in Python vereinfachen.

Adjazenz-Matrix ist ein Mittel zur Darstellung der Kanten eines Graphen
Jeder Eintrag in der Matrix größer Null steht für eine Verbindung zwischen den Knoten, die Höhe des Eintrags steht für die Länge / den Wert / die Kosten der Verbindung
Bei gerichteten Graphen muss die Matrix nicht spiegelbildlich / ??? sein

```
import network as netw
graf = netw.cycle_graph(10)  // Beispiel: Zyklus mit 10 Knoten
adjMatrix = netw.adjacency_matrix(graf)

print (adjMatrix.todense())
```

- - - - - - - - erweiterte Fortsetzung

```
import matplotlib.pyplot as zeichnung

netw.draw_networkx(graf)
zeichnung.show()
```

- - - - - - - - erweiterte Fortsetzung

```
graf.add_edge(knoten1, knoten2)
netw.draw_networkx(graf)
zeichnung.show()
```

8.6.7. Modul / Bibliothek re

Bei der Bibliothek handelt es sich um ein Modul zur Nutzung von rregulären Ausdrücken.

Suchmuster lassen sich mit re wesentlich effektiver nutzen, da diese in re sehr Maschinen-nah programmiert wurden. Damit ist re deutlich effektiver als die meisten selbst-programmierten Muster-Suchen.

```
import re
 muster=re.compiler(r'(regulärer_Ausdruck))
eingabe=beispieltext
ausgabe=muster.search(eingabe).groups()
print(ausgabe)
```

(re) gruppiert reg.Ausd.e und behält den übereinstimmenden Text
(?: re) gruppiert reg.Ausd.e ohne den übereinstimmenden Text
(?# …) ist ein Kommentar; wird nicht vom Compiler verarbeitet
re? Maximal eine Übereinstimmung im vorherigen Ausdruck
re* keine oder mehrmalige Übereinstimmung im vorherigen Ausdruck (→ KLINE-Stern)
re+ mindestens eine Übereinstimmung im vorherigen Ausdruck
(?> re) Übereinstimmung in einem unabhängigen Muster ohne Rückverfolgung
[^ …] Übereinstimmung in jedem möglichen einzelnen Zeichen oder einer Reihe / Gruppe von Zeichen, die nicht innerhalb der Klammern vorkommen
[…] Übereinstimmung in jedem möglichen einzelnen Zeichen oder einem Bereich / einer Gruppe von Zeichen, die innerhalb der Klammern vorkommen
re(n, m) Übereinstimmung mit mindestens n und maximal m Vorkommen des vorherigen Ausdrucks
\n, \r, \t, … Übereinstimmung mit Steuerzeichen (neue Zeile, Zeilenumbruch, Tabulator, …)
\d Übereinstimmung mit Ziffern (äquivalent mit: [0-9])
\D Übereinstimmung mit einem Nicht-Ziffern-Symbol
\S Übereinstimmung mit einem Nicht-Leerzeichen-Symbol (alles außer Leerzeichen)
\s Übereinstimmung mit einem Leerzeichen (äquivalent zu: [\t\n\r\f])
\b Übereinstimmung mit Wortgrenzen außerhalb der Klammern (Übereinstimmung mit Backspace (0x08) innerhalb der Klammern)
\B Übereinstimmung mit leeren Zeichenketten, die vor oder hinter einem Wort stehen (→ Trimmen des Ausdruck's)
\w Übereinstimmung mit Wortzeichen
\W Übereinstimmung mit einem Nicht-Buchstaben-Symbol (alles außer Buchstaben)
\A Übereinstimmung mit dem Beginn einer Zeichenkette
^ Übereinstimmung mit dem Beginn einer Zeile
$ Übereinstimmung mit dem Zeilenende
\z Übereinstimmung mit dem Ende einer Zeichenkette
\Z Übereinstimmung mit dem Ende einer Zeichenkette, wenn eine folgende Zeile existiert (Übereinstimmung vor dem Zeilenumbruch)
\1 … \9 Übereinstimmung mit dem n-gruppierten Unterausdruck
\G Übereinstimmung mit den Zeichen der zuletzt gefundenen Übereinstimmung
a | b Übereinstimmung mit a oder b
re{ n} Übereinstimmung mit exakt der Anzahl n des Vorkommens des vorherigen Ausdrucks
re{ n, } Übereinstimmung mit mindestens der Anzahl n des Vorkommens des vorherigen Ausdrucks

(?= re) gibt eine Position unter Verwendung eines Musters zurück (das Muster hat keinen bestimmten Bereich)
(?! re) gibt eine Position unter Verwendung der Negation eines Musters zurück (das Muster hat keinen bestimmten Bereich)
(?-imx) schaltet die (Compiler-)Optionen i-, m- und x- zeitweise aus (wenn Ausdruck in Klammern, dann gilt die Optionen-Abschaltung nur für den Klammer-Ausdruck (→ (?-imx: re)))
(?imx) schaltet die (Compiler-)Optionen i-, m- und x- zeitweise ein (wenn Ausdruck in Klammern, dann gilt die Optionen-Einschaltung nur für den Klammer-Ausdruck (→ (?imx: re)))

8.6.8. Modul / Bibliothek pymongo

Arbeiten mit einer NoSQL-Datenbank

```
import pymongo
import pandas as pds
from pymongo import Connection
verbindung = Connection()
datenbank = verbindung.database_name
eingabeDaten = datenbank.collection_name
daten = pds.DataFrame(list(input_data.find()))
```

8.6.9. Modul / Bibliothek ?? (Word Embedding)

Tokenisierung
Ist die Segmentierung eines Satzes auf Einheiten der Wort-Ebene.

Stemming
Ist der Prozess der Reduzierung eines Wortes auf seinen Wortstamm.

Entfernen von Suffixen

Entfernen von Präfixen

Stop-Wörter
Sind solche Wörter, die für das Satz-Verständnis durch uns Menschen eine wichtige Rolle spielen, für die Text-Analyse durch Computer aber eine untergeordnete Rolle haben.
Beispiele für Stop-Wörter: ein, und, die, diese, …

Bag-of-Words-Modell
als Ergebnis einer Tokenisierung erhält man eine Wort-Menge, die als eine Struktur (Menge) verfügbar ist
Sammeln der vorhandenen Wörter
Grammatik und Wortreihenfolgen werden ignoriert
Bag-of-Words (Wort-Behälter / -Tasche, Wort-Container) lässt sich dann für Klassifizierungen und / oder andere Analysen (weiter-)verwenden
Im Vorfeld sollte eine Entfernung von Sonder- und / oder Steuer-Zeichen , ein Stemming und das Entfernen von Stop-Wörtern erfolgen

N-Gramme
Ist eine kontinuierliche Folge von (allen) Elementen aus einem Text
N-Gramm kann ein Symbol, Silbe, Symbolfolge, ein Wort oder z.B. eine Basen-Sequenz (der DNA) sein
N-Gramm der Länge Eins heißt Unigramm, mit der Länge 2 , mit Länge 3 Trigramm usw. usf.
z.B. für die Vorhersage von folgenden Sequenzen wichtig
weiterhin Vergleiche von Texten und / oder Autoren möglich
(interessant auch die Beziehung zwischen den häufigsten Wortlängen und der Popularität von Texten (z.B. Welt-Literatur) sowie der Zielgruppe (Leserschaft))

TF-IDF-Transformation
Kommt von Term Frequency times Inverse Document Frequency ()
Verfahren, bei dem die Textlänge kompensiert wird (also kürzere und längere Texte vergleichbar gemacht werden soll)
Verfahren macht deutlich, wie wichtig ein Wort für den Text ist (Häufigkeit wird zur Länge des Textes in Bezug gesetzt)
TF ermittelt die Häufigkeit des Wortes
IDF bestimmt die Wichtigkeit des Wortes im / für den Text

8.6.99. Cheat Sheet's für einige Bibliotheken

Zusammenstellungen / Übersichten / Hilfs-Blätter heißen Neudeutsch Cheat Sheet's.

von DataComp.com sind einige sehr gut zusammengestellte im Internet verfügbar
wenn ich noch andere zu den erwähnten Bibliotheken / Themen gefunden habe, dann wurden sie von mir gleich dahinter angegeben

welche Cheat Sheet's zu einem selbst passen, muss man einfach ausprobieren
so gewaltig unterscheiden sich die einzelnen Übersichten aber nicht

zu NumPy
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

zu Matplotlib
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf

http://matplotlib.1069221.n5.nabble.com/Matplotlib-3-1-cheat-sheet-td49476.html

zu SciPy (lineare Algebra)
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_SciPy_Cheat_Sheet_Linear_Algebra.pdf

Scikit-Learn (Machine learning)
https://datacamp-community-prod.s3.amazonaws.com/5433fa18-9f43-44cc-b228-74672efcd116

zu Pandas
http://datacamp-community-prod.s3.amazonaws.com/dbed353d-2757-4617-8206-8767ab379ab3
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Pandas_Cheat_Sheet_2.pdf

weitere Cheat Sheet's

Importing Data
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Cheat+Sheets/Importing_Data_Python_Cheat_Sheet.pdf

Tidyverse (transforming and visualizing data)
https://datacamp-community-prod.s3.amazonaws.com/e63a8f6b-2aa3-4006-89e0-badc294b179c

Jupyter Notebook
https://datacamp-community-prod.s3.amazonaws.com/48093c40-5303-45f4-bbf9-0c96c0133c40

keras (Neural networks)
https://datacamp-community-prod.s3.amazonaws.com/94fc681d-5422-40cb-a129-2218e9522f17

Seaborn (Statistic Data Visualization)
https://datacamp-community-prod.s3.amazonaws.com/48093c40-5303-45f4-bbf9-0c96c0133c40

PySpark (Spark DataFrames / SparkSQL)
https://datacamp-community-prod.s3.amazonaws.com/65076e3c-9df1-40d5-a0c2-36294d9a3ca9

Bokeh (Daten-Präsentation in Web-Browsern)
https://datacamp-community-prod.s3.amazonaws.com/f9511cf4-abb9-4f52-9663-ea93b29ee4b7

spaCy (advanced NLP)
http://datacamp-community-prod.s3.amazonaws.com/29aa28bf-570a-4965-8f54-d6a541ae4e06

R ()
data.table R ()
https://datacamp-community-prod.s3.amazonaws.com/6fdf799f-76ba-45b1-b8d8-39c4d4211c31

xts (time series in R)
https://datacamp-community-prod.s3.amazonaws.com/e04c5a6b-4aca-46f5-8cd5-803d975ccc4b

Data Science
https://datacamp-community-prod.s3.amazonaws.com/e30fbcd9-f595-4a9f-803d-05ca5bf84612

8.7. Graphik

Auf der Kommando-Ebene und in der Grund-Version bietet Python keinen Zugriff auf die graphischen Fähigkeiten von Windows oder vergleichbaren Betriebssystemen und / oder ihren Benutzer-Oberflächen.
Für einfache Zwecke kann man sich mit Pseudo-Graphiken behelfen. Dabei werden die verschiedensten ASCII-Symbole benutzt. Vor allem im erweiterten ASCII-Code ab Symbol 128 sind diverse Zeichen für Pseudographiken enthalten. Damit lassen sogar grobe Diagramme zeichnen.

Um die graphischen Möglichkeiten moderner betriebssysteme zu nutzen, bedarf es aber der Einbindung von geeigneten Modulen.
An vorderster Front ist hier die Turtle-Graphik mit dem Modul turtle (→ 8.8. Turtle-Graphik – ein Bild sagt mehr als tausend Worte)zu nennen. Sie geht auf

zurück.
Heute ist die Turtle-Graphik vollständig Pixel-orientiert. Die Schildkröte als Zeichen-Stift soll nur einer besseren Orientierung und der Verständlichkeit / Nachvollziehbarkeit dienen.

Andere graphische Systeme ermöglichen den Zugriff oder die Nutzung von typischen Bedien-Elementen aus den Betriebssystemen. Das sind zum Einen die Einzel-Bedienelemente, wie Text-Felder (Edit-Feld), Beschriftungen (Label), Optionen (), Auswahl-Listen (List- oder Combo-Boxen usw. usf. Zum anderen werden fertige Dialoge bereitgestellt. Hierzu zählen kleine Meldungs-Fenster (Message-Dialog) aber auch Datei-Speicher- oder -Öffnen-Dialoge.
Dies ermöglicht es dem Programmierer, sich auf den Kern seines Programm's zu konzentrieren und die sich wiederholenden, klassischen Aufgaben der Kompetenz von System-Profi's zu überlassen.
Die Module stellen dabei vorrangig Schnittstellen und Übersetzungen zwischen Python und den Graphik-Systemen der Betriebssysteme zur Verfügung.
Zu den bekanntesten Modulen, die Bedien-Elemente, Fenster und Dialoge bereitstellen, gehört Tkinter (→ 8.12. GUI-Programme mit Tkinter).

8.8. Turtle-Graphik – ein Bild sagt mehr als tausend Worte

didaktische Entwicklungs-Oberfläche (Interface-Builder)
schöne Bilder und coole Abläufe, Programmierung soll da so nebenbei mit aufgenommen werden
aber genau das ist die Gefahr aus meiner Sicht, das Nebenbei wird häufig zu wenig aufgenommen und von einem Jahr Programmierung bleiben nur wenige Monate Ahnungs-Effekte - an die tolle Oberfläche kann sich aber jeder noch erinnern
besonders gut für einen sehr frühen Kontakt mit Programmierung

wir werden hier die schon getätigten Schritte dieses Skriptes zur Einführung in die Programmierung quasi auf graphischem Niveau wiederholen

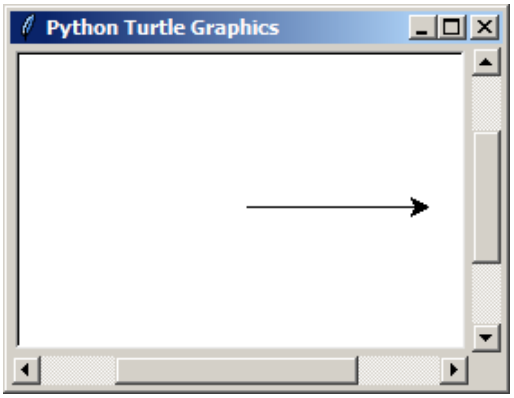
8.8.1. Turtle auf der Shell

Die Turtle-Graphik ist ein internes Modul, was mit der Installation von Python schon mit eingerichtet wurde. Um die Turtle-Graphik nutzen zu können, müssen wir unsere aktuelle Python-Version etwas pushen.

```
>>> import turtle
>>> turtle.forward(0)
>>> turtle.forward(100)
>>>
```

Durch die Eingabe von **`import turtle`** werden die Befehle und Funktionen der Turtle-Graphik geladen und verfügbar gemacht.
Eine alternative Bibliothek ist **gturtle**.

In den meisten Fällen passiert gar nichts, weder gibt es eine Meldung in der Shell, noch sehen wir eine Schildkröte.
Gibt man als nächsten Befehl z.B. **`turtle.forward(0)`** ein, dann erscheint ein Graphik-Fenster, in dessen Mitte eine Pfeilspitze zu sehen ist. Die Pfeilspitze ist unsere Schildkröte. Sie zu bewegen und sie dazu anzuregen Spuren zu hinterlassen, dass ist der Hintergedanke bei der Turtle-Graphik.

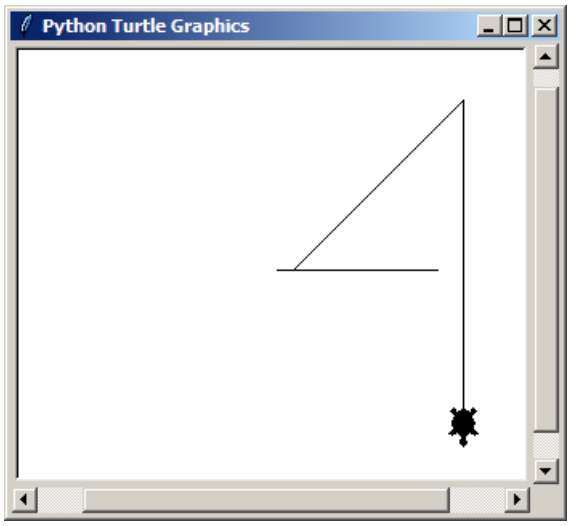


Mit **`turtle.forward(100)`** bewegen wir die Schildkröte 100 Pixel (auf dem Bildschirm) vorwärts. Üblicherweise werden die Pixel als Schritte interpretiert.
Der zurückgelegte Weg wird als Linie (gesetzte Pixel) sichtbar. Rückwärtes geht's mit **`turtle.backward(`***`schritte`***`)`**. Die zurückgelegten Wege sind dann vollständig oder teilweise deckungsgleich. Es ist aber nur eine Linie zu sehen.

```
>>> turtle.backward(90)
>>> turtle.left(45)
>>> turtle.forward(150)
>>> turtle.right(135)
>>> turtle.forward(200)
>>> turtle.shape("turtle")
```

Richtungs-Änderungen lassen sich mit **`turtle.left()`** und **`turtle.right()`** erreichen. Als Argument müssen wir den (Dreh-)Winkel (in Grad) übergeben. Wem das Dreieck als Turtle zu abstrakt ist und lieber eine echte Schildkröte wandern sehen möchte, der kann ja mal **`turtle.shape("turtle")`** ausprobieren.

Den ursprüngliche Dreiecks-Zeiger erhält man über den Text `"classic"`. Weitere Formen sind `"arrow"`, `"circle"`, `"square"` und `"triangle"`.



Ein Neustart der Turtle-Graphik – quasi das Löschen des Ausgabe-Bildschirm ("Vergessen der alten Wege") erreicht man mit `turtle.reset()`.
Beim Beenden des Shell-Fensters wird auch die Turtle-Graphik geschlossen. Will man dieses Fenster weiterhin sehen, gibt man in der Shell den Befehl `turtle.exitonclick()` ein.
Erwähnt sei hier auch noch der Befehl `turtle.undo()`, der den jeweils letzten Befehl rückgängig macht.
Hat man etwas Geduld bei der Eingabe der turtle-Befehle, das erscheint nach dem Eintippen des Punktes eine Code-Ergänzungs-Auswahlbox.
Mit der Tab-Taste wird die ausgewählte Funktion übernommen.
Vorher kann man mit der Maus oder mitttels der Eingabe weiterer Buchstaben die geeignete Funktion heraussuchen.

Aufgaben:

1. *Aktivieren Sie die Turtle-Graphik und lassen Sie sich das Graphik-Fenster mit der Schildkröte in Lauerstellung anzeigen!*
2. *Bewegen Sie die Schildkröte so, dass ein Summen-Zeichen (∑) auf dem Bildschirm zu sehen ist!*

Kommen wir nun schon im Vorfeld der echten Programmierung zu den Befehlen, die besonders von Mädchen / weiblichen Programmierern als erstes erfragt werden. Das ist ein Befehl, der die Farbe der Schildkröte und damit auch ihre Spur-Farbe ändern kann.
In einer ersten Möglichkeit legt man die Farbe mittels eines Color-Strings fest.
Das sind vordefinierte Farben mit den üblichen englisch-sprachigen Bezeichnungen.
Die zweite Variante der Farb-Festlegung ist weitaus Leistungs-fähiger, aber auch aufwändiger und komplizierter in der Umsetzung.

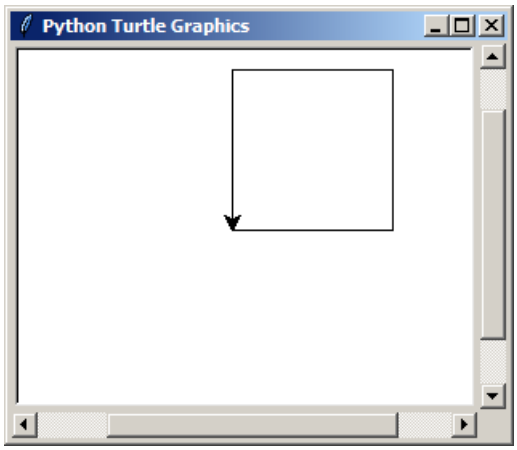
Bei dieser Variante wird die Farbe als RGB-Tupel übergeben. Es können nun die Zahlen für die Rot-, Grün- und Blau-Anteile von 0 bis 255 verändert werden.

Verwendet man die Funktion turtle.color() ohne Argumente, dann wird die aktuelle Farbeinstellung als Color-String oder Hexadezimal-Code zurückgegeben.
Der Hintergrund des Graphik-Fenster lässt sich mit der Funktion turtle.bgcolor(farbcode) einstellen.
Ein geschlossener Polygonzug kann auch in der Fläche eingefärbt werden.

8.8.2. Turtle-Programme und Sequenzen

Das Erstellen von Programmen unterscheidet sich kaum von der Eingabe auf der Konsole (Shell) bzw. von der bei anderen Programmen. Das forward(0) zum Aktivieren / Anzeigen des Graphik-Fenster kann ausbleiben. Python zeigt das Fenster mit den ersten Programmschritten sofort an.
Wichtiger Hinweis auf einen Fehler-Klassiker bei der Turtle-Programmierung. Schnell vergibt man den Dateinamen für den selbstgeschriebenen Quelltext mit turtle.py. Danach funktioniert die Turtle-Programmierung nicht mehr, weil das Turtle-Modul quasi durch das eigene Programm ersetzt wurde. Man importiert das eigene Programm als Modul, was keinen Sinn macht und auch noch rekursiv ins Nirvana führt.
Unsere erste Aufgabe soll das Zeichnen eine Quadrates sein. Dazu brauchen wir die vier gleichen Seiten einer bestimmten Länge und am Ende der Seite immer eine Drehung um 90°.

```
# Zeichen eines Quadrates
import turtle
laenge=100
turtle.forward(laenge)
turtle.left(90)
turtle.forward(laenge)
turtle.left(90)
turtle.forward(laenge)
turtle.left(90)
turtle.forward(laenge)
```



Das Ergebnis überrascht wenig.
Aber das einfache Aneinanderreihen von Turtle-Anweisungen nervt jetzt schon ein bisschen. Viel tippen und wenig Leistung des Programms. Wir wissen ja schon, dass man mit Schleifen effektiver arbeiten kann.
Eigentlich würden jetzt zuerst die Verzweigung folgen, aber Graphik-Programmierung lebt mehr von Wiederholungen als von Alternativen. Die sind aber gleich danach dran (→ 8.8.4. Verzweigungen).

Aufgaben:

1. ***Erstellen Sie ein Programm, in dem die Schildkröte ein Rechteck mit den Kantenlängen 250 und 320 zeichnet!***
2. ***Erstellen Sie ein Programm, das ein Quadrat mit einer Kantenlänge von 200 zeichnet!***
3. ***Lassen Sie die Schildkröte das Quadrat so zeichnen, dass es um 45° gedreht ist!***
4. ***Erstellen Sie ein Programm mit der (ungeprüften) Eingabe eines Winkels (am Zeichen-Ursprung) für ein rechtwinkliges Dreieck! Die erste Kantenlänge soll 200 betragen! Lassen Sie alle anderen Winkel und Kanten berechnen! (Die Quadrat-Wurzel-Funktion sqrt() ist in der math-Bibliothek verfügbar.)***

für die gehobene Anspruchsebene:

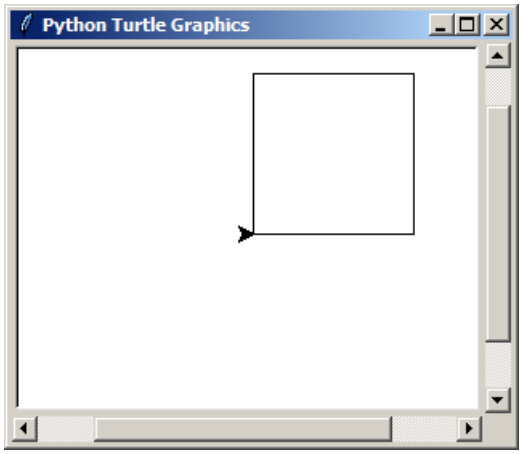
5. ***Gesucht ist das Programm, mit dem ein Rhombus mit einer Seitenlänge von 180 und den Winkeln 60 und 120 Grad gezeichnet wird! In der Erweiterung soll der gleicher Rhombus um 45° gedreht zusätzlich dargestellt werden!***

(c,p) 2015 - 2022 lsp: dre

8.8.3. Schleifen

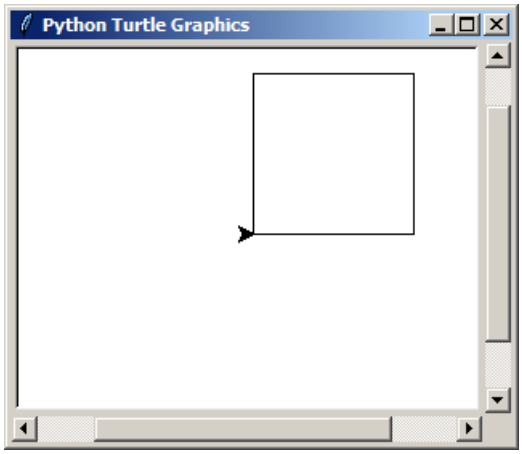
Also programmieren wir unser Quadrat über eine Schleife. Der Einfachheit halber nehmen wir auch noch eine abschließende Drehung in das Programm auf. Die Schildkröte wird somit auf die Ausgangs-Position und –Richtung zurückgesetzt.

```
# Zeichen eines Quadrates
import turtle
laenge=100
for i in range(4):
    turtle.forward(laenge)
    turtle.left(90)
```



Natürlich lässt sich das Gleiche auch mit einer while-Schleife erreichen. Was man praktisch wählt ist auch ein bisschen Geschmackssache. Im Falle von klaren Zähl-Vorgängen ist die Nutzung von for-Schleifen aber logisch verständlicher.
Mich persönlich schreckt auch immer der zusätzliche Aufwand (Vorbelegung der Laufvariable, Korrektur der Laufvariable) ab. Alles Fehler-Quellen, die sich durch eine "schöne" Zählschleife in Grenzen halten lassen.

```
# Zeichen eines Quadrates
import turtle
laenge=100
anzahl=0
while anzahl<4:
    turtle.forward(laenge)
    turtle.left(90)
    anzahl+=1
```



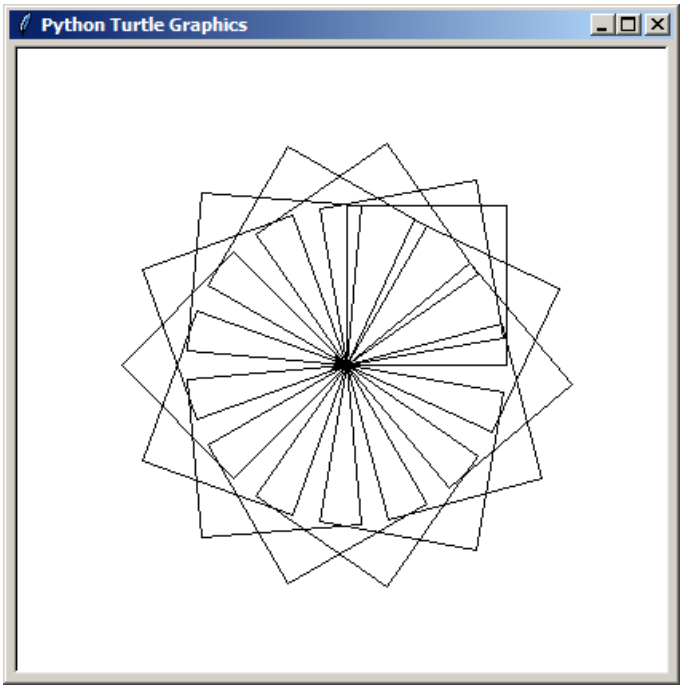
Beim Erzeugen von Mustern oder Wiederholungen, deren Anzahlen sich schwer abschätzen lassen, sind while-Schleifen dann natürlich die bessere Wahl.
Zur Demonstration nehmen wir hier mal das Zeichnen eines Musters aus Quadraten, die immer leicht verdreht zueinander solange gezeichnet werden sollen, bis die Umkreisung vollständig ist.

```
# Zeichen eines Musters aus Quadraten
import turtle
laenge=100
drehwinkel=25
gesamtwinkel=0
while gesamtwinkel<360:
    for i in range(4):
        turtle.forward(laenge)
        turtle.left(90)
    turtle.right(drehwinkel)
    gesamtwinkel+=drehwinkel
```

Zeichnen begrenzen; Schluß nach mind. 360°
einegentliches Zeichnen des Quadrates

Drehung auf neue Anfangsricht
(Gesamt-)Drehwinkel verfolgen / korrigieren

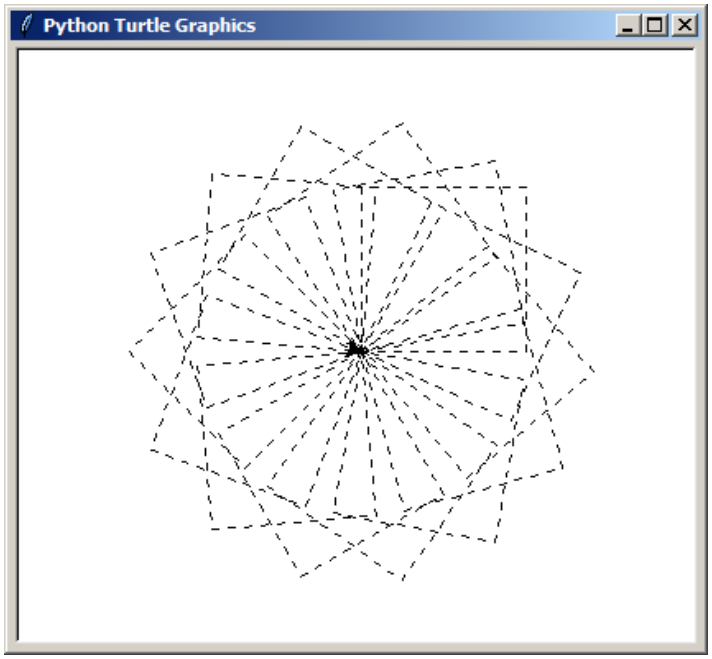
Schleifen-Konstrukte können wir auch benutzen, um z.B. gestrichelte Linien zu erzeugen.
Dazu müssen wir natürlich wissen, wie man die Schildkröte ohne Spur bewegt. Mit der Anweisung **`turtle.penup()`** wird der Zeichenstift (für die Spur) abgehoben und mit **`turtle.pendown()`** wieder aufgesetzt.
Will man nun ein Muster mit getsrichelten Linien zeichnen, dann kommt noch eine dritte Schleife hinzu, die quasi die alte Linienführung (turtle.forward(laenge) durch eine zusätzliche Muster-Erzeugung ersetzt.
Hier würde ich intuitiv lieber eine while-Schleife nehmen. Vor allem, weil die Länge ja als intern Veränderlich im Programm steht. Da will man auf alle Eventualitäten vorbereitet sein.



Um Ihnen nicht den Spaß am Programmieren und Experimentieren zu nehmen stelle ich hier nur die Erzeugung einer einzelnen gestrichelten Linie vor. Das Zusammenstellen und Einarbeiten in eigene Programme überlasse ich Ihnen.

```
# Strichel- und Nichtstrichel-Länge müssen zusammen
# einen Teiler von Länge bilden!
strichellaenge=5
nichtstrichellaenge=5
…
gesamtlaenge=0
while gesamtlaenge<laenge:
     turtle.forward(strichellaenge)
     turtle.penup()
     turtle.forward(nichtstrichellaenge)
     turtle.pendown()
     gesamtlaenge+=(strichellaenge+nichtstrichellaenge)
```

Ein Muster aus Quadraten mit Strichelmustern könnte dann so aussehen.



Aufgaben:

1. ***Erstellen Sie ein Programm, dass ein gleichseitiges Dreieck mit der Kantenlänge 75 erzeugt!***
2. ***Erweitern Sie das Dreiecks-Programm nun um die Drehung um 5° solange bis die Schildkröte mindestens einmal um sich selbst gekreist ist!***
3. ***Lassen Sie ein Sechseck mit einer Kantenlänge von 125 zeichnen!***
4. ***Erstellen Sie ein Muster aus um sich kreisenden Sechsecken, die jeweils immer um 25° zueinander verdreht sind! Das Muster-Zeichnen soll erst dann beendet werden, wenn die Schildkröte wieder exakt die Ausgangsrichtung besitzt!***
5. ***Verändern Sie das Programm von 3. so, dass nur while-Schleifen zur Anwendung kommen!***

für die gehobene Anspruchsebene:

6. ***Erstellen Sie ein Muster, wie oben, aus selbstgewählten n-Ecken! Die Linien sollen getrichelt ausgeführt werden!***
7. ***Erstellen Sie Programm, dass ein Muster aus Quadraten erstellt, bei dem der Nutzer vorher eingeben darf, wie lang die Strichel- und die Nichtstrichellinien sein sollen! (Die Begrenzung auf Teilbarkeit soll und muss hier aufgehoben und umschifft werden!)***

8.8.4. Verzweigungen

übliche Verzweigungen
z.B. Auswertung von Eingaben

Verzweigungen basierend auf Turtle-Eigenschaften

Beispiel Abfrage, ob der Stift zeichnet oder nicht (unten oder oben ist)
isdown() liefert entsprechend True oder False zurück

oder Abfrage der X- bzw. Y-Koordinaten, um z.B. Überschreitungen von Grenzen auszuwerten
Auswertung über Vergleiche

weitere auswertbare Eigenschaften der Schildkröte findet man in der Zusammenstellung der Turtle-Funktionen (→ Anweisungen, Funktionen und Methoden des Turtle-Graphik-Moduls)

Aufgaben:

1. ***Erstellen Sie ein Programm, das eine getrichelte Linie zeichnet! Der Nutzer soll vorher als Eingaben die Gesamt-Länge und die Strichel-Länge eingeben! (Die Prüfung der Exaktheit der Daten soll im Programm erfolgen und ev. Fehlerhinweise ausgegeben werden! Nur dann zeichnen, wenn die Werte ok sind.))***
2.

für die gehobene Anspruchsebene:

3. ***Statt, wie in Aufgabe 1 soll eine Strich-Punkt-Linie gezeichnet werden! Der Punkt wird hier mit einer Kantenlänge von 2 festgelegt! Die Striche müssen mindestens doppelt so lang sein!***

8.8.5. Funktionen

Nutzung von Python-eigenen Funktionen bzw. Funktionen aus importierten Modulen kein Problem
Nutzung, wie üblich mit vorgestelltem Modul-Namen oder dem speziellen Importnamen

Zusammenfassung von Turtle-Anweisungen

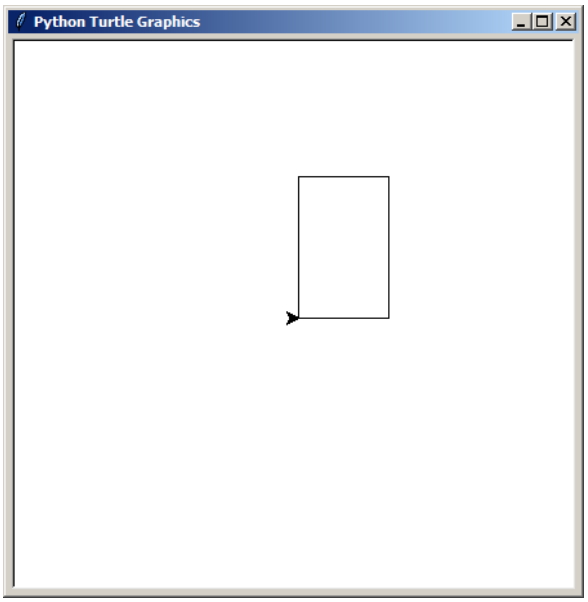
Beschreibung von Objekten z.B. ein Quadrat, n-Eck, …

übliche def-Struktur

ein Rückgabewert wird meist nicht gebraucht, kann aber – wie üblich – zurückgegeben werden

z.B. um Fehlerwerte oder Berechnungs-Ergebnisse dem übergeordneten Programm bzw. der übergeordneten Funktion mitzuteilen

```
from turtle import *

def rechteck(a,b):
    for _ in range(2):
        forward(a)
        left(90)
        forward(b)
        left(90)

rechteck(70,110)
```



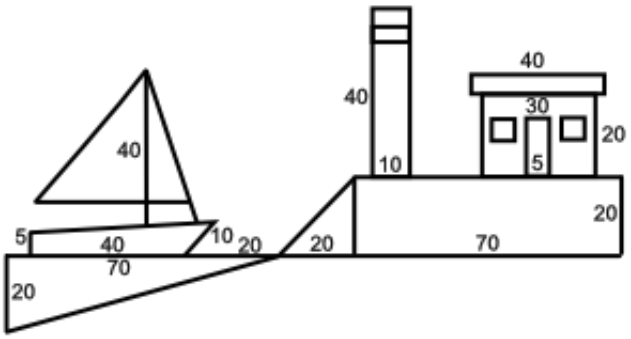
geschlossene farbige Polygonzüge

```
from turtle import *
from random import *

Farbliste=[]

def farbQuadrat(x,y,l,farbe):
    up()
    goto(x,y)
    down()
    begin_fill()
    fillcolor(farbe)
    for _ in range(4):
        forward(l)
        left(90)
    end()


for farbe in Farbliste:
    x=randint(1,400)
    y=randint(1,400)
    l=randint(1,100)
    farbQuadrat(x,y,l,farbe)
```

Aufgaben:

1. ***Lassen Sie das Haus vom Nikolaus über eine Funktion zeichnen (die Seitenlänge soll 100 betragen! Die Schildkröte soll am Schluß wieder in der Start- bzw. Ausgangs-Ausrichtung stehen! (Überlegen Sie sich, ob es Sinn macht, mit Parametern zu arbeiten!)***
2. ***Lassen Sie sich mit Ihrer Nikolaus-Haus-Funktion 5 Nikolaus-Häuser direkt nebeneinander zeichnen!***
3. ***Ändern Sie das Programm von Aufgabe 2 nun so, dass die Häuser statt des üblichen rechtwickligen Dach's nun ein gleichseitiges bekommen!***
4. ***Verändern Sie das Progamm von Aufgabe 3 nun so, dass beliebige Seitenlängen eingegeben werden können!***
5. ***Erstellen Sie ein Programm, dass mit Hilfe einer Funktion gleichSeitigesDreieck(laenge) eine Reihe von 12 Dreiecken direkt nebeneinander zeichnet!***
6. ***Machen Sie aus dem Programm von Aufgabe 5 ein neues, dass eine Reihe von 7 Dreiecken zeichnet, die immer abwechselnd nach oben und unten zeigen! Es darf nur eine Funktion gleichSeitigesDreieck(...) verwendet werden!***
7. ***Lassen Sie eine neue Programm-Version von Aufgabe 6 eine Dreiecks-Reihe zeichnen, die aus 5 Basis-Dreiecken besteht! Die Dreiecks-Spitzen sollen miteinander verbunden sein, so dass eine "Stahl-Brücke" im EIFEL-Stil entsteht! (Es darf nur die neue Funktion gleichSeitigesDreieck(...) verwendet werden!)***
8. ***Zeichnen Sie ein Sechseck aus gleichseitigen Dreiecken!***

komplexe Aufgaben:



1. *Erstellen Sie ein Programm, das die abgebildete Szene zeichnet! (Gehen Sie schrittweise vor! Verwenden Sie sinnvolle Funktionen!)*
2. *Denken Sie sich ein eigenes Muster oder eine Szene aus! Skizieren Sie diese auf ein Blatt Papier (z.B. kleinkariert) und legen Sie die Maße fest!*
   Lassen Sie die Szene vom Kursleiter abzeichnen! Setzen Sie die Szene in ein Turtle-Programm um! (Wenn das Programm funktioniert, können Sie den Mal-Effekt durch eine delay()-Funktion zwischen den Programmteilen noch verstärken!)
3. *Erstellen Sie eine Funktion wellenOrnament(), die nebenstehendes Muster erzeugt!*

4. *Gesucht ist ein Programm, dass die Turtle-Zeichenfläche mit diesem Muster umgibt!*

x.

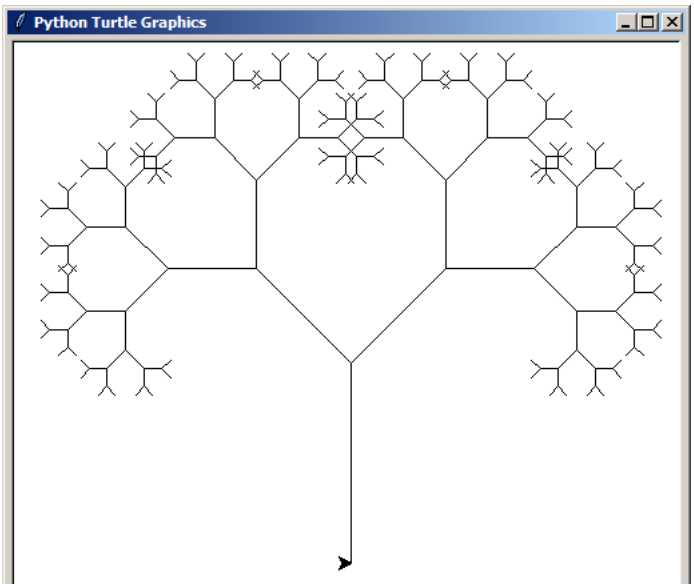
für die gehobene Anspruchsebene:

x.

8.8.6. Rekursion

```
#rekursives Zeichen eines Baumes mit Turtle
import turtle

def baum(laenge,tiefe):
    if tiefe>=0:
        turtle.forward(laenge)
        turtle.left(45)
        zweiglaenge=laenge/1.5
        baum(zweiglaenge,tiefe-1)
        turtle.right(90)
        baum(zweiglaenge,tiefe-1)
        turtle.left(45)
        turtle.backward(laenge)

### main
# Eingaben
print("Zeichnen eines selbstähnlichen Baums")
laenge=eval(input("Stamm-Länge: "))
tiefe=eval(input("Verzweigungstiefe: "))
# Beginn des Zeichnens
turtle.left(90) # Drehen zum Zeichnen nach oben
baum(laenge,tiefe)
turtle.right(90) # wieder auf die ursprüngliche Richtung drehen
```

Das Ergebnis und vor allem die Arbeit der Schildkröte beim Erstellen der Graphik ist richtig cool.



Aufgaben:

1. *Erklären Sie, warum die Schildkröte auf dem Rückweg immer die richtige Weglänge weiss!*

8.8.7. Eingaben mit der Maus

onclick(*auszuführende_Funktion*)

```
from turtle import *

def anzeige(x,y):
    print(Mausklick auf Position: ",x,",",y)

onclick(anzeige)
```

Programm nach einem Durchlauf beendet.

mit der Funktion **mainloop()** wird eine unendliche Kontrollschleife (für Eingaben / Ausgaben) erzeugt, die praktisch während der gesamten Programm-Laufzeit aktiv ist

```
from turtle import *

def anzeige(x,y):
    print(Mausklick auf Position: ",x,",",y)

onclick(anzeige)

mainloop()
```

Nutzung nun z.B. um die Schildkröte an die Klick-Position zu bewegen

```
from turtle import *

def anzeige(x,y):
    goto(x,y)

onclick(anzeige)

mainloop()
```

so erhalten wir ein kleines Zeichen-Programm

8.8.8. Und wie geht es weiter?

Die Objekt-orientierte Turtle-Programmierung folgt hinter der theoretischen Vorstellung und ersten praktischen Übungen zur Objekt-orientierten Programmierung ganz allgemein.

Windrad aus Rechtecken

```
# windrad.py
from turtle import *

def rechteck(seite): # Prozedur rechteck wird definiert
    for i in [1,2]:
        forward(seite); left(90)
        forward(seite/4); left(90)

tracer(0) # maximale Zeichengeschwindigkeit
width(2) # Zeichenstiftbreite
for i in range(1,9):
    rechteck(100)
    left(45)
```

Parkettierung (mit Rhomben)

```
# parkett.py
from turtle import *
def raute(laenge, winkel, strich_dicke, col):
    width(strich_dicke)
    color(col)
    for i in range (1,3):
       forward (laenge); left(winkel)
       forward (laenge); left(180-winkel)
tracer(0)
anzahl_reihe = 10
up(); backward(280); left(90); forward(220); right(90); down()
for i in range(1,15):
  for j in range(1,anzahl_reihe+1):
    raute(50, 45, 1, 'darkgreen')
    up(); forward(50); down()
  up(); backward(anzahl_reihe*50); right(90); forward(35); left(90); down()
```

Wald aus Bäumen

```
# baeume.py
# Wald mit Bäumen
from turtle import *
from random import random

def baum(a):
   for i in range(1,3):
       color("brown")
       fill(1)
       forward(a); left(90)
       forward(2*a); left(90)
       fill(0)
   up(); left(90); forward(2*a); left(90); forward(a); left(180); down()
   color("darkgreen")
   fill(1)
   forward(3*a); left(110)
   forward(4.39*a); left(140); forward(4.39*a); left(110)
   fill(0)
   up(); forward(a); right(90); forward(2*a); left(90); down()

tracer(0)      # max. Zeichengeschwindigkeit
up(); backward(220); left(90); forward(220); right(90); down()
# Turtle im Windwows-Fenster nach links oben setzen
a=8 # Breite Baumstamm
anzahl_x = 12  # Anzahl der Bäume in x-Richtung
anzahl_y = 8   # Anzahl der Baumreihen in y-Richtung
for j in range(1,anzahl_y+1):
  for i in range(1,anzahl_x+1):
    baum(a); up(); forward((0.5+random())*5*a); down()
  up(); backward(anzahl_x*5*a); right(90); forward(8*a); left(90); down()
```

Q: http://www.michael-holzapfel.de/progs/python/python_beisp.htm

Zeichnen eines Strauches

```
# strauch.py
# Strauch
from turtle import *
import time

def strauch(a, n):
# Die Prozedur strauch ruft sich rekursiv selber auf!
  if n>0:
     forward(a);left(30); forward(a);
     strauch(a/2,n-1); backward(a); right(30); forward(a);
     right(30); forward(a/2);
     strauch(a/2,n-1); backward(a/2); left(30); forward(a);
     strauch(a/2,n-1); backward(3*a)

tracer(0)
a=30               # Länge a
color("darkgreen")
width(2)           # Strichdicke
left(90)
strauch(a,5)       # Aufruf der Prozedur strauch
time.sleep(4)      # Programm hält 4 Sekunden an
exit()
```

Q: http://www.michael-holzapfel.de/progs/python/python_beisp.htm

Baum mit Früchten

```
# orangenbaum.py
from turtle import *

def baum(s,t):
    if t>1:
       color("brown")
       for i in range(1,3):
           fill(1)
           forward(s); left(90)
           forward(3*t); left(90)
           fill(0)
       forward(s)
       left(30)
       baum(s*0.6, t-1)
       right(55)
       baum(s*0.65, t-1)
       left(25)
       backward(s)
    elif t>=0:
       color("darkgreen")
       fill(1)
       forward(4*s); circle(2*s); backward(4*s)
       fill(0)
       color("red")
       fill(1)
       forward(3*s); circle(5); backward(3*s)
       fill(0);
       forward(s)
       left(50);
       color("brown")
       baum(s*0.6, t-1)
       right(85)
       baum(s*0.65, t-1)
       left(35)
       backward(s)

setup (width=400, height=400, startx=0, starty=0)
# Fenstergröße
title(" Orangenbaum")
# Fenstertitel

tracer(0)
left(90)
width(1)
up()
backward(150)
down()
baum(100,6)
```

Q: http://www.michael-holzapfel.de/progs/python/python_beisp.htm

Python-Stern

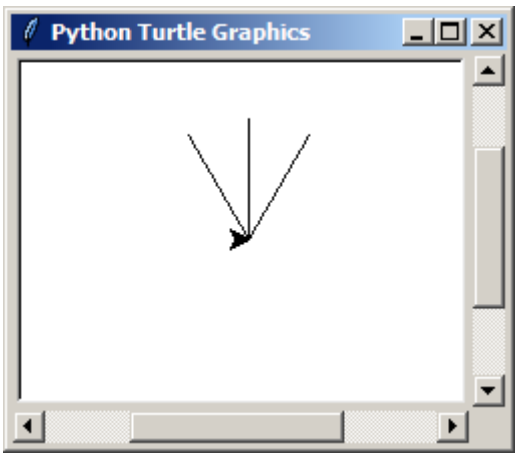
```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

8.8.9. Turteln bis zu Umfallen - rekursive Probleme schrittweise Lösen

Ein rekursives Graphik-Problem zu lösen ist nicht immer so trivial, wie es die Definition einer Rekursion suggeriert.
In Anlehnung an den gerade gezeigten Baum wollen wir nun einen Strauch rekursiv zeichnen. Er soll keinen Stamm haben und gleich mit Zweigen anfangen und statt der dichtomen (zwei-spaltigen) Teilung noch einen mittleren Zweig enthalten (also trichotom geteilt sein).
Wir fangen bei ganz einfachen Versionen an und arbeiten uns dann zu einem vollständigen Programm vor.
Im ersten Schritt erstellen wir eine einfache Funktion (noch ohne rekursive Elemente) für einen rudimentären Strauch und ein einfaches Haupt-Programm. Selbst auf Eingaben verzichten wir erst einmal um die Zeichnung ohne viel Schnick-Schnack auf den Bildschirm zu bekommen.

```
import turtle

def strauch(laenge,tiefe):
    turtle.left(30)
    turtle.forward(laenge)
    turtle.backward(laenge)
    turtle.right(30)
    turtle.forward(laenge)
    turtle.backward(laenge)
    turtle.right(30)
    turtle.forward(laenge)
    turtle.backward(laenge)
    turtle.left(30)


# Main
laenge=60
tiefe=4
turtle.left(90)  # Turtle aufrecht drehen
strauch(laenge,tiefe)
turtle.right(90)   # Turtle in die Ausgangslage zurückdrehen
```

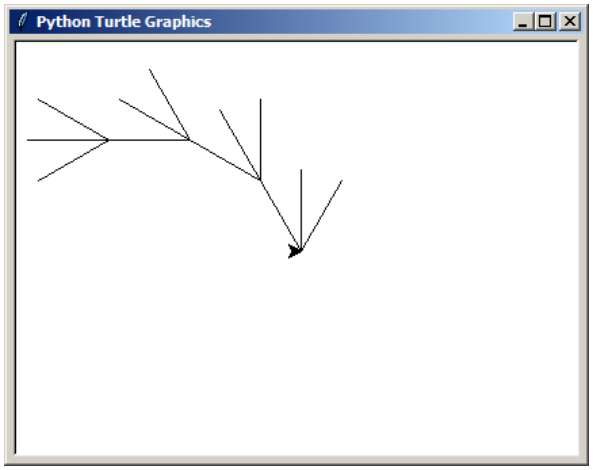
Nachdem der rudimentäre Strauch steht, können wir uns nun die Positionen heraussuchen, wo ein rekursiver Aufruf erfolgen soll. Das muss immer jeweils am Ende der 3 Zweige erfolgen. Die Länge belassen wir zuerst einmal so, wie in der ersten Rekursions-Ebene. Verkürzungen organisieren wir als Nächstes.
Was man natürlich nicht vergessen darf, ist der Rekursions-Abbruch, ansonsten zeichnet das Sysem ziemlich lange.



Wer will kann auch ersteinmal nur den ersten (linken) Zweig programmieren, damit Fehler noch gut sichtbar sind.

```
import turtle

def strauch(laenge,tiefe):
    if tiefe>0:
        # linker Zweig
        turtle.left(30)
        turtle.forward(laenge)
        stauch(laenge,tiefe-1)
        turtle.backward(laenge)
        # mittlerer Zweig
        turtle.right(30)
        turtle.forward(laenge)
        turtle.backward(laenge)
        turtle.right(30)
        # rechter Zweig
        turtle.forward(laenge)
        turtle.backward(laenge)
        turtle.left(30)


# Main
laenge=5
tiefe=0
turtle.left(90)   # Turtle aufrecht drehen
strauch(laenge,tiefe)
turtle.right(90)   # Turtle in die Ausgangslage zurückdrehen
```

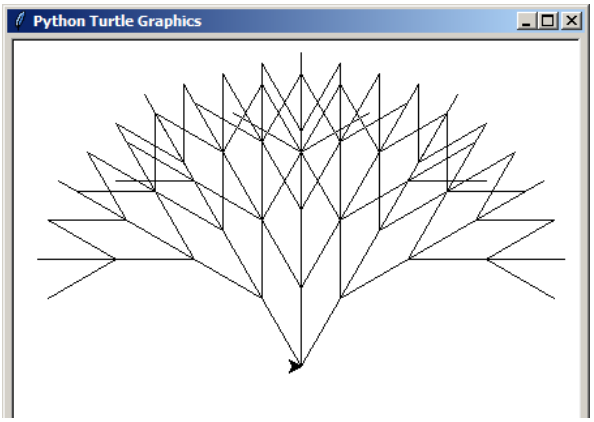
Das Graphik-Fenster zeigt saubere Linien und die Schildkröte ist auch wieder an ihrem Start-Platz angekommen. Soweit scheint die Programmierung und die Rekursion zu funktionieren.




```
import turtle

def strauch(laenge,tiefe):
    if tiefe>0:
        # linker Zweig
        turtle.left(30)
        turtle.forward(laenge)
        strauch(laenge,tiefe-1)
        turtle.backward(laenge)
        # mittlerer Zweig
        turtle.right(30)
        turtle.forward(laenge)
        strauch(laenge,tiefe-1)
        turtle.backward(laenge)
        turtle.right(30)
        # rechter Zweig
        turtle.forward(laenge)
```

```
        strauch(laenge,tiefe-1)
        turtle.backward(laenge)
        turtle.left(30)


# Main
laenge=5
tiefe=0
turtle.left(90)   # Turtle aufrecht drehen
strauch(laenge,tiefe)
turtle.right(90)   # Turtle in die Ausgangslage zurückdrehen
```

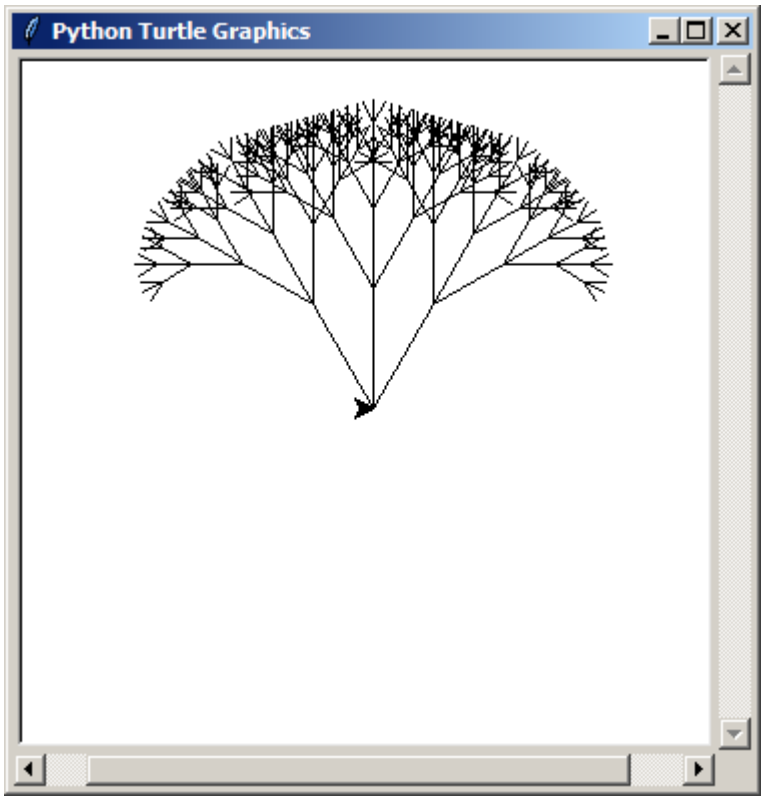
Wenn die gesamte Funktion durchdefiniert ist und es funktioniert, dann kann man noch das Rahmen-Programm anpassen. Dazu gehören sicher ordentlich abgesicherte Eingaben und die angepasste (- kürzere -) Zweiglänge für die untergeordneten Zweige.




```
import turtle

def strauch(laenge,tiefe):
    if tiefe>0:
        # linker Zweig
        turtle.left(30)
        turtle.forward(laenge)
        zweiglaenge=int(laenge/1.5)
        strauch(zweiglaenge,tiefe-1)
        turtle.backward(laenge)
        # mittlerer Zweig
        turtle.right(30)
        turtle.forward(laenge)
        strauch(zweiglaenge,tiefe-1)
        turtle.backward(laenge)
        turtle.right(30)
        # rechter Zweig
        turtle.forward(laenge)
        strauch(zweiglaenge,tiefe-1)
        turtle.backward(laenge)
        turtle.left(30)


# Main
laenge=5
tiefe=0
turtle.delay(0)  # Turtle beschleunigen
while not(laenge>=10 and laenge<=200):
    laenge=eval(input("Zweiglänge [10 .. 200]: "))
if laenge>=10 and laenge<200:
    while not(tiefe>=1 and tiefe<=10):
        tiefe=eval(input("Verzweigungen [1 .. 10]: "))
if tiefe>=1 and tiefe<=10:
    turtle.left(90)   # Turtle aufrecht drehen
    strauch(laenge,tiefe)
    turtle.right(90)   # Turtle in die Ausgangslage zurückdrehen
```

```
>>>
Zweiglänge [10 .. 200]: 60
Verzweigungen [1 .. 10]: 5
```

Ob man das Ergebnis nun als Strauch durchgehen lässt oder es eher einem Blütenstand eines Doldenblüten-Gewächs entspricht, bleibt der Phantasie des Betrachters überlassen.



Aufgaben:

1. *Ändern Sie die Funktion* strauch( ) *so ab, dass sie mit einem vom Haupt-Programm vorgegebenen Winkel zwischen den Zweigen arbeiten kann!*

für die gehobene Anspruchsebene:

2. *Ändern Sie das Programm so ab, dass die Zweige einwenig zufällig differieren! Geht das überhaupt, oder muss der Baum / Strauch immer symmetrisch sein? Überlegen Sie sich eine begründete Anwort vor dem Lösen des Problems!*

Lösen von weiteren graphischen Aufgaben und Problemen

am Ende des Abschnittes zur Turtle-Graphik gibt es eine kleine Zusammenstellung der verschiedenen Turtle-Befehle

Zeichnen eines Labyrint's (Aaron Bies)

```
import turtle, random, math

# Zeichnet zwar kein Quadrat,
# dafür aber ein perfektes Labyrinth.
#
# Der Algorithmus ist rekusiv, also
# kann es sein, dass es nach ein paar
# Sekunden abstürtzt. Einfach Skript
# neustarten, wenn das passiert.
#
# Wenn jemand weiss, warum kein Quadrat
# rauskommt, schreibt mir bitte.

turtle.delay(2)

size = 42
scale = round(380/size)
visited = []

def generate(x=0, y=0, lor=0):
    visited.append((x,y))

    order = list(range(4))
    random.shuffle(order)

    for direct in order:
        newX = max(-size/2, min(size/2,
               x+round(math.cos(direct*math.pi/2))))
        newY = max(-size/2, min(size/2,
               y+round(math.sin(direct*math.pi/2))))

        if (newX,newY) not in visited:
            turtle.goto(newX*scale, newY*scale)
            generate(newX, newY, lor+1)
            turtle.goto(x*scale, y*scale)

generate()
turtle.hideturtle()
```

Aufgaben: (relativ einfach)

1. Erstellen Sie das Haus vom Nikolaus nach der klassichen Regel, das keine Linie doppelt gezogen werden darf! (Als besondere intellektuelle Herausforderung suchen wir noch das nebenstehende Haus vom Weihnachtsmann.)
2. Zeichnen Sie eine Strichel-Linie, bei der die Strichel-Linien immer ein kleines Stück länger werden! (es reichen 10 Strichel!)
3. Erstellen Sie ein Programm, dass 10 ineinander geschachtelte Quadrate (mit dem gleichen Startpunkt) zeichnet! Das erste Quadrat soll eine Kantenlänge von 20 Pixeln haben, die nachfolgenden sollen immer um 10 Pixel verlängert werden!
4. Erstellen Sie ein "Spielfeld" für ein Tic-Tac-Toe-ähnliches Spiel aus 9 einzelnen Quadraten, die sich in den betreffenden Kanten berühren, aber nicht überschneiden! Zur Demonstration, dass es sich auch wirklich um einzelne Quadrate handelt, können Sie z.B. eine Diagonale mit einer Farbe ausfüllen.
5. Schreiben Sie ein Programm, bei dem 15 Rechtecke (Start-Seitenlängen 30 und 50 Pizel) mit einer Seiten-Verlängerung um 10 Pixel ineinander (eigentlich ja auseinander) schachtelt werden!
6. Schreiben Sie eine Funktion *linie_ohne_bewegung(länge)*, bei der eine Linie der mit der gewünschten Länge gezeichnet wird, die Schildkröte aber wieder zum Ausgangspunkt zurückkehrt!
7. Zeichnen Sie mit Hilfe der Funktion von Aufgabe () einen Strahlenkranz mit dem Radius 150 Pixel!

Aufgaben: (schon schwerer)

1.
2. Erstellen Sie ein Programm, dass aus Quadraten ein Dreh-Muster erstellt, wobei immer abwechselnd links und rechts gezeichnet wird! (quasi ein Flügel-Effekt)
3. Gesucht wird ein Programm, bei dem 12 ineinander geschachtelte Dreiecke gezeichnet werden, bei denen sich keine Kanten berühren und die Dreicke zueinander immer 20 Pixel Abstand haben! Das äußerste Dreieck soll eine Kantenlänge von 400 Pixeln haben!
4. Erstellen Sie ein Muster aus 7 (gleichseitigen) Sechsecken, die zu einem Wabenmuster angeordnet sind! Das Zeichnen eines Sechseckes ist als Funktion zu realisieren!

Aufgaben: (schwer)

1. *Gesucht wird die Simulation einer Schildkröte, die sich in einem Kasten (Kantenlänge 300 x 500) bewegt!*
2. *Erstellen Sie ein Programm für eine Teilchen-Simulation (Kugel) in einem Gefäß (Kasten 400 x 200)! Die Teilchen-Bewegung erfolgt Zufalls-gesteuert (BROWNsche Molekularbewegung). An den Wandungen wird das Teilchen nach den Gesetzen der Physik zurückgeworfen.*

Aufgaben für die gehobene Anspruchsebene: (richtig schwer)

x.

Anweisungen, Funktionen und Methoden des Turtle-Graphik-Moduls
Cheat sheet zur Bibliothek "turtle"

allgemeine Hinweise / Bemerkungen

X- und Y-Positionen können Integer oder Float-Werte sein; als Rückgabe-Werte der Funktionen sind es immer Float-Werte

None bedeutet, dass der Wert / das Argument i.A. weggelassen werden kann

zulässige Farbwerte sind: ""; "yellow"; "red"; "brown"; "green", "violet", "blue", "", "" oder z.B.: für weiss: '#ffffff' od. 255 bzw. schwarz: '#000000' od. 0;

Schildkröten-Anweisung turtle. …	**Beschreibung / Leistung / Funktion**	**Beispiele / Hinweise**
addshape(*formname, objektname*)	registriert ein definiertes Objekt und macht es unter *objektname* (als Turtle) benutzbar	*formname* kann auch eine GIF-Datei sein
addcomponent(*komponente, farbstring, hintergrundfarbe*)	fügt zu einem Objekt eine komponente mit Vordergrund- und Hintergrund-Farbe hinzu	Objekt muss vom Typ "compound" sein (→ Shape)
back(*länge*)	→ backward	
backward(*länge*)	bewegt Turtle um *länge* Pixel rückwärts	
begin_fill()	Start- / Initialisierungs-Aufruf für nachfolgendes Füllen	beenden mit → end_fill(); Farbe setzen mit → fillcolor()
begin_poly()	Start- / Initialisierungs-Aufruf für nachfolgendes Polygon-Zeichnen	aktuelle Position ist erster Polygon-Punkt
bgcolor	liefert die Hintergrundfarbe des zugrundeliegenden screen-Objektes zurück	es kann auch screen.bgcolor() genutzt werden
bgpic(bildname) bgpic()	setzt den Bildschirmname mit *bildname* oder gibt ihn zurück	es kann auch screen.bgpic() genutzt werden
bk(*länge*)	→ backward	
bye()	schließt das Zeichenfenster	
circle(*radius*) circle(*radius, sektor*) circle(*radius, sektor, schritte*)	zeichnet einen Kreis mit dem angegebenem radius; *sektor* ist der gezeichnete Kreisbogen in rad; *schritte* legt die Anzahl Polygone fest, aus der der Kreis gezeichnet werden soll	*sektor* und *schritte* können auch None sein
clear	→ clearscreen	
clearscreen	löscht die aktuelle Turtle-Zeichnung	die Turtle wird nicht bewegt od. ihre Parameter verändert!
clearstamp(*stempel_ID*)	löscht den durch *stempel_ID* (stamp-ID) gekennzeichneten Turtle-Stempel	
clearstamps() clearstamps(*anzahl*) clearstamps(*-anzahl*)	löscht die aktuelle Liste der stamp-ID's bzw. die durch anzahl bestimmten ersten bzw. letzten Turtle-Stempel	
clone()	erstellt bzw. liefert einen Klon der Turtle an der aktuellen Position	
color() color(*farbstring, hintergrundfarbe*) color(*farbstring*) color(*RGBtripel*) color(*RGBtupel, RGBtupel*)	liefert Tupel aus Vorder- und Hintergrundfarbe zurück bzw. setzt Vordergrund- und ev. auch Hintergrund-Farbe	*hintergrundfarbe* ist ein *farbstring*
colormode(*colormodus*)	setzt den *colormodus* (Wert kann 1 od. 255 sein)	begrenzt die Spanne für die RGB-Anteile
degrees(*gradzahl*)	legt die *gradzahl* / Schritte für einen Vollkreis fest	Standard sind 360°

Schildkröten-Anweisung turtle. …	Beschreibung / Leistung / Funktion	Beispiele / Hinweise
delay(*verzoegerung*) delay()	setzt die *verzoegerung* in ms oder gibt sie zurück	
distance(koordinatenpaar) distance(x_position, y_position)	liefert die Entfernung zu einem anvisierten Punkt zurück	
done()	macht letzte Anweisung rückgängig	
dot(*groesse, farbe*) dot() dot(*groesse*)	zeichnet einen Punkt mit den Parametern *groesse* und *fa*rbe an der aktuellen Position	*farbe* ist ein Farbstring oder ein RGBtripel *groesse* kann auch None sein
down()	→ pendown	
end_fill()	End- / Destruktions-Aufruf für Füll-Vorgang (der erste und letzte Punkt innerhalb der Füll-Seqenz werden am Schluß automatisch verbunden)	starten mit → begin_fill(); Farbe setzen mit → fillcolor()
end_poly	End- / Destruktions-Aufruf für Polygon-Zeichen-Vorgang	aktuelle Position ist der letzte Punkt des Polygons und wird mit dem ersten verbunden (→ begin_poly)
exitonclick()	bindet die bye-Methode an einen Mausclick auf / in das Zeichenfenster	
fd(*länge*)	→ forward	
fillcolor() fillcolor(*farbstring*) fillcolor(*RGBtupel*) fillcolor(*rotwert, gruenwert, blauwert*)	gibt aktuelle Farbwerte (als *RGBtupel*) zurück oder setzt die Werte die Füllung wird dann für die Formen zwischen begin_fill() und end_fill() benutzt	*farbstring* kann sein: Farbnamen → s.a. oben oder ein: *RGBhexadezimalcode* (Start und Ende des Füllens mit → begin_fill() bzw. end_fill())
filling()	gibt True oder False zurück jenachdem, ob der Füllmodus eingschaltet oder nicht-eingeschaltet ist	
forward(*länge*)	bewegt Turtle um *länge* Pixel vorwärts	
get_poly()	liefert das letzte gezeichnete Polygon zurück	
get_shapepoly()	liefert das Polygon der aktuellen Turtle-Form als Koordinaten-Tupel zurück	
getcanvas()	liefert Zeichenfläche als Objekt zurück	Objekt ist ein Tkinter-Objekt und kann damit weiter verwendet werden
getpen()	liefert das Turtle-Objekt sich selbst (zeigt Speicher-Adresse des Objektes)	
getscreen()	liefert die Zeichen-Fläche als Objekt zurück (zeigt Speicher-Adresse des Objektes)	einzelne Attribute lassen sich dann ändern
getshapes()	liefert eine Liste mit den Namen der möglichen Turtle-Formen zurück	
getturtle()	liefert das Turtle-Objekt sich selbst (zeigt Speicher-Adresse des Objektes)	
goto(*position*)	→ setposition	
heading		
hideturtle()	versteckt die Turtle samt Spur (z.B. bei komplexen Zeichenvorgängen)	→ isvisible
home()	setzt Turtle wieder auf die Start- / Ausgangs-Position	entspricht: turtle.setpos(0,0)
ht()	→ hideturtle	

Schildkröten-Anweisung turtle. …	Beschreibung / Leistung / Funktion	Beispiele / Hinweise
isdown()	gibt True oder False zurück jenachdem, ob der Stift zeichnet oder nicht-zeichnet	
isvisible		
left(*winkel*)	dreht Turtle um *winkel* nach links	
listen(*dummy_x_position, dummy_y_position*)	setzt den Focus auf die Zeichenfläche; die Dummy-Argumente werden für die onclick-Methode genutzt	
lt(*winkel*)	→ left	
mainloop()	startet die Ereignis-Abfrage-Schleife des übergeordneten Objektes (screen von Tkinter)	muss die letzte Anweisung in einem Turtle-Programm sein (im Script-Modus nicht notwendig)
mode(modus) mode()	setzt den modus für die Turtle-Graphik oder liefert ihn zurück modus kann sein: "logo", "world", "standard"	"standard": Ausrichtung Ost, Drehung entgegen Uhrzeiger; "logo": Ausrichtung Nord, Drehung mit Uhrzeiger
numinput(*titel, text, vorgabe, minimum, maximum*)	erzeugt ein Popup-fenster mit dem / einem *titel* und der Eingabe-Aufforderung *text*; optional können eine *vorgabe*, das *minimum* und *maximum* für die einzugebene Zahl angegeben werden	
onclick(*funktion*)	Aufruf einer Argument-losen *funktion* beim Klicken mit der *linken Maustaste*	
onclick(*funktion, maustaste, add*)	Aufruf einer zwei-argumentigen *funktion* (Click-Position) mit einer *maustaste*;	*maustaste* ist normalerweise: 1 .. linke Maustaste 2 .. 3 ..
ondrag		
onkey(*funktion, taste*)	Aufruf einer Argument-losen *funktion* beim Loslassen einer *taste*	*taste* kann auch ein Tasten-Sybol-String z.B. "space" sein
onkeypress	Aufruf einer Argument-losen *funktion* beim Drücken einer *taste*	*taste* kann auch ein Tasten-Sybol-String z.B. "space" sein
onkeyrelease(*funktion, taste*)	Aufruf einer Argument-losen *funktion* beim Loslassen einer *taste*	*taste* kann auch ein Tasten-Sybol-String z.B. "space" sein
onscreenclick		
ontimer(*funktion, zeit*)	Aufruf einer Argument-losen *funktion*, nach einer bestimmten *zeit* in ms	
pd	→ pendown	
pen(*kategorie*)	liefert Informationen zu bestimmten Kategorien über die Turtle zurück: kategorie: "shown"; "pendown"; "pencolor"; "fillcolor"; "pensize"; "speed"; "resizemode"; stretchfactor"; "outline"; "tilt"	die Rückgabewerte sind entweder True / False oder die üblichen Über- bzw. Rückgabe-Werte / -Typen der Kategorie)
Pen		

Schildkröten-Anweisung turtle. …	Beschreibung / Leistung / Funktion	Beispiele / Hinweise
pencolor() pencolor(*farbstring*) pencolor(*RGBtupel*) pencolor(*rotwert, gruenwert, blauwert*)	liefert aktuellen Farbwert als *RGBtupel* zurück bzw. setzt die Farbwerte	*farbstring* kann sein: Farbnamen → s.a. oben oder ein: *RGBhexadezimalcode*
pendown()	senkt den Stift zum Zeichnen ab (→ Turtle-Spur)	
pensize(*dicke*) pensize()	bestimmt die *dicke* der Spur bzw. liefert die Dicke der Spur zurück	*dicke* kann auch None sein
penup	hebt den Stift ab (→ keine Turtle-Spur)	
pos		
position()	liefert die aktuelle Turtle-Position als Tupel zurück	turtlePos=turtle.pos()
pu	→ penup	
radians(*gradmass*)	setzt die Messeinheit (*gradmass*) für (die nächste Aktion(en)) auf rad fest	???
RawPen		
RawTurtle		
register_shape(*formname, objektname*)	registriert ein definiertes Objekt und macht es unter *objektname* (als Turtle) benutzbar	*formname* kann auch eine GIF-Datei sein
reset()	→ resetscreen	
resetscreen	löscht aktuelle Turtle-Zeichnung und setzt alle Turtle-Parameter wieder auf die Ausgangswerte	Turtle ist wieder auf Ausgangsposition mit allen Standardwerten
resizemode(*modus*)	*modus* kann sein: "auto", "user", "noresize"	
right(*winkel*)	dreht Turtle um *winkel* nach rechts	
rt	→ right	
Screen	Tkinter-Objekt:	
ScrolledCanvas	Tkinter-Objekt:	
screensize(*bildschirmweite, bildschirmhoehe, farbstring*)	setzt die Zeichenfläche (Turtle-Bildschirm / übergeordnetes screen-Objekt) auf eine bestimme Weite (x-Ausdehnung), Höhe (y-Ausdehnung) und Hintergrundfarbe	*farbstring* kann sein: Farbnamen → s.a. oben oder ein: *RGBhexadezimalcode* es kann auch screen.screensize() genutzt werden
seth	→ setheading	
setheading(*winkel*)	legt Orientierungs-Richtung *(*als *winkel*) für die Turtle fest	im logo-Modus ist Norden bei 0°; sonst ist 0° Richtung Osten von Standard-Start nach Nord → setheading(90)
setpos(*position*)	→ setposition	
setpostion(*position*)	setzt Turtle auf die *position* *position* ist ein Vec2D oder ein Koordinaten-Paar	setpos(20, 50) setpos((20,50))
settilangle(*winkel*)		die Ausrichtung der Turtle wird nicht geändert
setundobuffer(*schritte*)	setzt oder deaktiviert den Rückschritt-Speicher (Keller-Speicher) auf *schritte*	*schritte* kann None sein

Schildkröten-Anweisung turtle. …	Beschreibung / Leistung / Funktion	Beispiele / Hinweise
setup(*x_pixel, y_pixel, x_start, y_start*) setup(*x_anteil, y_anteil*)	gibt die Ausdehnung des Zeichenfensters in Pixeln und die Start-Position vor; bei Angabe einer Float-Zahl wird der Anteil am Gesamtbildschirm gewählt	Standard sind 50% = 0.5 des Gesamt-Bildschirms
setworldcoordinates(*x_linksoben, y_linksoben, x_rechtsunten, y_rechtsunten*)	setzt die Zeichenflächen-Koordinaten (Runter-Diagonale)	
setx(*x_wert*)	setzt *x_wert* der Turtle-Position (Horizontal-Position)	
sety(*y_wert*)	setzt *y_wert* der Turtle-Position (Vertikal-Position)	
shape(*form*)	legt das Aussehen der Turtle fest; *form* kann sein: "arrow", "turtle", "circle", "square", "triangle", "classic"	*form* kann auch None sein
Shape	Tkinter-Objekt	
shapesize() shapesize(*x_faktor, y_faktor, umriss_staerke*)	liefert die aktuelle Vegrößerungsfaktoren und die Umriss-Stärke zurück oder setzt sie (im → resizemodus = "user")	*x_faktor, y_faktor* und *umriss_staerke* sind positive Werte od. None
shapetransform(*t11, t12, t21, 22*)	transformiert die Matrix der Turtle-Form	*t11, t12, t21, 22* können auch None sein
shearfactor()	gibt oder setzt	die Ausrichtung der Turtle wird nicht geändert
showturtle()	zeigt die (unsichtbare) Turtle bzw. deren Spur an (seit letztem Unsichtbarmachen)	→ isvisible
speed() speed(*geschwindigkeit*) speed(*tempostring*)	bestimmt die *geschwindigkeit* des Turtle's; Werte von 0 .. 10 werden ausgewertet, andernfalls wird 0 gesetzt; 0 .. ohne Animation	*tempostring*: "fastest" .. 0; "fast" .. 10; "normal" .. 6; "slow" .. 3; "slowest" .. 1
st()	→ showturtle	
stamp()	hinterlässt eine Turtle-Abdruck an der aktuellen Position und liefert eine stamp_ID zurück	s.a. clearstamp
Terminator		
textinput(*titel, text*)	erzeugt ein Popup-Fenster mit dem / einem *titel* und der Eingabe-Aufforderung *text*	
tilt(*winkel*)		die Ausrichtung der Turtle wird nicht geändert
tiltangle(*winkel*)		die Ausrichtung der Turtle wird nicht geändert
title(*titel*)	setzt den *titel* des Zeichenfensters	
towards(x-position, y_position) towards(koordinatenpaar)		
tracer(*anzahl, verzoegerung*) tracer(*schalter*)	*schalter* legt fest, ob die Zeichnung mit voller Geschwindigkeit (1) ohne sichtbare Turtle oder verzögert (0) mit sichtbarer Turtle erstellt werden soll	→ delay

Schildkröten-Anweisung turtle. …	Beschreibung / Leistung / Funktion	Beispiele / Hinweise
Turtle	liefert ein neues Turtle-Objekt (Konstruktor)	
turtles()	liefert eine Liste der Turtle's vom Bildschirm zurück	
TurtleScreen	Tkinter-Objekt:	
turtlesize		
undo() undo(*anzahl*)	macht die letzte bzw. die durch *anzahl* bestimmte Menge an Turtle-Aktionen rückgängig	
undobufferentries()	liefert die Anzahl der Einträge im Rückschritt-Speicher zurück	
up	→ penup	
update	zeigt den aktuellen Bildschrim z.B. während eines tracer-Modus	→ tracer
Vec2D	Tkinter-Objekt:	
width(*dicke*) width()	bestimmt die *dicke* der Spur bzw. liefert die Dicke der Spur zurück	*dicke* kann auch None sein
window_height	gibt die Fenster-Höhe (y-Ausdehung) des Zeichenfensters zurück	
window_widht	gibt die Fenster-Breite (x-Ausdehung) des Zeichenfensters zurück	
write(*schreibobjekt, bewegt, ausrichtung, schrift*)	schreibt ein *schreibobjekt* mit der *ausrichtung* ("left"; "center", "right") und der *schrift* (*schriftname, schriftgroesse, schrifttyp*) an der aktuellen Position	Bsp. für *schrift*: "Arial, 11, "normal" *bewegt* besagt, ob die Schreib-Position auf Ausgang oder Ende des Schreib-Objektes gesetzt werden soll
write_docstringdict		
xcor()	liefert die x-Koordinate der Turtle zurück	
ycor()	liefert die y-Koordinate der Turtle zurück	

u.a. Q: https://docs.python.org/3.5/library/turtle.html (hier Dokumentation nach Kategorien!)

Default- / Vorgabe-Einstellungen für Turtle-Graphik (in turtle.cfg gespeichert)

```
width = 0.5
height = 0.75
leftright = None
topbottom = None
canvwidth = 400
canvheight = 300
mode = standard
colormode = 1.0
delay = 10
undobuffersize = 1000
shape = classic
pencolor = black
fillcolor = black
resizemode = noresize
visible = True
language = english
exampleturtle = turtle
examplescreen = screen
title = Python Turtle Graphics
using_IDLE = False
```

Es existieren verschiedene Demo-Programme / -Skripts zur Turtle-Graphik. Diese können mit:

```
python –m turtledemo
```

entpackt werden.

8.8.10. Verändern des Schildkröten-Zeigers

Mit der Funktion shape() kann man die Anzeige-Form der Schildkröte anpassen. Zugelassen sind dabei die Formen:

- arrow → Pfeil
- circle → Kreis
- square → Quadrat
- triangle → Dreieck
- classic → Pfeilspitze

Die gewünschte Form wird als String in die shape()-Funktion eingegtragen.
Man kann aber auch eigene Formen festlegen und diese dann benutzen. Dazu muss man zuerst die neue Form registrieren:

```
register_shape("rhombus", ((0,5),(5,10),(10,5),(5,0)))
```

und dann später genau diese registrierte Form zuweisen:

```
shape("rhombus")
```

8.8.11. Animationen mittels turtle-Grafik

Bisher war die Dynamik unserer Zeichnungen auf die Erstellung beschränkt. Nun wollen wir uns an echte Animationen machen.
Nehmen wir als Beispiel einen Fisch. Dies könnte ein Skalar sein, der durch sein rhomische Seiten-Ansicht ein Hinkucker in jedem Aquarium ist. Wir nutzen zuerst einmal nur wenige Linien zur Veranschaulichung. Später können wir dann noch ein paar Details ergänzen.

Die Grundform könnte ein Quadrat und ein Dreieck sein. Eine andere Lösung basiert auf drei rechten Winkeln.

Da es später große und kleine Fische geben soll, definieren wir eine Funktion fisch() mit möglichen Eigenschaften. Für uns wäre das wohl die Seitenlänge und die Farbe. Auch eine Schwimm-Richtung wäre wohl angebracht.
Unser Fisch soll aus Winkeln zusammengesetzt sein. Für diese Winkel entwickeln wir zuerst auch eine Unter-Funktion. Neben Größe und Farbe interessiert uns sicher auch die Richtung.

Auch für die Winkel definieren wir Richtungen. Als Orientierung verwende ich hier die Himmels-Richtungen.
Die Winkel-Funktion soll möglichst effektiv ablaufen, da sie ja sehr häufig benutzt wird.
In welcher Zusammenstellung wir aus Winkeln einen "Fisch" machen, ist uns überlassen. Ich wähle hier ein Quadrat aus zwei Winkeln und die Schwanzflosse als ein Winkel. Da kann ich dann später vielleicht auch Fisch mit andersfarbiger Flosse erstellen.

```
from turtle import *

def winkel(laenge, richtung):
    # Richtung: von 0 bis 3 für N, O, S u. W
    # am Ende hat Turtle wieder Start-Pos. u. –Richtg.
    case richtung on:
        1:
    else: # für 0
    end
    forward(laenge)
    left(90)
    forward(laenge)
    backward(laenge)
    right(90)
    backward(laenge)
    case richtung on:

    end

# MAIN (Test-Programm)
delay(0)
winkel(100,0)
winkel(100,1)
winkel(100,2)
winkel(100,3)
```

Aufgaben:

1. *Realisieren Sie die Winkel-Funktion!*
2. *Testen Sie die Geschwindigkeit der Winkel-Funktion, indem Sie sie z.B. 10'000x aufrufen und dabei die benötigte Zeit messen!*
3. *Varieren Sie die Erstellungs-Möglichkeiten für einen Winkel! Testen Sie die Leistungsfähigkeit Ihrer Varianten! Wählen Sie die schnellste Variante aus!*

für die gehobene Anspruchsebene:

4. *Testen Sie die Leistungsfähigkeit anderer Erstellungs-Möglichkeiten für einen Fisch! Welche Variante ist warum die Günstigste?*

Meine Variante ist aus mehreren Gründen nicht sehr effektiv. Sicher haben Sie eine schnellere Variante für das Zeichnen gefunden. Diese sollten Sie nun weiter für das Erstellen eines Fisches verwenden.

```
def fisch(laenge, farbe, richtung):
    # Richtung: -1 od. 1 für links od. rechts
    pencolor(farbe)
```

An dieser Stelle darf man auch Mal hinterfragen, ob es günstig ist, bei der Winkel-Funktion immer an die Start-Position zurückzukehren oder ob man besser fährt, wenn man am Zeichen-Endpunkt stehen bleiben (und die Richtung vielleicht zurückmeldet)?
Egal, wie nun unser Fisch gezeichnet wird, jetzt kann er schon mal bewegt werden.
Für einen ersten Test nehmen wir eine Zählschleife mit

```
def vorwaerts(schritte, richtung, flaenge, ffarbe, frichtung):
    # zuerst alten Fisch löschen
    fisch(flaenge, 0, frichtung)
    # Fisch an neuer Position zeichnen
    penup()
    apos=pos().x
    if richtung == 1:
        apos+=schritte
    else:
        apos-=schritte
    goto(apos, pos().y)
    pendown()
    fisch(flaenge, ffarbe, frichtung)
```

```
# MAIN
pause=100
richtung=100
# Fisch zeichnen (Start-Situation)
laenge=50
farbe=3
richtung=1
fisch(laenge, farbe, richtung)
while true:
    delay(pause)
```

Erstellen eigener Figuren (Sprite's)

register_shape(Name, Punktliste)

Beispiel: "Schiff"
register_shape('schiff', ((0,10),(5,0),(15,0),(20,10)))

setzen der (Vordergrund-)Farbe mit:
color(Farbname) 'red', 'blue', 'black', 'green', …

für mehrere Objekte:
t1 = Turtle() erzeugt ein neues turtle-Objekt mit dem Namen t1

Zugriff über Variablennamen und dann mittels Punkt abgetrennt die zugehörigen Attribute und Methoden, z.B.:

t1.shape('schiff')
t1.goto(posx,posy)

```
from turtle import *

register_shape('schiff',((0,10),(5,0),(15,0),(20,10)))

# 1. Schiff
t1=Turtle()
t1.shape('schiff')
t1.left(90)
t1.up()
t1.color('red')
t1.goto(0,50)

# 1. Schiff
t2=Turtle()
t2.shape('schiff')
t2.left(90)
t2.up()
t2.color('green')
t2.goto(0,50)

#
for x in range(100):
    t1.goto(100-x,50)
    t2.goto(x,-50)
```

Eine Turtle-ähnliche Bibliothek ist **frog** (→ http://www.viktorianer.de/info/prog-frog.html), die sich eher spielerisch an das Problem der Graphik-Programmierung macht und deshalb mehr für jüngere Python-Programmierer geeignet ist

8.8.12. Realisierung des Snake-Spiel's mittels turtle-Grafik

Entwicklungs-Schritte an der Umsetzung durch den HPI-Python-Kurs (2020) orientiert
dazu sollte man sich die Video's von dort ansehen
hier erfolgt nur eine (leicht veränderte) Darstellung der dortigen Umsetzung

Kopf der Schlange (Snake) ist ein schwarzes Quadrat
Start-Position ist die übliche Turtle-Graphik-Fläche

```
from turtle import *

shape("square")
color("black")
penup()
```

Bewegung des Quadrates ist immer mit Positions-Veränderungen um 20 Pixel jeweils in x- bzw. y-Richtung verbunden
z.B. um eine Raster-Position nach oben, dann mit

```
goto(0,20)
```

erzeugt eine langsame Bewegung des Quadrates
gewünscht ist ev. / original im Spiel sprung-hafte Veränderung
dazu kann mit speed(0) die interne Verzögerung des Turtle-Moduls auf 0 gesetzt werden

```
speed(0)
goto(0,20)
```

ergänzt wird der erste Abschnitt um die Speicherung der (aktuellen Bewegungs-Richtung für den Schlangen-Kopf (ist als solcher schon als direction in der Turtle-Bibliothek verfügbar.
zuersteinmal setzen wir diesen initial auf "stop" (was für keine Bewegungs-Richtung steht)

```
direction="stop"
```

es folgt die Definition der Nahrung für die Schlange – hier als rote Kreise
später wird die Position zufällig erzeugt, hier zuerst einmal auf eine bestimmte Position festgelegt

```
shape("circle")
color("red")
penup()
speed(0)
goto(0,100)
```

jetzt kann man den ersten Prototypen schon mal ausprobieren

dabei stellen wir fest, das zuerst mal kurz das Quadrat erscheint, dann aber schnell vom Kreis abgelöst wird und als solcher am Ende zu sehen ist
Problem ist, das wir praktische nur eine Schildkröte steuern, die zu Anfang ein Quadrat ist, dann aber in einen Kreis gewandelt wir und als solcher nach dem programm-Ende immer noch sichtbar ist
zum getrennten Benutzen von zwei Schildkröten müssen wir Objekt-orientiert arbeiten und jede Schilldkröte als einzenes (Turtle-)Objekt definieren und benutzen

```
from turtle import *

kopf=Turtle()
kopf.shape("square")
kopf.color("black")
kopf.penup()
kopf.speed(0)
kopf.goto(0,20)
kopf.direction="stop"

essen=Turtle()
essen.shape("circle")
essen.color("red")
essen.penup()
essen.speed(0)
essen.goto(0,100)
```

damit sind die graphischen Grund-Elemente definiert
als nächstes Problem nehmen wir uns die Steuerung der Schlangen-Bewegung vor
dies soll über vier grüne Richtungs-Dreiecke in der rechten unteren Ecke des Graphik-Feldes erfolgen

jedes Richtungs-Dreieck wir als eigenständiges Turtle-Objekt realisiert

```
rechts=Turtle()
rechts.shape("triangle")
rechts.color("green")
rechts.speed(0)
rechts.penup()
rechts.goto(180,-160)
```

```
unten=Turtle()
unten.shape("triangle")
unten.color("green")
unten.right(90)
unten.speed(0)
unten.penup()
unten.goto(160,-180)
```

```
links=Turtle()
links.shape("triangle")
links.color("green")
links.left(90)
links.speed(0)
links.penup()
links.goto(160,-140)

oben=Turtle()
oben.shape("triangle")
oben.color("green")
oben.right(180)
oben.speed(0)
oben.penup()
oben.goto(160,-140)
```

die Eingaben sollen als Maus-Klicks auf die Dreiecke erfolgen
diese Klicks müssen im Programm selbst ständig ausgewertet werden

```
def interpretiere_eingabe(x,y):
    if (x>=150 and x<=170):
        if (y>=-190 and y<=-170):
            nach_unten_ausrichten()
        elif (y>=-  and y<=- ):
            nach_oben_ausrichten()
    elif (y>=-170 and y<=-150):
        if (x>=170 and y<=190):
            nach_rechts_ausrichten()
        elif (x>=-  and x<=- ):
            nach_links_ausrichten()
    kopf_bewegen()    # muss später ergänzt werden

onclick(interpretiere_eingabe)
```

bei der neuen Ausrichtung wollen wir verhindern, dass der Schlangen-Kopf sich um 180° dreht, dies würde bedeuten, die Schlange beißt sich in den Körper / Schwanz, was zum Spielende führen würde

```
def nach_unten_ausrichten():
    if kopf.direction != "up":  # dient dem Ausschluß der 180°-Drehung
```

```
        kopf.direction="down"

def nach_oben_ausrichten():
    if kopf.direction != "down":
        kopf.direction="up"

def nach_rechts_ausrichten():
    if kopf.direction != "left":
        kopf.direction="right"

def nach_links_ausrichten():
    if kopf.direction != "right":
        kopf.direction="left"
```

nun können wir die eigentliche Bewegung des Kopfes realisieren

```
def kopf-Bewegen():
    if kopf.direction=="down":
        y=kopf.ycor()     # Abfrage der aktuellen y-Position
        kopf.sety(y-20)   # Setzen der neuen y-Position

    if kopf.direction=="right":
        x=kopf.xcor()     # Abfrage der aktuellen y-Position
        kopf.setx(x+20)   # Setzen der neuen y-Position

    if kopf.direction=="up":
        y=kopf.ycor()
        kopf.sety(y+20)

    if kopf.direction=="left":
        x=kopf.xcor()
        kopf.setx(x-20)
```

an dieser Stelle kann das Ganze als Prototyp ausprobiert werden
vor allem Auffinden von Programmierfehlern, ev. Anpassungen von Koordinaten usw. usf. vornehmen
Schlange(n-Kopf) sollte sich nun über das Spielfeld bewegen lassen
(für die Analyse von (graphischen) Fehlern kann das Auskommentieren der speed()-Funktion helfen) es wird jetzt alles sehr langsam gezeichnet, aber (erste) Graphik-Fehler lassen sich schon hier besser korrigieren, als später in einem fast fertigen Programm

jetzt können wir das Essen-Aufnehmen planen
in einer speziellen Funktion wird geprüft, ob wir mit dem Kopf die Position des Essen's gefunden haben, dabei reicht ein seitliches Berühren

```
def checke_kollision_mit_essen():
    if kopf.distance(essen)<20
      # neues Essen positionieren
        # Start mit ungültiger Position auf Steuerung
        x=160
        y=-160
        # zufällig neue Position erstellen und prüfen, bis sie funktioniert
        while (x>=-140 and y<=-140):
            x=randint(-9,9)*20
            y=randint((-9,9)*20
        essen.penup()
        essen.speed(0)
        essen.goto(
      # Schlange verlängern
```

für die obige Funktion benötigen wir für das Wachsen der Schlange eine Liste der Segmente
Diese müssen wir natürlich vorher leer anlegen.
Jedes Segment soll dann ein eigenständiges Turtle-Objekt sein

```
segmente=[]
```

die Kollision mit dem Fenster- / Spielfeld-Rand ist ein Abbruch-Kriterium. Der Speiler hat dann verloren.

```
def check_kollision_mit_fensterrand():
    if (kopf.xcor()<-190 or kopf.xcor()>190
        or kopf.ycor()<-190 or kopf.ycor>190):
      # Neustart des Programms
        penup()
        speed(0)
        goto(0,0)
        direction="stop"

        segmente_entfernen()
      # ev. Ausgabe über verlorenes Spiel
```

Da das Neustarten des Programm's auch noch gebraucht wir, wenn man z.B. über den eigenen Schwanz läuft, werden die relevaten Befehle in eine eigene Funktion gepackt und diese in check_kollision_mit_fensterrand() einegbaut.

```
def spiel_neustarten():
    penup()
    speed(0)
    goto(0,0)
    direction="stop"


def check_kollision_mit_fensterrand():
    if (kopf.xcor()<-190 or kopf.xcor()>190
        or kopf.ycor()<-190 or kopf.ycor>190):
      # Neustart des Programms
        spiel_neustarten()
        segmente_entfernen()

      # ev. Ausgabe über verlorenes Spiel
```

```
def check_kollision_mit_segmenten():
    for segment in segmente:
        if segment.distance(kopf)<20:
            spiel_neustarten()
```

```
def koerper_bewegen():
    for index in range(len(segmente)-1,0,-1):
```

Hauptprogramm

- definitionen
- onclick(interprtiere_eingabe)
- checke_kollision_mit_essen()
- check_kollision_mit_fensterrand()
- koerper_bewegen()
- kopf_bewegen()
- checke_kollision_mit_segmenten()

```
def wiederhole_spiellogik():
    checke_kollision_mit_essen()
    check_kollision_mit_fensterrand()
    koerper_bewegen()
    kopf_bewegen()
    checke_kollision_mit_segmenten()
```

fertiges Programm (zusammengesammelt)

```
from turtle import *
from random import randint

def interpretiere_eingabe(x, y):
    if (x >= 150 and x <= 170 and y >= -190 and y <= -170):
        nach_unten_ausrichten()
    elif (x >= 170 and x <= 190 and y >= -170 and y <= -150):
        nach_rechts_ausrichten()
    elif (x >= 150 and x <= 170 and y >= -150 and y <= -130):
        nach_oben_ausrichten()
    elif (x >= 130 and x <= 150 and y >= -170 and y <= -150):
        nach_links_ausrichten()
    # ...

onclick(interpretiere_eingabe)

def kopf_bewegen():
    if kopf.direction == "down":
        y = kopf.ycor()
        kopf.sety(y - 20)
    elif kopf.direction == "right":
        x = kopf.xcor()
        kopf.setx(x + 20)
    elif kopf.direction == "up":
        y = kopf.ycor()
        kopf.sety(y + 20)
    elif kopf.direction == "left":
        x = kopf.xcor()
        kopf.setx(x - 20)

def checke_kollision_mit_essen():
    if kopf.distance(essen) < 20:
```

```
        # Teil 1: Essen an neue Position bewegen
        x=160
        y=-160
        # zufällig neue Position erstellen und prüfen, bis sie funktioniert
        while (x>=-140 and y<=-140):
            x=randint(-9,9)*20
            y=randint(-9,9)*20
        essen.goto(x,y)
        # Teil 2: Schlange wachsen lassen
        neues_segment=Turtle()
        neues_segment.shape("square")
        neues_segment.color("yellow")
        neues_segment.penup()
        neues_segment.speed(0)
        # neues_segment.goto(kopf.xcor(),kopf.ycor())
        neues_segment.goto(0,0)
        neues_segment.direction=kopf.direction
        segmente.append(neues_segment)

def spiel_neustarten():
        # Kopf in der Mitte platzieren
        kopf.goto(0,0)
        # Richtung auf "stop" setzen
        kopf.direction="stop"
        segmente_entfernen()
        # Ausgabe, dass Spielrunde vorbei ist
        print("Leider verloren! Auf ein Neues ...")


####################Hauptprogramm###################

# Schlange
kopf=Turtle()
kopf.shape("square")
kopf.color("black")
kopf.penup()
kopf.speed(0)
kopf.goto(0,20)
kopf.direction="stop"

segmente=[]


# Essen
essen=Turtle()
essen.shape("circle")
essen.color("red")
essen.penup()
essen.speed(0)
essen.goto(0,100)

# Steuer-Region
rechts = Turtle()
rechts.shape("triangle")
rechts.color("green")
rechts.speed(0)
rechts.penup()
rechts.goto(180, -160)

unten = Turtle()
unten.shape("triangle")
unten.color("green")
unten.right(90)
unten.speed(0)
```

```
unten.penup()
unten.goto(160, -180)

links = Turtle()
links.shape("triangle")
links.color("green")
links.right(180)
links.speed(0)
links.penup()
links.goto(140, -160)

oben = Turtle()
oben.shape("triangle")
oben.color("green")
oben.left(90)
oben.speed(0)
oben.penup()
oben.goto(160, -140)
```

8.9. Musik mit python

8.9.1. Musik mit Board-Mitteln

siehe dazu im Skript "Muenker-Intro_Python_DSP.pdf"

8.9.2. Musik mit python-sonic

externes Modul

aktuelle Version unter:
https://github.com/gkvoelkl/python-sonic

sehr gut mit dem Raspberry Pi realisierbar, hier ist die Version "sonic pi" schon in mehreren Betriebssystem-Distributionen vorinstalliert

Kopfhörer können direkt angeschlossen werden

einfacher Einstieg mit einfachen Ergebnissen möglich (wird in diesem Skript auch den wesentlichen Teil der Besprechung ausmachen)

relativ komplexes System
sehr Leistungs-fähig
für Anfänger und nicht so Noten-affinen Nutzern schnell zu / sehr kompliziert
Fehler-Findung recht schwierig (Welche Note ist wann und wie falsch?)

8.10. das Modul "pygame"

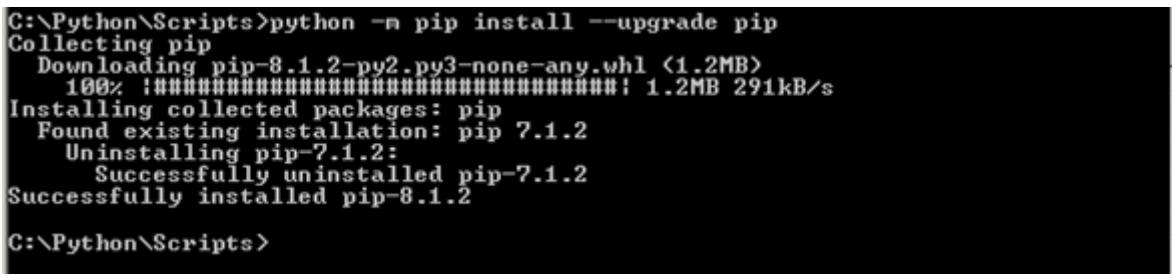
Da Python nicht direkt auf die Hardware zugreifen kann, dieses aber eigentlich für viele Programme und besonders schnelle Spiele usw. notwendig ist, bietet das Modul pygame indirekte Zugriffe und Funktionen an. So bleiben wir bei Programmieren auf Python-Ebene und die Programme können diverse Funktionen moderner Multimedia-Hardware nutzen. Pygame sorgt auch dafür, dass es uns völlig egal ist, welche konkrete Graphik- oder Sound-Karte (oder entsprechende onboard-Version) auf unserem Rechner installiert ist. Da ist Sache von pygame und braucht uns nicht zu kümmern.

8.10.0. Quellen und Installation

Als Download-Quellen sind einmal die offizielle pygame-Seite im Internet (www.pygame.org) und eine inoffizielle – aber scheinbar bestens gepflegte – Quelle für alle möglichen Module zu Python (http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame) zu nennen.
Eigentlich sollte es mit den Installations-Dateien (msi-Dateien für Windows-Systeme) eine Installtion gelingen. Bei mir funktionierten das nur mit einem Python-2-System.
Bei der letzteren Quelle sind sogenannte whl-Dateien zu downloaden, die mit dem internen pip-Progamm zu installieren sind. Das war bei mir der einzige funktionierende Weg.
Je nach Betriebssystem-Type (32- oder 64bit) und Python-Version findet man dort die passende whl-Datei.
Am einfachsten ist es, diese gleich in den Scripts-Ordner der lokalen Python-Installation zu downloaden. Wenn die Datei vom Browser woanders hin gespeichert wird, dann kopiert man sie einfach in das Verzeichnis Scripts.
Nun brauchen wir eine Eingabe-Aufforderung (Konsole) in dem Ordner. Dazu geht man im Windows-Explorer (od. Arbeitsplatz) zum Python- und dann in den Scripts- Ordner.
Mittels Hochstell-Taste (Shift) und rechter Maustaste kann man sich eine Eingabe-Aufforderung öffnen.
Nun muss man die Befehle sehr exakt eintippen, da sonst Fehlermeldungen folgen oder gar nichts passiert.
Zuerst aktualisieren wir das pip-Programm:

```
python –m pip install --upgrade pip
```

Das Ergebnis sollte dann in etwa so aussehen. Die Versionen werden sich sicher schon wieder unterscheiden.



Nun können wir die passende whl-Datei installieren:

```
pip3 install pygame-……whl
```

Statt der Punkte müssen da die exakten Angaben der whl-Datei verwendet werden. Man kann sich einen Dateiname im Explorer kopieren und dann über die rechte Maustaste in der Konsole einfügen.
Kommen Fehler-Meldungen, dann bleibt nur ein erneuter Versuch. Allerdings ist jetzt (meist) ein upgrade nowendig.

```
pip3 install --upgrade pygame-……whl
```

Wenn alles geklappt hat, dann sollte eine Bestätigungs-Meldung in der Konsole erscheinen.

```
C:\Python\Scripts>pip3 install --upgrade pygame-1.9.2a0-cp35-none-win32.whl
Processing c:\python\scripts\pygame-1.9.2a0-cp35-none-win32.whl
Installing collected packages: pygame
  Found existing installation: pygame 1.9.2a0
    DEPRECATION: Uninstalling a distutils installed project (pygame) has been de
precated and will be removed in a future version. This is due to the fact that u
ninstalling a distutils project will only partially uninstall the project.
    Uninstalling pygame-1.9.2a0:
      Successfully uninstalled pygame-1.9.2a0
Successfully installed pygame-1.9.2a0

C:\Python\Scripts>
```

Jetzt steht ersten Tests nichts mehr im Weg. Ein Neustart des Rechners ist zu empfehlen, damit die integrierten DLLs ordnungsgemäß geladen werden.

Bei aktuellen Python-Installationen kann man es zuerst einmal mit:

```
pip3 install pygame
```

probieren.

8.10.1. Ausprobieren / Testen / Grundlagen

Bevor wir uns nun auf die Möglichkeiten von pygame einlassen, testen wir erst einmal die Installation. Das nochfolgende Programm erzeugt nur ein schwarzes Fenster und warten auf das reguläre Schließen über den zugehörigen Fenster-Knopf.

```
import pygame

pygame.init()
screen=pygame.display.set_mode([640,480])
aktiv=True
while aktiv:
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            aktiv=False
pygame.quit()
```

Sollte das programm nicht funktionieren, sollte man nochmals pygame installieren / upgraden. Bleibt dieses erfolglos, dann bleibt nur die große Suche nach Hilfe im Internet oder ein Überspringen dieses Kapitels.
Nach meinen ersten anfänglichen Schwierigkeiten mit pygame fand ich das unten folgende Test-Programm für die pygame-Schnittstellen von der Seite www.spieleprogrammierer.de. Auf dieser steht auch ein online-Tutorial zur Verfügung.

```
# Pygame-Modul importieren.
import pygame

# Überprüfen, ob die optionalen Text- und Sound-Module geladen werden konnten.
if not pygame.font: print('Fehler pygame.font Modul konnte nicht geladen werden!')
if not pygame.mixer: print('Fehler pygame.mixer Modul konnte nicht geladen wer-
den!')

def main():
    # Initialisieren aller Pygame-Module und
    # Fenster erstellen (wir bekommen eine Surface, die den Bildschirm repräsen-
tiert).
    pygame.init()
    screen = pygame.display.set_mode((800, 600))

    # Titel des Fensters setzen, Mauszeiger nicht verstecken und Tastendrücke wie-
derholt senden.
    pygame.display.set_caption("Pygame-Tutorial: Grundlagen")
    pygame.mouse.set_visible(1)
    pygame.key.set_repeat(1, 30)

    # Clock-Objekt erstellen, das wir benötigen, um die Framerate zu begrenzen.
    clock = pygame.time.Clock()

    # Die Schleife, und damit unser Spiel, läuft solange running == True.
    running = True
    while running:
        # Framerate auf 30 Frames pro Sekunde beschränken.
        # Pygame wartet, falls das Programm schneller läuft.
        clock.tick(30)

        # screen-Surface mit Schwarz (RGB = 0, 0, 0) füllen.
        screen.fill((0, 0, 0))

        # Alle aufgelaufenen Events holen und abarbeiten.
        for event in pygame.event.get():
            # Spiel beenden, wenn wir ein QUIT-Event finden.
            if event.type == pygame.QUIT:
                running = False

            # Wir interessieren uns auch für "Taste gedrückt"-Events.
            if event.type == pygame.KEYDOWN:
                # Wenn Escape gedrückt wird, posten wir ein QUIT-Event in Pygames
Event-Warteschlange.
                if event.key == pygame.K_ESCAPE:
                    pygame.event.post(pygame.event.Event(pygame.QUIT))

        # Inhalt von screen anzeigen.
        pygame.display.flip()


# Überprüfen, ob dieses Modul als Programm läuft und nicht in einem anderen Modul
importiert wird.
if __name__ == '__main__':
    # Unsere Main-Funktion aufrufen.
    main()
```

Q: www.spieleprogrammierer.de

Links:
http://www.spieleprogrammierer.de
www.pygame.org (die offizielle pygame-Seite; offizielle Installations-Dateien für alle Betriebssysteme)
http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame (whl-Dateien für die Installation über pip)

8.10.1. Sound mit pygame

Sound-Dateien erzeugen (bzw. aufnehmen) und abspielen
Python bzw. das Modul pygame kann mit folgenden Sound-Dateien arbeiten:

- **WAV** Wave-Dateien
-
- **MP3**
- **WMA** WindowsMedia
- **OGG** Ogg Vorbis-Dateien komprimierte Sound-Dateien mit sehr guter Dynamik und geringen Konvertierungs-Verlusten

Sound erzeugen über einen einfachen integrierten Synthesizer

8.10.1.1. Sound-Dateien abspielen

```
# Initialisierung der Soundverarbeitung
import pygame
pygame.init()
pygame.mixer.init()
…
# Auswahl der Datei
dateiname="musik.wav"

# eigentliches Abspielen -- Variante 1
soundobjekt1=pygame.mixer.Sound(dateiname)
soundobjekt1.play()
…
# erneutes Abspielen -- Variante 1
soundobjekt1.play()
…

…
# eigentliches Abspielen -- Variante 2
pygame.mixer.music.load(dateiname)
pygame.mixer.music.play()
…
# erneutes Abspielen -- Variante 2
pygame.mixer.music.load(dateiname)
pygame.mixer.music.play()
…
```

```
# Initialisierung der Soundverarbeitung
import pygame
pygame.init()
pygame.mixer.init()
…
# Auswahl der Datei
dateiname="musik.wav"

# eigentliches Abspielen -- Variante 1
soundobjekt1=pygame.mixer.Sound(dateiname)
soundobjekt1.play()
…
# erneutes Abspielen -- Variante 1
soundobjekt1.play()
…

…
# eigentliches Abspielen -- Variante 2
pygame.mixer.music.load(dateiname)
pygame.mixer.music.play()
…
# erneutes Abspielen -- Variante 2
pygame.mixer.music.load(dateiname)
pygame.mixer.music.play()
…
```

8.10.1.2. Sound-Dateien erzeugen / aufnehmen

8.10.1.3. Musik aus dem Synthesizer

8.10.2. Grafik mit pygame

Haupt-Programm (Starter) – oft main.py genannt:

```
import pygane, sys, ObjektModul

# Definitionen / Konstanten
fensterBreite=800
fensterHoehe=600

# Initialisierungen
pygane.init()
```

```
fenster=pygame.display.set_mode(fensterBreite,fensterHoehe)
spites=pygame.sprite.Group()
anzeigeObjekt= …      # aus ObjektModul
sprites.add(anzeigeObjekt)
zeit=pygame.time.Clock()

# Hauptschleife
while True:
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            pygame.quit()
            sys.exit()
        # ev. noch andere Events / Eingaben abfragen

        fenster.fill((255,255,255))
        sprites.aktualisieren()
        sprites.draw(fenster)

        pygame.display.flip()
        zeit.tick(30)
```

Ein dazu gehörendes Objekt-Modul könnte dann so aussehen:

```
class AktionObjekt(pygame.sprite.Sprite) # erbt von pygame…

    # Initialisierung
    def __init__(self, fensterBreite, fensterHoehe):
        super().__init__()
        self.fensterB=fensterBreite
        self.fensterH=fensterHoehe
        self.bild=pygame.image.load("???.png")
        self.bildFlaeche=self.image.rect()
        self.rect.center=(self.fensterB/2,self.fensterH/2)

    # Funktionen (Bewegungen, …)
    def aktualisieren(self):
        eingabeTaste=pygame.key.get_pressed()

        # Eingabe-Auswertung und Aktionen, Funktionen, …
        if eingabeTaste[pygame.K_RIGHT]:
            self.rect.x+=10
        if eingabeTaste[pygame.K_LEFT]:
            self.rect.x-=10
        if eingabeTaste[pygame.K_UP]:
            self.rect.y+=10
        if eingabeTaste[pygame.K_DOWN]:
            self.rect.y-=10

        # Ojekt einfangen
        self.rect.clamp_fp(pygame.Rect(0,0,self.fensterB,self.fensterH))
```

Dieses Modul muss dann natürlich im Haupt-Programm (statt: ***ObjektModul***) importiert werden. Bei mehreren unterschiedlichen Objekten – die andere Funktionen usw. gebrauchen – sind auch mehrere Objekt-Module notwendig.

```
…
```

8.11. Objekt-orientierte Programmierung

Die **Objekt-orientierte** Programmierung ist neben der **imperativen** und der **deklarativen** ein weiteres **Programmier-Paradigma**. Paradigmen beschreiben grundsätzliche Denk- und Arbeitsweisen. In der Informatik verstehen wir darunter vor allem den Programmier-Stil. Insgesamt haben sich schon viele verschiedene Programmier-Paradigmen herauskristallisiert. Praktisch kann man mit jeder Herangehensweise ein Programm für ein spezielles Problem erstellen. Dabei sind Aufwand und die Qualität des Programm's oft sehr unterschiedlich. Ziel ist aber immer ein Fehler-freies, effektives, gut lesbares, Redundanz-freies, modulares und Nebenwirkungs-freies Programm zu entwickeln. Gute Programmierer wählen immer eine spezielle Programmiersprache – die meist unterschiedlichen Paradigmen zugehören – um ein Problem zu lösen.
Viele Programmier-Sprachen lassen sich mit mehreren Herangehensweisen benutzen. So können wir Python imperativ und Objekt-orientiert verwenden.
In der letzten Jahren sind viele völlig neuartige Paradigmen entwickelt worden. Hier seien einige kurz genannt:

neuartige Programmier-Paradigmen

- **Komponenten-orientierte Programmierung**
- **Agenten-orientierte Programmierung**
- **Aspekt-orientierte Programmierung**
- **generative Programmierung**
- **generische Programmierung**
- **Subjekt-orientierte Programmierung**
- **Datenstrom-orientierte Programmierung**
- **Graphen-ersetzende Programmierung**
- **konkatenative Programmierung**
- **multi-paradigmatische Programmierung**
-

Bei der
beim Objekt-orientiuerten Ansatz geht es darum Eigenschaften (Attribute) und Prozeduren (Methoden), die zu einem Ding (Objekt) zugehören gemeinsam zu verwalten und zu programmieren
bei Modulen ist die Zusammenfassung eher inhaltlich gemein z.B. mathematische Funktionen oder Funktionen zur Zeit (Berechnungen, Umrechnungen, …)

hier nur grobe Verwendung und Nutzung

HelloWorld
mit 2 Texten Begrüßung und Verabschiedung

Bevor wir uns in die Objektwelt von Python begeben, wollen wir erst einmal klären mit was wir es hier zu tun haben.
Objekte sind in der Realität irgendwelche konkreten Dinge, wie z.B. der Mitschüler Friedrich, der Lehrer Müller, der Porsche von Frau Geizig oder die gelbe Blume auf dem Küchentisch.

In der Informatik werden ebenfalls Objekte der Realität modelliert. Wir schaffen uns also ein informatisches Objekt, um Informationen rund um das Real-Objekt herum elektronisch / informatisch bearbeiten zu können.
Praktisch jedes Objekt kann man einer oder mehrerer Gruppen zuordnen. In der Informatik heißen die Gruppen Klassen. Objekte mit gleichen gemeinsamen Merkmalen werden in einer Klasse bearbeitet. Im Prinzip kann man zu einer Klasse immer spezielle Mengen / Arten von Informationen verarbeiten. Gerade deshalb lohnt es sich nicht für jedes Objekt die einzelnen Informations-verarbeitenden Vorgänge einzeln zu programmieren, sondern es ist effektiver, sie irgendwie gemeinsam zu erstellen. Eine Möglichkeit so etwas zu machen, haben wir mit den Funktionen kennengelernt. Sie liefern aufgrund bestimmter übergebener Argumente einen oder mehrere Werte (Ergebnisse) zurück, egal für welche konkreten Variable oder Wert dies erfolgt.

Definition(en): Objekt (allg.)
Objekte sind Dinge der Realität, die miteinander Informationen austauschen (kommunizieren).

Definition(en): Objekt (informatisch)
Informatische Objekte sind Abbildungen real-existierender Dinge in einem Computer-Modell.

Wenn wir ein Objekt einer Klasse zugeordnet haben oder aus ihr heraus entwickeln, dann nennen wir das Objekt eine Instanz (der Klasse).
Jedes Objekt hat eine bestimmte Menge von Eigenschaften, wie z.B. einen Namen oder eine Farbe usw. usf. In Klassen werden die gemeinsamen Arten von Eigenschaften Attribute genannt. Bei der Notierung hat sich punktierte Schreibung durchgesetzt. So wird der Name von dem Lehrer-Objekt **Müller** über die Notierung **Müller.Name** erreicht.
Da alle Instanzen z.B. der Lehrer-Objekte – also der Klasse Lehrer – über einen Namen verfügen, wird der Klasse das Attribut Name zugeordnet. Geschrieben wird dann **Lehrer.Name**.
Neben den charakterisierenden Eigenschaften eines Objektes (eben die Attribute) brauchen wir noch Verfahren z.B. zum Abfragen oder Ändern des **Lehrer.Name**'s. Die Verfahren werden Methoden genannt. Praktisch gehört fast immer zu jedem Attribut einer Klasse eine setzende und eine abfragende Methode. Die beiden heißen fast immer SET und GET.
Die Notierung würde dann **Lehrer.Name.set(…)** bzw. **Lehrer.Name.get(…)** lauten. GET und SET sind also praktisch Funktionen.

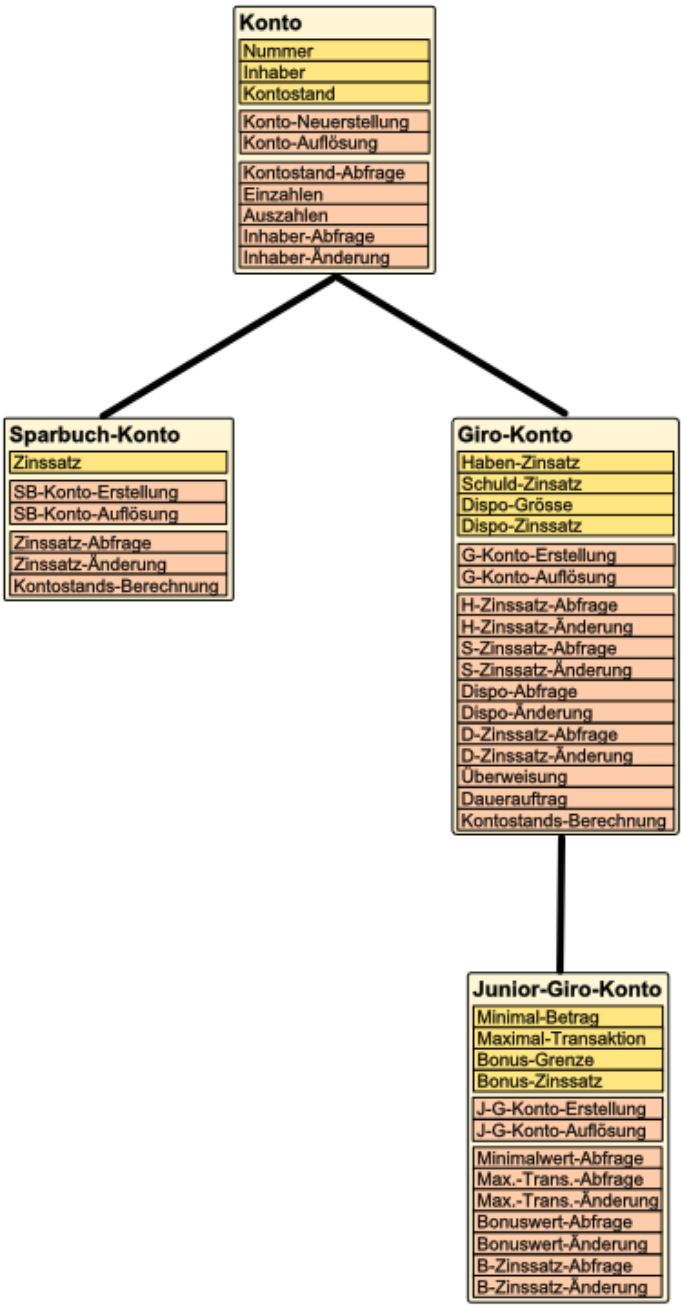
Definition(en): Klassen
Klassen sind die allgemeinen Beschreibungen / Bildungs-Vorlagen für (informatische) Objekte.
Eine Klasse ist ein Bildungsschema für ähnliche / vergleichbare Objekte.

Definition(en): Attribute
Attribute sind die individuellen Eigenschaften von Objekten.

Definition(en): Methoden
Methoden sind die Funktionen / Arbeitsmöglichkeiten / … von Objekten.

Zum besseren Verständnis und für eine genauere Übersicht werden Objekte und Klassen in verschiedenen Schemata dargestellt.

Wie das Objekt intern funktioniert, also wie es z.B. den Namen abspeichert oder beim Konto den aktuellen Konto-Stand berechnet bleibt für die Umgebung uninteressant und (meist) auch unsichtbar.



beim Objekt-orientierten Ansatz geht es darum Eigenschaften (Attribute) und Prozeduren (Methoden), die zu einem Ding (Objekt) gehören gemeinsam zu verwalten und zu programmieren
bei Modulen ist die Zusammenfassung eher inhaltlich gemein z.B. mathematische Funktionen oder Funktionen zur Zeit (Berechnungen, Umrechnungen, …)

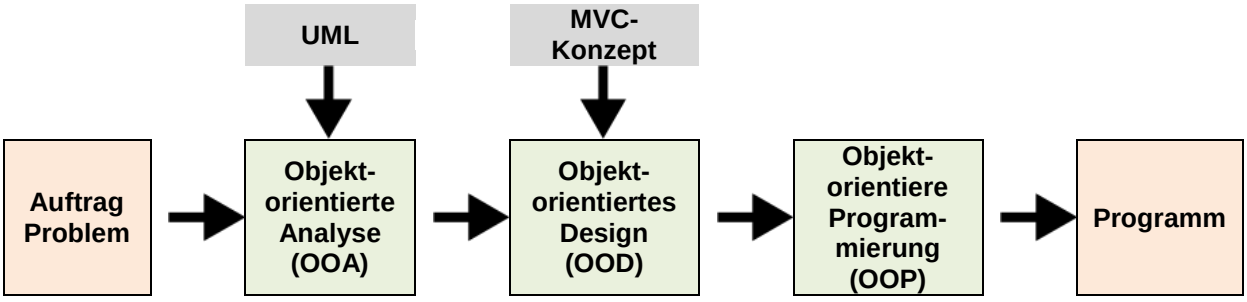
Instanz-Variablen sind die Attribute eines Objektes

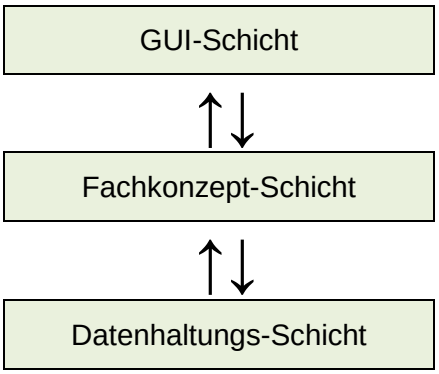
Objekt-orientierte Programmierung (OOP)

Konzepte der OOP

- **Abstraktion**
- **(Daten-)Kapselung**
- **Feedback**
- **Persistenz**
- **Polymorphie**
- **Vererbung**

Definition(en): Objekt-orientierte Programmierung (OOP)
Die Objekt-orientierte Programmierung ist eine Form der Software-Erstellung auf der Basis von reelen Grund-Strukturen und deren Umsetzung in informatische Modelle.
Unter Objekt-orientierter Programmierung versteht man das Programmier-Paradigma, das Daten und Programm-Code in übersichtlichen Einheiten – Klassen genannt – kapselt / behandelt.
Objekt-orientierte Programmierung ist eine Methode der Modularisierung von Programmen, bei der in Anlehnung an Gruppen von realen Sachverhalten informatische Modelle erzeugt und verarbeitet werden.
Unter der Objekt-orientierten Programmierung versteht man das Programmieren von komplexen Software-Systemen, bei dem Objekte und deren Kommunikation untereinander im Vordergrund steht.





Sicht des Nutzers auf die Daten (Daten-Eingabe und –Anzeige; Bedienung)

Datenverarbeitung; Fach-Modell;

Daten-Speicherung z.B in einer Datenbank

Design pattern – Entwurfsmuster

Analyse-Situation	**Lösung / UML-Diagramm**	
Gemeinsame Attribute und Methoden	**abstrakte Oberklasse**	
mehrere Klassen haben einige gemeinsame Attribute in der Oberklasse klassische Vererbungs-Situation der **Verallgemeinerung**		
spezielle Unterklasse	**konkrete Oberklasse**	
zu einer (konkreten) Klasse kommt eine Unterklasse mit		
Daten einer Beziehung festhalten **Koordination von Objekten**	**Assoziation über eine vermittelnde Klasse**	
Container / Kollektion und ihr Inhalt	**Aggregation in einer Sammlung**	
Beschreibung **Registrierung von Ereignissen**	**Aggregation Beschreibungen**	
Attribut-Werte weitergeben	**Aggregation zur Zuordnung gleicher Attribut-Werte**	

nach Q: http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/ooa-ood/designpattern.htm

8.11.x. Objekt-orientierte Programmierung mittels Turtle-Grafik

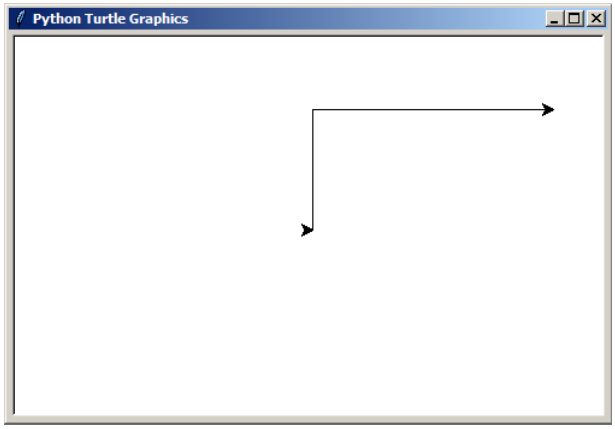
Das Modul **turtle** ermöglicht auch die Objekt-orientierte Programmierung von zeichnenden Schildkröten. Der Umgang mit Objekten gehört heute für eigentlich alle Module zum Standard. Moderne Programme für grafische Bedienoberflächen – wie Windows, iOS, Android oder Linux-KDE bzw. Linux-gnome – lassen sich anders gar nicht mehr benutzen. In unseren ersten Turtle-Übungen haben wir nur unterschwellig mit einem Turtle-Objekt gearbeitet (→ 8.8. Turtle-Graphik – ein Bild sagt mehr als tausend Worte). Es wurde gleich für uns automatisch angelegt und wir konnten es benutzen, ohne dass wir uns um irgendwelche "Objekte" einen Kopf machen mussten. Das war zu diesem Zeitpunkt auch sinnvoll, da wir uns einfach (und unkompliziert) die grafische Seite der Programmierung ansehen wollten.
Nun gehen wir aber einen deutlichen Schritt in Richtung moderne Programmierung.
Übrigens werden Sie merken, dass die meisten Programme aus dem Internet auch immer Objekt-orientiert programmiert sind.
Wir wollen zuerst einmal zwei Schildkröten auf dem Bildschirm erzeugen. Diese sollen dann eigenständig Bewegungen ausführen.
Zuerst brauchen wir – wie üblich – das Modul turtle, welches wir hier importieren.

```
from turtle import *

t1=Turtle()
t2=Turtle()
```

Nun erstellen wir uns zwei (unabhängige) Turtle-Objekte. Üblich ist die Notierung der Klassen-Namen mit einem Großbuchstaben beginnend.
Lassen wir das Programm an dieser Stelle laufen, sehen wir allerdings nur eine Schildkröte. Das liegt daran, dass beim Erstellen die gleichen Vorgaben benutzt wurden. Beide Schildkröten haben die gleiche Farbe und liegen an der gleichen Position. Der Konstruktor – der aus der Klassen-Vorgabe ein konkretes Objekt erzeugt hat, hatte nur diese Vorgaben.
Um nun zu zeigen, dass wir es mit zwei eigenständigen Objekten zu tun haben, können wir die eine Schildkröte mit unseren typischen Funktionen bewegen.Die Klasse Turtle wird also zweimal bemüht und die beiden Objekte t1 und t2 zu erstellen.
Allerdings müssen wir jetzt immer sagen, welche Schildkröte wir bewegen wollen. In der Objekt-orientierten Programmierung wird dazu üblicherweise die Punkt-Notierung genutzt. Wir geben zuerst das benutzte Objekt an und dann hinter einem Punkt die auszuführende Aktion.

```
from turtle import *

t1=Turtle()
t2=Turtle()

t1.goto(0,100)
t1.forward(200)
```

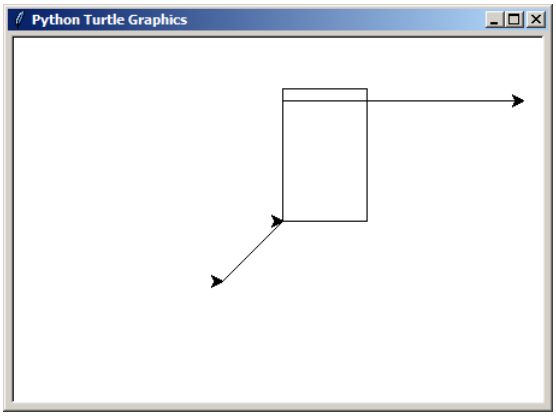
Zuerst lassen wir die Schildkröte t1 an eine andere Position wandern.
Danach bewegen wir sie ein Stück vorwärts. Die Aktionen, Funktionen usw. werden Methoden genannt. Es ergibt sich also die allgemeine Notierungs-Vorschrift **objekt.methode**. Man sieht sehr schön, dass eine Schildkröte am Koordinaten-Ursprung zurückbleibt.



Beide Schildkröten haben jetzt unterschiedliche Positionen. Diese Objekt-Eigenschaften werden unabhängig für jedes Objekt verwaltet und heißen Attribute. Jede Schildkröte hat zwar die gleichen Attribute, aber unterschiedliche gespeicherte Attribut-Werte.

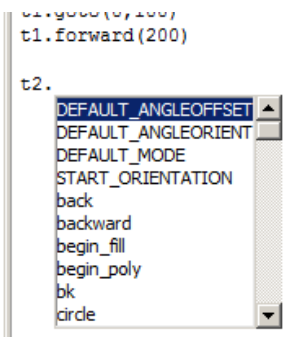
Die "zurückgebliebene" Schildkröte wollen wir nun auch testweise bewegen. Lassen wir sie z.B. ein Rechteck mit einer unserer getesteten Funktionen (→ 8.8.5. Funktionen) zeichnen.
Zum deutlichen Abgrenzen setzen wir sie auch noch auf einen anderen Startplatz.
Fraglich ist ja eigentlich schon, welche Schildkröte bewegt wird. Ist es die letzte angesprochene?
Die Überraschung ist perfekt, mit einem Mal haben wir drei Schildkröten.
Die Schildkröte t2 wurde zwar schräg nach unten bewegt, aber das Rechteck ging nicht von dieser Position und nicht von dieser Schildkröte aus.
Ganz offensichtlich fuktioniert zwar unsere Rechteck-Funktion, aber nicht Objekt-bezogen.
Um einen Objekt-Bezug hinzubekommen, müssen unsere Funktionen speziell definiert werden. Als ersten Parameter übergeben wir einen Objekt-Bezug (**rot** hervorgehoben). Der wird üblicherweise self genannt. Mit diesem self-Objekt (**blau** hervorgehoben) werden dann alle Aufgaben innerhalb der Funktion – besser jetzt Methode genannt – erledigt.
Nun befriedigt uns das Ergebnis auch hinsicht des erwarteten Ergebnisses.

```
from turtle import *

def rechteck(a,b):
    for _ in range(2):
        forward(a)
        left(90)
        forward(b)
        left(90)

t1=Turtle()
t2=Turtle()

t1.goto(0,100)
t1.forward(200)


t2.goto(-50,-50)
t2.rechteck(70,110)
```

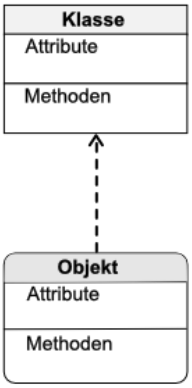



```
...

def rechteck(self,a,b):
    for _ in range(2):
        self.forward(a)
        self.left(90)
        self.forward(b)
        self.left(90)

...
```

Tippt man langsam genug – bzw. wartet man einen kleinen Augenblick nach der Eingabe des Punktes, dann zeigt uns IDLE aufeinmal ein kleines Auswahlwahl-Fensterchen an. Hier können wir die verfügbaren Methoden



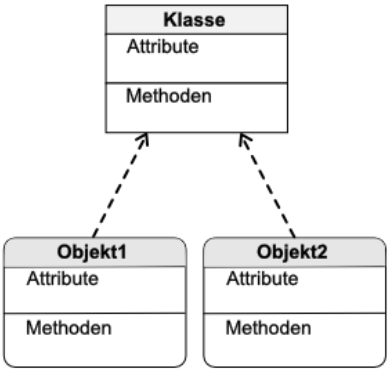
Informatisch betrachtet haben wir es bei der Objekt-orientierten Programmierung ja immer mit Klassen und Objekten zu tun. Mit dem Modul turtle bekommen wir eine Klasse Turtle zur Verfügung gestellt, die alle Attribute (Eigenschaften) und Methoden (Funktionen) enthält.



Von der Klasse Turtle können wir uns nun beliebig viele Schildkröten zum Zeichnen ableiten. Jede Schildkröte hat nun quasi einen Namen, da beim Ableiten eines Turtle-Objektes immer ein Variablen-Name angegeben werden musste. Über diesen Namen ist das Objekt im folgenden Programm ansprechbar. Der Varibalen- bzw. Objekt-Name steht immer vor dem Punkt in einer Objekt-Anweisung.

Jedes Objekt bekommt einen Satz eigener Attribute. Das sind z.B. die x- und y-Position oder auch die Farbe. Schließlich soll sich ja auch gjedes Objekt frei auf der Zeichenfläche bewegen können.

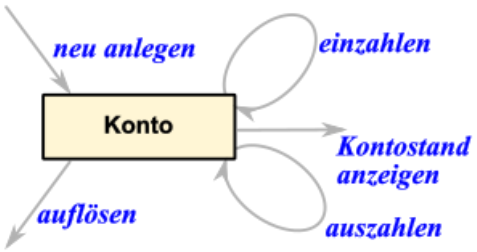
Die abgeleiteten turtle-Objekte haben Zugriff auf den gesamten Methoden-Satz. Jedes Turtle-Objekt kann sich unabhängig von den anderen vorwärts oder rückwärts bewegen oder auch drehen. Die für das jeweilige Objekte gewünschte Methode wird immer hinter dem Punkt der Objekt-Anweisung notiert.



8.11.x. Klassen – selbst erstellen

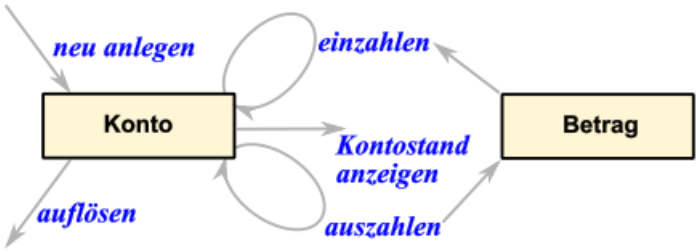
Beispiel Klasse "Konto":

Wenn man zuerst einmal sehr stark vereinfacht, dann haben wir nur wenige Aktionen, die wir mit einem Konto machen wollen / können wollen.
Mit den Pfeilen kennzeichnen wir Aktionen. Startet ein Pfeil bei einer Klasse, dann wird diese für die Aktion gebraucht.
Am Ende des Pfeil's steht die resultierende Klasse.



Je konkreter man wird, umso mehr Klassen werden in das Modell einbezogen.

Beim Umsetzen in ein Programmiersprache macht man dann oft einen Zwischenschritt und formuliert in einer Pseudosprache.



So könnte das Einzahlen etwa so formuliert werden:

(neuer)**Kontostand** ← **einzahlen**((aktueller)**Kontostand**; **Betrag**)

Die Aktion **einzahlen** benötigt zum Funktionieren einen (aktuellen) **Kontostand** und einen (einzuzahlenden) **Betrag**. Als Ergebnis erhalten wir einen (neuen) **Kontostand** zurück. Die Zuweisung (Ergibt-Anweisung) wird hier als gerichtete Handlung durch einen Pfeil gekennzeichnet.
Etwas Python-typischer ist die folgende Formulierung:

(neuer)**Kontostand** = **einzahlen**((aktueller)**Kontostand**; **Betrag**)

Die PASCALer würden es so notieren:

(neuer)**Kontostand** := **einzahlen**((aktueller)**Kontostand**; **Betrag**)

In jedem Fall meinen wir eine Zuweisung des rechten Teils zum linken.
Natürlich benutzen wir dann auch die in der Objekt-Orientierung festgelegten Begriffe Methoden und Attribute. Die Methoden sind eben die vorgestellten Aktionen. Attribute sind die einzelnen Eigenschaften, welche die Objekte / Instanzen dann besitzen. Zu den Objekten kommen wir dann später. Bis jetzt erstellen wir "nur" allgemeine Beschreibungen – eben die Klassen.

Aufgaben:

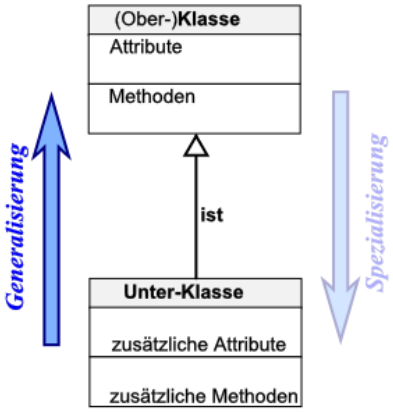
1. *Verbessern Sie das Klassen-Methoden-Diagramm so, dass ein Konto nur angelegt werden kann, wenn ein Besitzer vorhanden ist und und eine Einzahlung vorgenommen wird!*
2. *Erstellen Sie ein Klassen-Diagramm mit zugehörenden Methoden für die folgenden Klassen:*
   a) Briefmarken-Sammlung b) Adressbuch c) Bibliothek
3. *Geben Sie für die einzelnen Methoden an, ob sie etwas benötigen (z.B. einzahlen(Betrag)), etwas zurückliefern (z.B. Kontostand = anzeigenKontostand()) oder ev. auch beides!*

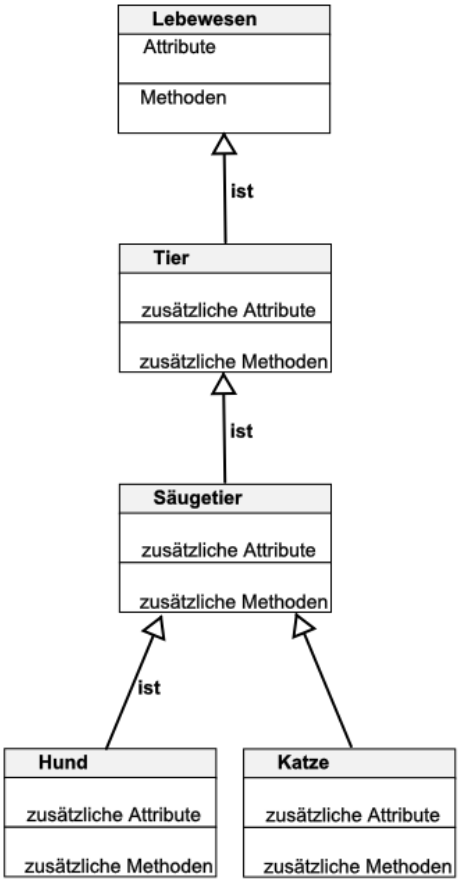
am Besten vor der praktischen Umsetzung in eine Programmiersprache jeweils das Klassen-Diagramme erstellen bzw. ein vorhandenes nutzen
gute Vorplanung spart Arbeit und schützt zumindestens teilweise vor bösen Überraschungen

geeignete Form UML-Diagramme

UML steht dabei für **Unified Modeling Language** (dt: einheitliche Modellierungs-Sprache).

stellen standardisiert Klassen mit ihren Attributen und Methoden sowie die Beziehungen zu anderen Klassen dar





nebenstehend für eine minimalistische Konto-Klasse

	Konto	*Klassenname*
-	Kontostand	*Attribute*
+C	neu	*Methoden*
+?	zeigeKontostand(Kontostand)	
+!	einzahlen(Betrag)	
+!	auszahlen(Betrag)	
+D	löschen	

die üblichen Alternativen sind die private oder globale Nutzung / Freigabe der Variable
private (interne) Variablen erhalten ein Minus-Zeichen (**-**)
mit einem Plus-Zeichen (**+**) kennzeichnet man globale (public) Variablen
C steht für **C**onstructor (Konstrukteur) und ist die Methode, mit der neue Objekte (Instanzen) erzeugt werden. In Python heißt diese **init()**.
Mit **D** wird der **D**estructor (Zerstörer) gekennzeichnet. Mit ihm werden vorhandene Objekte / Instanzen gelöscht (sauber entfernt).
Python verwendet nur selten Destructoren. Dies ist nur notwendig, wenn Attribut-Werte gerettet werden müssen. Das könnte z.B. ein Rest-Kontostand sein, der notwendigerweise beim Auflösen des Konto ausgezahlt oder einem anderen Konto zugeordnet werden muss. Die Benennung ist freigestellt, meist nutz man **del()**. In anderen Sprachen sind Destruktoren zwingend vorgeschrieben, um eine sauberes Objekt-Handling zu realisieren.
Steht vor der Methode ein Fragezeichen (**?**), dann gibt diese einen oder mehrere Werte zurück. Die Methode hat (an-)fragenden Charakter.
Mit einem Ausrufe-Zeichen (**!**) werden solche Methoden gekennzeichnet, die schreibend auf die (lokalen) Attribute wirken. Die Methode ist anweisend / befehlend.

verbesserte bzw. erweiterte Konto-Klasse

Verwendung sehr elementarer Methoden

z.B. wird **einzahlen()** auf das Einbringen eines Betrag's reduziert. Will man den Kontostand vorher oder hinterher wissen, dann muss man **zeigeKontostand()** benutzen.

	Konto	*Klassenname*
+	Kontenzähler	*Attribute*
-	Kontostand	
+	Dispo	
-	Inhaber	
-	Berechtige	
+C	neu	*Methoden*
+?	zeigeKontostand(Kontostand)	
+!	zeigeInhaber(Inhaber)	
+!	ändereInhaber(neuerInhaber)	
+?	berechtigt(Akteur)	
+!	einzahlen(Betrag)	
+!	auszahlen(Betrag)	
+D	löschen	

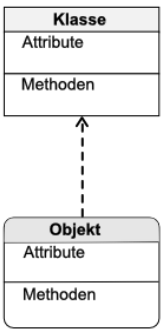
Aufgaben:

1. *Erstellen Sie ein UML-Klassen-Diagramm für die Klasse "Adressbuch"!*
2. *Überlegen Sie sich ein Klassen-Methoden- und ein UML-Diagramm für eine Klasse "DVD-Ausleihe"!*

für die gehobene Anspruchsebene:

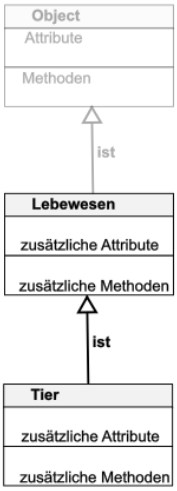
3. *Erstellen Sie Klassen-Methoden- und UML-Diagramm für einen Streaming-Dienst mit Einzel-Abrechnung (Kein Abo!)!*

Klasse-Objekt-Beziehung



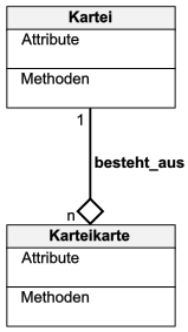
Aus einer Klasse können wir Objekte generieren. Die Informatiker sprechen auch von Instanzen (d(ies)er Klasse). Dies sind die ganz konkreten Dinge / Sachverhalte, die wir verarbeiten wollen. Also z.B. die Kuh "Else", der Ziegenhirt "Peter" oder das Auto "Speedi 3000". Sie werden nach dem Muster erstellt, was in der Klasse vordefiniert ist.
Wenn man einzelne Objekte (vielleicht als Beispiel) in ein UML-Diagramm bringen will, dann werden die Rechte dafür mit abgerundeten Ecken gezeichnet. Die Beziehung wird als einfacher Pfeil mit gestrichelter Linie dargestellt.
Die Attribute werden dann meist auch mit konkreten Werten geführt. Die Methoden werden ohne Veränderungen von der Klasse übernommen.

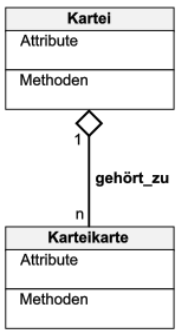
"ist"-Beziehung (Vererbung)



Bei den "ist"-Beziehungen handelt es sich um klassische Über- und Unter-Ordnungen von Klasse als Hierrarchie. Die untergeordnete Klasse **ist** eine Verfeinerung der oberen Klasse. Sie erbt von der übergeordneten Klasse viele Attribute und Methoden, kann diese aber in sich überschreiben oder erweitern.
In der Informatik gehören alle Klassen automatisch zur Klasse "Objekt". Sie erben von dieser Klasse bestimmte minimale Attribute und Methoden, die minimal notwendig sind. Das könnten z.B. eine Speicher-Adresse und die minimalste init()-Methode sein. Diese Methode würde dann vielleicht nur eine Speicher-Adresse festlegen.
Die Klasse "Objekt" darf nicht mit einem konkreten Objekt – besser einer Instanz – verwechselt werden. Die Klasse "Objekt" ist die allgemeine (minimalste) Ableitungs-Vorschrift für jedes spätere Objekt (- jede Instanz).

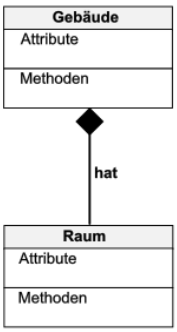
"besteht_aus"-Beziehung (Aggregation)





Hier beinhaltet eine Klasse eine oder mehrere andere Klassen, die für sich eigenständig sind und keine Verfeinerung darstellen. Die Instanzen der Unterklasse können auch weiter existieren, wenn das übergeordnete Objekt gelöscht wird. Diese Objekte müssen nicht zwangsläufig mit einem übergeordneten Objekt gemeinsam gelöscht werden.
Karteikarte ist eine Klasse mit eigenständigen Objekten.
Man kann sich später beliebig viele einzelne Karteikarten als Objekte vorstellen, ohne dass diese eine Kartei bilden.
Eine Kartei ist nur dann vorhanden, wenn mindestens ein Karteikarten-Objekt vorhanden ist.
Ein anderes klassisches Beispiel ist die Klasse "Auto", die wiederum aus vielen Bauteil-Klassen besteht. Sie könnte z.B. die Klassen "Motor", "Rad" und "Sitz" enthalten. Jede der später erstellten Instanzen – z.B. von "Motor" – kann unabhängig von einem konkreten "Auto" für sich existieren und auch für ganz andere Objekte (vielleicht ein Motor-Boot) verwendet werden.
In einigen UML-Diagrammen findet man auch die Umschreibung "gehört_zu". Hier wird dann die "Pfeil"-Richtung getauscht.

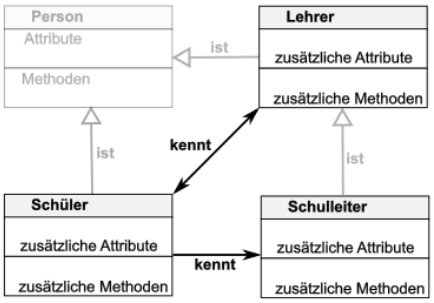
"hat"-Beziehung (Komposition)



Bei einer "hat"-Beziehung beinhaltet eine Klasse ebenfalls eine oder mehrere andere Klasse. Nur hier existieren diese Klassen nur im Zusammenhang mit der Oberklasse.
Ein klassisches Beispiel ist die Klasse "Raum" zur Klasse "Gebäude". Räume existieren nur im Zusammenhang mit dem Gebäude.
Werden später Instanzen von "Gebäude" gelöscht, dann existieren die darin angelegten Räume (Instanzen der Klasse "Raum") nicht mehr. Sie werden mit gelöscht. Eine Existenz ohne ein Gebäude ist nicht denkbar.

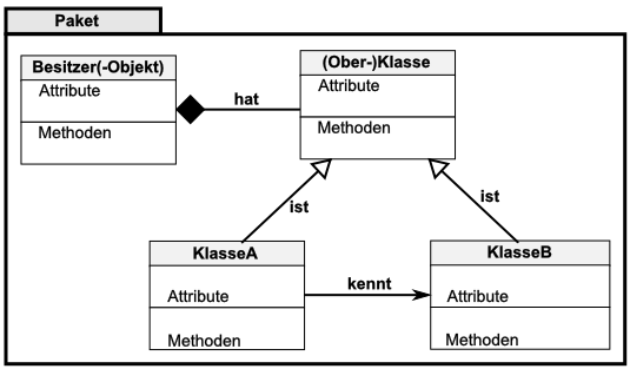
Im Zusammenhang von "hat"-Beziehungen spricht man bei der übergerdneten Klasse auch gerne von einer Besitzer-Klasse. Wenn der Besitzer nicht mehr existiert, dann existieren die zugeordnten Instanzen anderer Klassen auch nicht mehr.

"kennt"-Beziehung



Mit dieser Beziehung werden kommunikative Beziehungen charakterisiert. Zwischen den Objekten der Klassen sollen dann später Nachrichten ausgetauscht werden. "Kennt"-Beziehungen werden meist zwischen Klassen der gleichen oder benachbarten (direkt über- oder unter-geordneten) Klasse(n) aufgebaut.
Im nebenstehenden Beispiel sind "Schüler" und "Lehrer" Klassen auf der gleichen Hierrarchie-Ebene (nicht abhängig von der Darstellung!). Später müssen Lehrer und Schüler dann zu Schul-Klassen zusammengefasst werden, was dann über die "kennt"-Beziehung einfach realisierbar ist.

ein etwas größeres UML-Klassen-Diagramm könnte dann nebenstehenden Aufbau haben



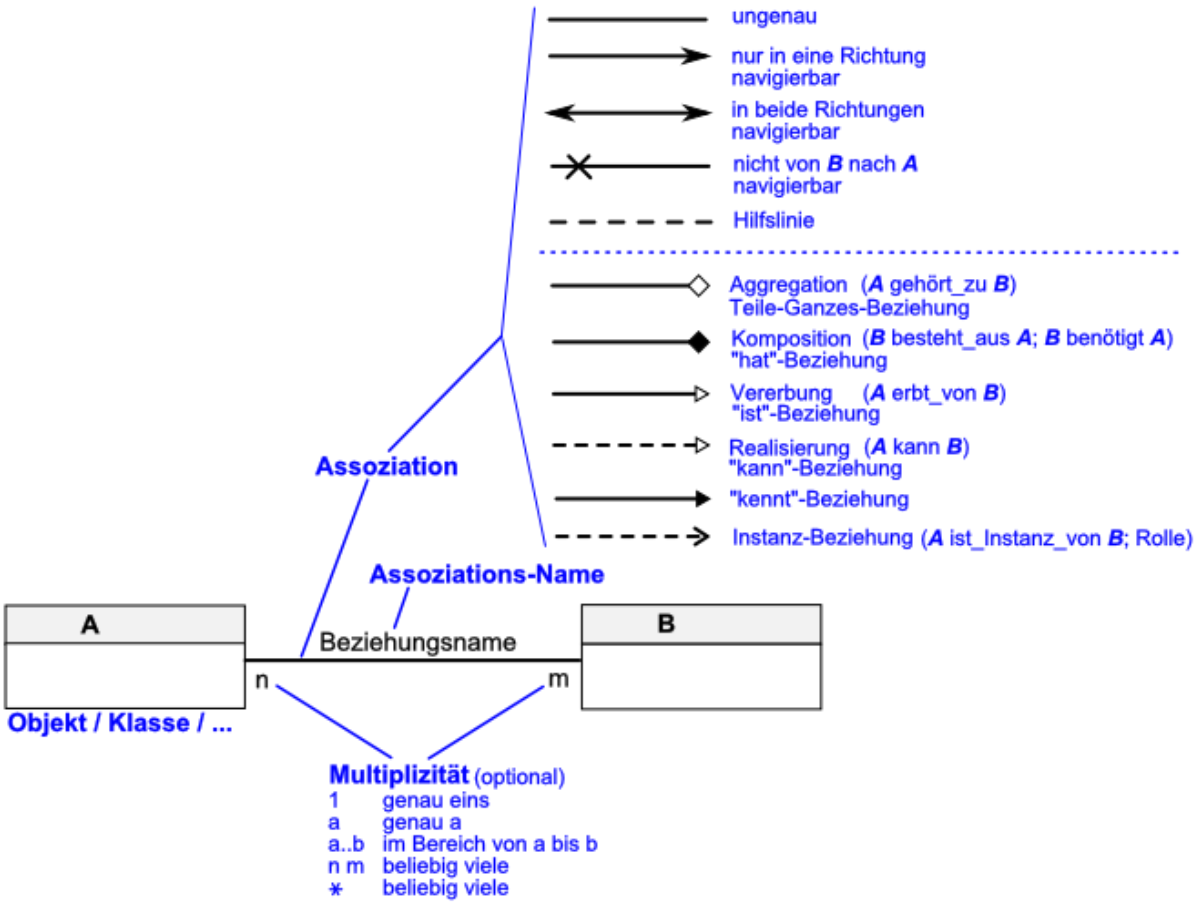
Aufgaben:

1. ***Erstellen Sie ein realistisches Klassen-Diagramm (UML) aus den folgenden Klassen und geben Sie auch immer ein passendes Objekt an! Im Klassen-Symbol nennen Sie immer mindestens ein Attribut und eine Methode (außer init() bzw. del())!***
   (LKW, Fahrrad, Fahrzeug, motorisiertesFahrzeug, PKW, Mercedes, nichtmotorisiertesFahrzeug)
2. ***Erstellen Sie ein Klassen-Diagramm für "Fahrrad"! Es reichen die wesentlichen Attribute und Methoden.***
3. ***Erstellen Sie ein realistisches Klassen-Diagramm (UML) aus den folgenden Klassen und geben Sie auch immer ein passendes Objekt an! Im Klassen-Symbol nennen Sie immer mindestens ein Attribut und eine Methode (zusätzlich zu init() und del())!***
   (Schüler, Schule, Klassenraum, Lehrer, Gebäude, Tisch, Schulleiter, Hausmeister, Inventar)

für die gehobene Anspruchsebene:

4. ***Ein Bauer will seine Wirtschaft vollständig digitalisieren. Erstellen Sie ein UML-Diagramm. das die von Ihnen als digital zu verwaltenden Klassen mit wichtigen (den wichtigsten!) Attributen und Methoden vorstellt!***

Übersicht / Legende zu UML-(Klassen-)Diagrammen:



8.11.x.1. Erstellen einer Klasse

Erzeugen einer Instanz / eines Objektes im Programm dann mit

instanzname = klassenname(initialattribute)

mit **pass** wird für die Phase der Programm-Entwicklung temporär die Notwendigkeit von inhaltlichen Quellcode ausgeschaltet
das gilt für Klassen und Methoden

Normalerweise sind in Python die Attribute und Methoden einer Klasse von außen sichtbar bzw. nutzbar. Zugriff immer über den Namen der Instanz und dem Punkt-getrennten entsprechenden Attribut bzw. dem Methodennamen.

Das ist aber in der Objekt-orientierten Programmierung nicht gewollt. Die Attribute sollen immer nur über geeignete Methoden verändert oder gelesen werden. Die Methoden werden meist als get(Attributname() und set(Attributname) definiert.
Wir sprechen auch vom get-set-Paar oder bei vielen von den Gettern und Settern.

Unsichtbare Klassen-Bestandteile werden mit **private** deklariert.
Gibt man direkt vor dem Namen einen Unterstrich an, dann ist der Klassenbestandteil als **protected** gekennzeichnet. Dieses gilt nur für Programmierer (als Konvention / Empfehlung), dem Interpreter ist dies egal. Für ihn sind die Attribute und Methoden immer (über die Instanz) sichtbar

Die Quellcode-Texte für ein spezielles Beispiel – hier eine Konto-Klasse sind grün umrahmt.

Konto-Beispiel (Schritt 1)

```
class Konto:
    pass
```

Die allgemeinen Quellcode's sind ohne diesen Rahmen und müssen immer für ein konkretes Beispiel angepasst werden.

```
class Klassenname():
      pass
```

8.11.x.1.1. der Konstruktor

```
class Klassenname():
    def __init__(self):
      pass
```

hat eine Klasse keinen Konstruktor, dann wird der Konstruktor der Oberklasse aufgerufen
man kann aber auch den Konstruktor der Oberklasse explizit aufrufen und ausgewählte Eigenschaften etc. verändern / überschreiben

```
class Klassenname(Objekt):
    def __init__(self):
        Objekt.__init__(self)
```

der Aufruf – und damit das Erzeugen eines (konkreten) Objektes erfolgt dann im Programm mit:

objektname = Klassenname()

Wenn man sich schon auf der Ebene des Konstruktors über die unbedingt zu definierenden (initialen) Variablen und ev. auch Werte schon klar ist, dann kann man diese gleich mit in die Vordefinition des Konstruktors einfließen lassen.
Grundlage dafür könnte z.B. ein gut ausgearbeitetes UML-Diagramm sein.

Konto-Beispiel (Schritt 2)

```
class Konto:
    def __init__(self,inhaber,betrag,autorisiert=["Banker"]):
        pass
```

Diese Vordefinition – immer gut am pass zu erkennen, muss dann später unbedingt mit Leben – sprich: Programm-Text – ausgefüllt werden.
Dazu müssen wir aber erst mal klären, wo diese Werte hin gehören.

8.11.x.2. Attribute einer Klasse

es wird zwischen **Klassen-Attribute**n und **Instanz-Attribute**n unterschieden
Klassen-Attribute werden für eine Klasse nur einmal angelegt und gelten für die Klasse allgemein bzw. für alle Instanzen gemeinsam
eine typische Verwendung ist das Zählen der Instanzen, die gerade von einer Klasse erzeugt / gehändelt werden
Klassenattribute folgen direkt im class-Block einschließlich der Zuweisung eines Initial-Wertes

Instanz-Attribute sind nur innerhalb der Instanz gültig,
ein Attribut der einen Instanz eines Klassenobjektes ist unabhängig vom gleichnamigen Attribut einer anderen Instanz der selben Klasse

public-Attribute sind von außen über die Punkt-Schreibung benutzbar

private-Attribute sind nur innerhalb einer Instanz benutzbar / sichtbar; ein Zugriff von Außen muss über darauf abgestimmte Methoden (z.B. GET- und SET-Funktionen) realisiert werden
in Python wird ein Attribut **private**, wenn man vor den Namen zwei Unterstriche schreibt

Attribute mit nur einem Unterstrich sind **protected**, sie sind praktisch public, aber Programmierer sollten nicht auf diese Zugreifen; die protected-Klassifizierung ist also nur eine Empfehlung, kein echtes Statement

ev. als Beispiel dummer und schlauer Melder aus "Python für Kids" S. 387 ff.

Schreibung der Variable	Status-Bezeichnung	Bedeutung / Eigenschaften
name	public	von außen sichtbar (lesbar und schreibar) innerhalb der Klasse lesbar und schreibbar
_name	protected	von außen sicht (lesbar und schreibar), ABER Zugriff durch Programmierer der Klasse nicht gewollt innerhalb der Klasse lesbar und schreibbar
__name	private	von außen nicht sichtbar (nicht lesbar und nicht schreibbar) innerhalb der Klasse lesbar und schreibbar

ev. Objekt-Hierarchie BlackBox mit verschiedenen inneren (nach außen) unsichtbaren Funktionen / Reaktionen
ähnlich Skalierungs-Objekt / -Klasse aus "Raspberry Pi programmieren" S.85 ff.

Konto-Beispiel (Schritt 2)

```
class Konto:
    kontenzaehler=0

    def __init__(self,inhaber,betrag,autorisiert=["Banker"]):
        self.__inhaber=inhaber
        self.__kontostand=betrag
        self.__autorisiert=autorisiert
        Konto.kontenzaehler+=1
        self.__kontonummer=Konto.kontenzaehler
```

8.11.x.3. Methoden einer Klasse

Methoden sind nichts anderes als Funktionen innerhalb einer Klasse. Sie sind auch nur innerhalb der Klasse sichtbar und nutzbar.
Will man eine Klassen-Methode nutzen, dann geht das immer nur über die erzeugte Instanz (das erstellte Objekt).
Ein Zugriff von außen kann verhindert werden, wenn die Methode als **privat** deklariert wurde. Eine private Methode kann nur innerhalb einer Klasse benutzt werden.
Alle Methoden müssen mindestens und damit als ersten Parameter eine Referenz auf sich selbst als aufrufendes Objekt enthalten.

def methode(self{, parameter})

def methode(self{, parameter=wert})

Konto-Beispiel (Schritt 3)

```
class Konto:
    kontenzaehler=0
    kontennummer=0

    def __init__(self,inhaber,betrag,autorisiert=["Banker"]):
        self.__inhaber=inhaber
        self.__kontostand=betrag
        self.__autorisiert=autorisiert+inhaber
        Konto.kontenzaehler+=1
        Konto.kontennummer+=1
        self.__kontonummer=Konto.kontennummer

    def abfragen(self):
        pass

    def einzahlen(self):
        pass

    def auszahlen(self):
        pass
```

Nun werden die einzelnen Methoden durch Quelltext untersetzt. Ev. sollten auch noch durch Fluß- oder UML-Diagramme die notwendigen Parameter geklärt werden. Einmal programmierte Funktionen sollten dann später auf der Ebene der Parameter-Listen nicht mehr geändert werden. Bei kleinen Projekten überschaut man die verschiedenen Stellen mit den zu ändernden Parameter-Listen in Methoden-Aufrufen noch, aber bei großen Projekten wird schnell mal eins übersehen.

```
…
    def abfragen(self):
        print("===> Kontostand: ",self.__kontostand
…
```

Hier ist quasi auch eine wesentliche Entscheidung über das Ausgabe-Prinzip gefallen. Wir geben in diesem Beispiel immer gleich in den Methoden aus. Praktisch kann das auch im

Haupt-Programm erledigt werden. Dann hätte die abfragen()-Methode z.B. so aussehen können:

```
…
    def abfragen(self):
        return self.__kontostand
…
```

Konto-Beispiel (Schritt 4)

```
…
    def einzahlen(self,betrag):
        if betrag > 0:
            self.__kontostand+=betrag
            print("===> Kontostand: ",self.__kontostand)
        else
            print("===> Fehler! (Kein gültiger Betrag!)")
…
```

Konsequenterweise sollten wir hier auch gleich unsere abfragen-Methode benutzen, statt die Ausgabe hier wieder zu organisieren.

```
…
    def einzahlen(self,betrag):
        if betrag > 0:
            self.__kontostand+=betrag
            self.abfragen
        else
            print("===> Fehler! (Kein gültiger Betrag!)")
…
```

```
…
    def auszahlen(self,betrag,initiator):
        if initiator in self.__autorisiert:
            if betrag <= self.__kontostand:
            self.__kontostand-=betrag
            print("===> AUSZAHLUNG: ",betrag)
        else
            print("===> Fehler! (Kein gültiger Betrag!)")
        print("===> Kontostand: ",self.__kontostand)
…
```

8.11.x.4. Speicher-Bereinigung

8.11.x.4.1. der Destruktor

def __del__()

in den Rumpf werden die Anweisungen geschrieben, die vor / beim Löschen des Objektes (der Instanz / Referenz) erledigt werden sollen / müssen

del(instanz**)**

Konto-Beispiel (Schritt 5)

```
…
    def __del__(self):
        if betrag > 0:
            self.auszahlen(self)
        else
            print("===> Fehler! (Kontenausgleich notwendig!)")
…
```

Aufgaben:

1. *Verbessern Sie das Konten-Beispiel so, dass mehrfache (gleiche) Fehler-Meldungen in eine eigene Methode ausgelagert werden!*
2. *Erweitern Sie das Konto-Programm so, dass alle Fehler-Meldungen über eine Methode ausgegeben werden!*
3. *Überlegen Sie sich, wie man alle Fehler-Meldungen in einer (neuen) Methode unterbringen könnte! Die Methode soll immer den gerade passenden Fehler ausgeben!*
4. *Regionaliseren Sie das Programm für den englischen Sprachraum!*
5. *Erstellen Sie eine Klasse "Auto" unter Beachtung der folgenden Vorgaben und geforderten Methoden! Testen Sie alle Methoden in einem kleinen Test-Programm! Nach einem erfolgreichen Test können einzelne Methoden aus dem Test-Programm auskommentiert werden – müssen aber wieder nutzbar gemacht werden können!*
   Ein Auto hat die Merkmale Kennzeichen, Verbrauch (gemeint pro 100 km), TachoStand, TankMaxVol (maximales Tank-Volumen) sowie TankAktVol (aktuelles Tank-Volumen).
   Beim Erstellen eines Auto's gehen wir davon aus, dass es ungefahren und unbetankt ist.
   Die Klassen-Definition soll alle (sinnvollen) Geter und Seter (Gib- und Setz-Methoden) enthalten.
   Als spezielle Möglichkeiten sollen einem späteren Hauptprogramm die folgenden Funktionen zur Verfügung stehen: StatusAnzeige() (als Nutzerfreundliche Anzeige aller Attribute in einer zweizeiligen Ausgabe), tanken(volumen) und fahren(kilometer).
   Für die Maximal-Bewertung werden auch Fehler-Meldungen in den Methoden erwartet, z.B. wenn versucht wird, zuviel Treibstoff einzufüllen usw. usf.! Eine erste Klassen-Definition kann auch noch mit z.B. negativen Tank-Volumen usw. arbeiten! Das Haupt-Programm liefert kontrollierte Werte für Eingaben / Parameter!
   Extra-Bewertung: die Tanken-Methode so gestalten, dass nicht gebrauchter Treibstoff an das Hauptprogramm zurückgegeben wird! Die Anzeige soll über das Haupt- bzw. Test-Programm erfolgen!

für die gehobene Anspruchsebene:

6. *Regionaliseren Sie Ihr Konto-Programm für den französischen oder spanischen oder … Sprachraum (mit lateinischen Buchstaben)! (Lassen Sie sich ev. von einem anderen Kursteilnehmer die Texte und / oder Stichworte) geben!*

Projekt-Aufgaben:

1. ***Erstellen Sie ein Menü-gesteuertes Programm nach unten aufgezeigten Beispiel mit einer eigenen / geänderten Klassen-Konstruktion!***
2. ***Gebraucht wird eine praktisch nutzbare Klassen-Definition (einschließlich Attributen, Methoden etc.) für ein Adressbuch! Im Adressbuch sollen Name, Vorname, Geburtsdatum, eMail und Telefonnummer gespeichert werden. Weiterhin soll die Klasse die Anzahl der Kontakte ausgeben können und eine Anzeige machen, wenn de Kontakt heute Geburtstag hat!***
3. ***Erstellen Sie ein kleines Testprogramm, um die Klasse mit ausgedachten Daten auszuprobieren!***

für die gehobene Anspruchsebene:

4. ***Erweitern Sie das Adressbuch um eine Vorwarnung, wenn jemand morgen bzw. übermorgen Geburtstag hat!***
5. ***Erstellen Sie ein Menü-gesteuertes Haupt-Programm für die Adressbuch-Klasse!***

bank.py

```
class Bankkonto:
   """Einfache Bankkonto-Klasse"""

   def __init__(self,startbetrag):
      """Konstruktor: erzeugt Bankkonto"""
      self.kontostand = startbetrag

   def einzahlung(self, betrag):
      self.kontostand = self.kontostand + betrag

   def auszahlung(self, betrag):
      self.kontostand = self.kontostand - betrag

   def anzeigen(self):
      print self.kontostand

# ausprobieren
konto1 = Bankkonto(100)
konto1.anzeigen()
konto1.einzahlung(200)
konto1.anzeigen()
konto1.auszahlung(125)
konto1.anzeigen()
print konto1.__doc__
```

Q: http://www.wspiegel.de/pykurs/kurs_index.htm

Beispiel 1

```
"""
direkter Dialog zwischen zwei Instanzen unterschiedlicher Klassen über deren Methoden
Nur client ist hier wirklich aktiv, server reagiert nur
Die melde-funktion von server dient hier nur zur Kontrolle
"""

class server:
  def __init__(self, wert=0):
    self.data = wert
def neu(self, wert):
  self.data = wert
def melde(self):
  return self.data

"""
server macht hier das Einfachste vom Einfachen
er merkt sich nur eine Zahl
"""

class client:
  def __init__(self, wert):
    self.wert = wert
  def setze(self):
    s.neu(self.wert)
```

```
  def frage(self):
    print s.melde()

s = server()
print s.melde()    # server in Grundstellung (wert = default)
c1 = client(123)

# nun sind zwei Instanzen geschaffen, die miteinander reden können

c1.setze()
# jetzt hat c1 wert von server neu gesetzt

s.melde()
```

Beispiel 2

```
"""
direkter Dialog zwischen Instanzen unterschiedlicher Klassen über deren Methoden
Auch hier ist nur client wirklich aktiv, server reagiert nur
Die melde-funktion von server dient nur zur Kontrolle
"""

class server:
  def __init__(self):
    self.data = []
  def hinzu(self, wert):
    self.data.append(wert)
  def melde(self):
    return self.data

"""
server macht hier schon mehr, als im Beispiel 1, er merkt sich die Zahlenwerte aller ange-
schlossenen client
aus Vereinfachungsgründen können diese Zahlenwerte durch client nach der Erstmeldung
nicht nochmal geändert werden
"""

class client:
  def __init__(self, wert):
    self.wert = wert
    s.hinzu(wert)
  def melde(self):
    return self.wert

s = server()
print "Wertesammlung in server vorher: ", s.melde()    # server - Liste noch leer


clients = []
for i in (6, 5, 100, 19, 27):
  c = client(i)
```

```
  clients.append(c)

print "Wertesammlung in server nachher:", s.melde()

"""
wo sind eigentlich die 5 clients geblieben, die wir in der for - Schleife erzeugt haben? Sie
liegen als Feld von Adresszeigern in der Liste clients und könnten dort jederzeit weiterver-
wendet werden. Das ginge dann so:
"""

for i in range(0, 5):
  print str(i+1) + ". client hat den Wert:", clients[i].melde()

"""
Und eben dies könnte doch auch die server - Klasse verwalten!
"""
```

Q: http://www.way2python.de/

Beispiel 3

```
"""
direkter Dialog zwischen Instanzen unterschiedlicher Klassen über deren Methoden
Hier werden sowohl client als auch server aktiv
die clients hinterlegen im Server ihre Adresse und werden durch server auf ihren Zustand
befragt.
"""

class server:
  def __init__(self):
    self.data = []
  def hinzu(self, adr):
    self.data.append(adr)
  def abfrage(self):
    werte = []
    for i in self.data:
      werte.append(i.melde())
    return werte

"""
server macht hier noch mehr, als im Beispiel 2, er merkt sich die Adressen aller angeschlos-
senen client
Zur Abfrage holt er sich die aktuellen Werte der clients, wenn es soweit ist.
Hier können die Zahlenwerte der clients nach der Erstmeldung jederzeit geändert werden
"""

class client:
  def __init__(self, wert):
    self.wert = wert
    s.hinzu(self)
    # jetzt ist es passiert, hier geht die client-adr in die Liste und nicht der Wert
  def melde(self):
```

```
    return self.wert
  def setzneu(self, wert):
    self.wert = wert

s = server()
print "Wertesammlung in server vorher: ", s.abfrage() # server - Liste noch leer

clients = []
for i in (6, 5, 100, 19, 27):
  c = client(i)
  clients.append(c)

print "Wertesammlung in server nachher:", s.abfrage()

clients[3].setzneu(4000)
print "Wertesammlung in server, nach einer Änderung:", s.abfrage()
```

Q: http://www.way2python.de/

```
# -----------------------------------------------
# Dateiname: konto.py
# Modul mit Implementierung der Klasse Konto. Sie wird von
# der Klasse Geld abgeleitet und modelliert ein Bankkonto
#
# Objektorientierte Programmierung mit Python
# Kap. 10
# Michael Weigend 20.9.2009
#-----------------------------------------------
import time
from geld2 import Geld
class Konto(Geld):
    """ Spezialisierung der Klasse Geld zur Verwaltung eines Kontos

        Öffentliche Attribute:
           geerbt: waehrung, betrag, wechselkurs

        Öffentliche Methoden und Überladungen:
           geerbt: __add__(), __cmp__(), getEuro()
           ueberschrieben: __str__()
           Erweiterungen:
             einzahlen(), auszahlen(), druckeKontoauszug()
    """
    def __init__(self, waehrung, inhaber):
        Geld.__init__(self,waehrung, 0)              #1
        self.__inhaber = inhaber                     #2
        self.__kontoauszug = [str(self)]             #3

    def einzahlen(self,waehrung, betrag):            #4
        einzahlung = Geld(waehrung,betrag)
        self.betrag =(self+einzahlung).betrag        #5
        eintrag = time.asctime()+ ' ' + str(einzahlung)+ \
                ' neuer Kontostand: ' + self.waehrung + \
                format (self.betrag, '.2f')
        self.__kontoauszug += [eintrag]              #6

    def auszahlen(self, waehrung, betrag):
        self.einzahlen(waehrung,-betrag)

    def druckeKontoauszug(self):                     #7
        for i in self.__kontoauszug:
            print(i)
```

```
        self.__kontoauszug = [str(self)]

    def __str__(self):                                  #8
        return 'Konto von ' + self.__inhaber + \
                ':\nKontostand am ' + \
                time.asctime()+ ': '+ self.waehrung + ' ' +\
                format (self.betrag, '.2f')
```

```
# ------------------------------------------------
# Dateiname: geld2.py
# Klasse Geld mit Überladung der Operatoren +, <, >, ==
# Objektorientierte Programmierung mit Python
# Kap. 10
# Michael Weigend 20.9.2009
#-------------------------------------------------
class Geld(object):
    wechselkurs={'USD':0.84998,
                 'GBP':1.39480,
                 'EUR':1.0,
                 'JPY':0.007168}

    def _berechneEuro(self):                            #1
        return self.betrag*self.wechselkurs[self.waehrung]

    def __init__(self, waehrung, betrag):
        self.waehrung=waehrung
        self.betrag=float(betrag)

    def __add__ (self, geld):                           #2
        a = self._berechneEuro()
        b = geld._berechneEuro()
        faktor=1.0/self.wechselkurs[self.waehrung]
        summe = Geld (self.waehrung, (a+b)*faktor )
        return summe

    def __lt__(self, other):
        a = self.getEuro ()
        b = other.getEuro ()
        return a < b

    def __le__(self, other):
        a = self.getEuro ()
        b = other.getEuro ()
        return a <= b

    def __eq__(self, other):
        a = self.getEuro ()
        b = other.getEuro ()
        return a == b

    def __str__(self):
        return self.waehrung + ' ' + format(self.betrag, '.2f')
```

8.11.x.6. eine "Auto"-Klasse

```
class Auto:

# Konstruktor
    def __init__(self, ):
        self.Name=name
        self.Kennzeichen=kennzeichen
        self.TankVolumen=tankvolumen
        self.Verbrauch=
…
```

8.11.x.6.1. Erweiterung der "Auto"-Klasse um LKW's

8.11.x.7. eine "Personen"-Klasse

Für eine Personen-Datenbank wird eine Klasse "Person" gebraucht. Diese soll dann später in einer Verwaltung für Familien-Betreuung genutzt werden.
Wir gehen dieses Mal in etwas größeren Schritten vor und erläutern die einzelnen Schritte nicht mehr zu ausführlich.
Zuerst erstellen wir uns die klassische Struktur aus Klassen-Defintion, Konstruktor und einem einfachen Test-Programm. Das Test-Programm wir immer nur schnell erweitert, um die neuen Attribute und Methoden unserer neuen Klasse gleich testen zu können. Sinn muss dieser Test-Teil nicht unbedingt ergeben.

```
class Person:

# Konstruktor
    def __init__(self, name, vorname):
        self.Name=name
        self.Vorname=vorname


#Test-MAIN
P1=Person("Muster","Oleg")
P2=Person("Schulz","Franka")
P3=Person("Bauer","Kim")
```

Beim Laufen-Lassen unseres kleinen Programm erhalten wir keine Fehler-Meldung, aber auch keine Anzeige.
Unsere nächste Aufgabe soll also eine Anzeige der gespeichert Personen-Daten sein. Die Klassen-Definition wir entsprechend um die Methode zeigePersonDaten() erweitert. Den Test der Methode hängen wir dann auch gleich im Test-Teil an.

```
…
    def zeigePersonDaten(self):
        print("Name: ",self.Name,"   Vorname: ",self.Vorname)


#Test-MAIN
…
P3.zeigePersonDaten()
P1.zeigePersonDaten()
```

Jetzt erzeugen wir bei Ausprobieren auch tatsächlich Ausgaben auf dem Bildschirm.

8.11.x.7.1. Erweiterung der "Personen"-Klasse auf eine Familie

8.11.x.7. eine "Nachrichten"-Klasse

Klasse Nachrichten mit den Attributen Sender, Empfänger und Text (ev. ++)
Methoden senden, antworten und weiterleiten

8.11.x.y. eine Graphik-Beispiel-Klasse

→ http://www.b.shuttle.de/b/humboldt-os/python/kapitel4/index.html

```
# Die Grafik-Klassen in   graph.py

#! /usr/bin/python

import Tkinter
from Tkconstants import *
import Canvas

class Image:
   """
   /* Bildklasse
   """
   def __init__(self,Name):
      """
      /* Name: string : Dateiname des Bildes
      """
      self.Bild=Tkinter.PhotoImage(file=Name)
      self.Breite=self.Bild.width()
      self.Hoehe=self.Bild.height()

   def get_Bild(self):
      """
      /* liefert das Bildobjekt für 'image' in Canvas.ImageItem
      /* (Darstellung des Bildes auf einer Zeichenfläche)
      """
      return self.Bild

   def get_Breite(self):
      """
      /* liefert die Bildbreite in Pixeln
      """
      return self.Breite

   def get_Hoehe(self):
      """
      /* liefert die Bildhoehe in Pixeln
      """
      return self.Hoehe

class TColor:
   """
   /* erste primitive Version mit nur wenigen Farben
   /* Die Farben können über die deutschen Namen oder über
   /* Zahlen abgerufen werden
   /* Hinweis:   "#fff" entspricht "weiss",
   /*      statt "#xxx" kann auch "red", "green"  usw. benutzt werden
   /* andere Lösungen mit true-colors sind denkbar
   """
   def __init__(self):
      """
      /* transparent, schwarz, blau, gruen, tuerkis, rot, gelb, grau, weiss
      """
      self.Fnamen = { \
                     0:"transparent", \
                     1:"schwarz" , \
                     2:"blau"    , \
                     3:"gruen"   , \
                     4:"tuerkis" , \
                     5:"rot"     , \
                     6:"gelb"    , \
                     7:"grau"    , \
                     8:"weiss"   \
                  }

      self.Farbe={ \
                     "transparent":"", \
                     "schwarz"    :"#000", \
                     "blau"       :"#00f", \
                     "gruen"      :"#0f0", \
                     "tuerkis"    :"#0ee", \
                     "rot"        :"#f00", \
                     "gelb"       :"#ff0", \
                     "grau"       :"#ccc", \
```

```
                    "weiss"      :"#fff" \
                }

    def getColor(self,nr):
        """
        /* nr : int : 0 .. 8  für die oben angegebenen Farben
        /* liefert die Farbdarstellung für X
        """
        return self.Farbe[self.getFarbnamen(nr)]

    def getFarbnamen(self,nr):
        """
        /* nr : int : 0 .. 8  für die oben angegebenen Farben
        /* liefert den (deutschen) Bezeichner der Farbnummer (s.o.)
        """
        return self.Fnamen[nr]

    def getFarbe(self,wort):
        """
        /* wort : string :  ein Element aus den oben angegebenen Farben
        /* liefert die Farbdarstellung fuer X
        """
        return self.Farbe[wort]

class TFigur:
    """
    /* interne Hinweise:
    /* ZF ist Referenz auf Zeichenfläche, wird später gesetzt
    /* grafObj ist das aktuelle Grafikobjekt
    """
    def __init__(self):
        """
        /* Alle Grafik-Klassen erben von TFigur. TFigur wird beschrieben durch
        /* folgende Attribute:
        /* X1,Y1 (linke obere Ecke)
        /* X2,Y2 (rechte untere Ecke)
        /* Farbe
        /* Fuellfarbe
        """
        self.X1=20
        self.Y1=20
        self.X2=100
        self.Y2=100
        self.Farben=TColor()
        self.Farbe=self.Farben.getColor(0)
        self.Fuellfarbe=self.Farben.getColor(0)

    def setPos(self,ax1,ay1,ax2,ay2):
        """
        /* ax1,ay1 : int :(linke obere Ecke)
        /* ax2,ay2 : int :(rechte untere Ecke)
        """
        self.X1=ax1
        self.Y1=ay1
        self.X2=ax2
        self.Y2=ay2

    def getXPos(self):
        """
        /* liefert x-Wert der Position der linken oberen Ecke
        """
        return self.X1

    def getYPos(self):
        """
        /* liefert y-Wert der Position der linken oberen Ecke
        """
        return self.Y1

    def setFarbe(self,F):
        """
        /* F : string : deutscher Bezeichner (s.o.)
        """
        self.Farbe=self.Farben.getFarbe(F)

    def getFarbe(self):
        """
        /* gibt akt. Farbe zurück : string : Farbrepr. für X
        """
        return self.Farbe
```

```
   def setFuellfarbe(self,F):
      """
      /* F : string : deutscher Bezeichner (s.o.)
      """
      self.Fuellfarbe=self.Farben.getFarbe(F)

   def getFuellfarbe(self):
      """
      /* gibt akt. Füllfarbe zurück : string : Farbrepr. für X
      """
      return self.Fuellfarbe

   def pos_versetzen_um(self,dx,dy):
      """
      /* versetzt die Position des heweiligen Grafikobjektes um dx und dy
      """
      self.X1=self.X1+dx
      self.X2=self.X2+dx
      self.Y1=self.Y1+dy
      self.Y2=self.Y2+dy

   def zeigen(self):
      """
      /* zeigt das Grafikobjekt auf dem Schirm an
      """
      pass

   def loeschen(self):
      """
      /* löscht das Grafikobjekt auf dem Schirm
      """
      self.grafObj.move(1000,1000)

   def entfernen(self):
      """
      /* entfernt das Grafikobjekt aus dem Speicher
      """
      self.grafObj.delete()


class TLinie(TFigur):
   """
   /* Klasse Linie
   """
   def __init__(self):
      TFigur.__init__(self)
      x = self.getFarbe()
      self.grafObj=Canvas.Line(TFigur.ZF,(self.X1, self.Y1),(self.X2, self.Y2)\
                                                          , {"fill": x})

   def zeigen(self):
      self.grafObj.config(fill=self.getFarbe())
      self.grafObj.coords(((self.X1,self.Y1),(self.X2,self.Y2)))

class TEllipse(TFigur):
   """
   /* Klasse Ellipse
   """
   def __init__(self):
      TFigur.__init__(self)
      x = self.getFarbe()
      y = self.getFuellfarbe()
      self.grafObj=Canvas.Oval(TFigur.ZF, (self.X1, self.Y1), \
                              (self.X2, self.Y2), {"outline": x, "fill": y})

   def zeigen(self):
      self.grafObj.config(fill=self.getFuellfarbe(),outline=self.getFarbe())
      self.grafObj.coords(((self.X1,self.Y1),(self.X2,self.Y2)))

class TKreis(TFigur):
   """
   /* Klasse Kreis
   """
   def __init__(self):
      """
      /* zus. Attribute sind hier: Radus, x-Mittelpunkt, y-Mittelpunkt
      """
      TFigur.__init__(self)
      self.R=0
```

```
      self.Mx=0
      self.My=0
      x = self.getFarbe()
      y = self.getFuellfarbe()
      self.grafObj=Canvas.Oval(TFigur.ZF, (self.X1, self.Y1), \
                               (self.X2, self.Y2), {"outline": x, "fill": y})

   def __berechne_Standard(self):
      self.X1=self.Mx-self.R
      self.X2=self.Mx+self.R
      self.Y1=self.My-self.R
      self.Y2=self.My+self.R

   def setRadius(self,r):
      """
      /* r : int : Radius
      """
      self.R=r
      self.__berechne_Standard()

   def getRadius(self):
      """
      /* liefert aktuelle Radiuslänge
      """
      return self.R

   def setMPos(self,ax,ay):
      """
      /* ax, ay : int
      /* setzt Mittelpunktskoordinaten
      """
      self.Mx=ax
      self.My=ay
      self.__berechne_Standard()

   def zeigen(self):
      self.grafObj.config(fill=self.getFuellfarbe(),outline=self.getFarbe())
      self.grafObj.coords(((self.X1,self.Y1),(self.X2,self.Y2)))


class TRechteck(TFigur):
   """
   /* Klasse Rechteck
   """
   def __init__(self):
      TFigur.__init__(self)
      x = self.getFarbe()
      y = self.getFuellfarbe()
      self.grafObj=Canvas.Rectangle(TFigur.ZF, (self.X1, self.Y1), \
                               (self.X2, self.Y2), {"outline": x, "fill": y})

   def zeigen(self):
      self.grafObj.config(fill=self.getFuellfarbe(),outline=self.getFarbe())
      self.grafObj.coords(((self.X1,self.Y1),(self.X2,self.Y2)))


class TText(TFigur):
   """
   /* Klasse Text zur Beschriftung der Zeichenfläche
   """
   def __init__(self):
      """
      /* Attribute sind
      /* Text : string
      /* Schriftart : String  (X-Fonts-Bezeichner)
      /* Zeichen-Hoehe : int : default = 10
      """
      TFigur.__init__(self)
      self.Text=""
      self.Schriftart="*"
      self.Hoehe=10

   def setPos(self,ax,ay):
      """
      /* ax, ay : int : Position des ersten Zeichens
      """
      self.X1=ax
      self.Y1=ay

   def setText(self,Text):
```

```
      """
      /* Text : string : auszugebender Text
      """
      self.Text=Text

   def setFont(self,Art="*",Grad=10):
      """
      /* Art : string : Font-Name
      /* Grad : int : Zeichengröße
      """
      self.Schriftart=Art
      self.Hoehe=Grad

   def zeigen(self):
      self.grafObj=Canvas.CanvasText(TFigur.ZF, self.X1, self.Y1, \
             anchor="w", fill=self.Farbe, font=(self.Schriftart, self.Hoehe))
      self.grafObj.insert(0,self.Text)


class TZeichenblatt:
   """
   /* Zeichenblatt entspricht Canvas. Mit Init wird ein Bild unterlegt
   /* Zeichenblatt vom Typ TZeichenplatt wird erzeugt und steht zur
   /* Verfügung.
   """
   def __init__(self):
      pass

   def Init(self,Name):
      """
      /* Init hinterlegt das Bild
      /* Name : string : Dateiname (gif)
      """
      self.oWindow=Tkinter.Tk()
      self.oWindow.title("Zeichenfläche - nach S. Spolwig    -----   Kokavecz")
      self.oBild=Image(Name)
      self.X1=0
      self.Y1=0
      self.X2=self.oBild.get_Breite()
      self.Y2=self.oBild.get_Hoehe()
      Geometrie=str(self.X2)+"x"+str(self.Y2)+"+0+0"
      self.oWindow.geometry(Geometrie)
      self.oEbene=Tkinter.Canvas(self.oWindow,relief=SUNKEN, bd=5, \
                                          width=self.X2, height=self.Y2)
      bild=Canvas.ImageItem(self.oEbene,(0,0),anchor="nw", \
                                             image=self.oBild.get_Bild())
      self.oEbene.pack()
      TFigur.ZF=self.oEbene


   def get_Breite(self):
      """
      /* liefert die Bildbreite in Pixeln
      """
      return self.oBild.get_Breite()

   def get_Hoehe(self):
      """
      /* liefert die Bildhoehe in Pixeln
      """
      return self.oBild.get_Hoehe()


   def refresh(self):
      TFigur.ZF.update()

# sollte oZeichenblatt oder mein_Zeichenblatt heißen:
Zeichenblatt = TZeichenblatt()
```

Klassen-Dokumentation → http://www.b.shuttle.de/b/humboldt-os/python/kapitel4/grafik.py.html

8.11.x.2. Polymophismus und Vererbung

class erbendeKlasse(vererbendeKlassenListe):

z.B. an erweiterter Konten-Klasse jetzt auch mit Zinsen und Schulden

Aufgaben:

1. *Erweitern Sie das Konten-Programm so, dass für jede Minute der Zins für einen Monat angesetzt wird!*

alle Attribute und Methoden der in der vererbenden Klassenliste aufgeführten Klassen sind nun auch in der erbenden Klasse verfügbar (diese können hier aber auch überschrieben werden!)
die originalen Methoden werden über **vererbendeKlasse.Methode** und die überschriebenen über **erbendeKlasse.Methode**
Aufrufe ohne Klassen-Angabe verbleiben erst einmal in der aktuellen Klasse

Beispiel: Bücher-Klasse

```
class Buch:
    BuecherZahl=0

    def __init__(self,titel,autor,verlag,isbn,preis):
        self.Titel=titel
        self.Autor=autor
        self.Verlag=verlag
        self.ISBN=isbn
        self.Preis=preis
        Buch.BuecherZahl+=1

    def zeigeBuchInfo(self):
        print("Buch-Info:")
        print("Autor: ",self.Autor,"   Titel: ",self.Titel)
        print("Verlag: ",self.Verlag,"   ISBN: ",self.ISBN)

    def pruefeISBN(self):
        pass

    def __del__(self):
        Buch.BuecherZahl-=1

# Main
buch1=Buch("Das Leben der Z","Silp","Universal","1234567890X",24)
buch1.zeigeBuchInfo()
print("Preis= ",buch1.Preis, "Euro")
print("--> akt. Buchbestand: ",Buch.BuecherZahl)
print()
print()
print("Entfernen Buch1")
del(buch1)
print("--> akt. Buchbestand: ",Buch.BuecherZahl)
print("Ende")
```

```
>>>
Buch-Info:
Autor:  Silp    Titel:  Das Leben der Z
Verlag:  Universal    ISBN:  1234567890X
Preis=  24 Euro
--> akt. Buchbestand:  1


Entfernen Buch1
--> akt. Buchbestand:  0
Ende
>>>
```

nun braucht man z.B. für Fachbücher neben den üblichen Angaben vielleicht auch noch Informationen zu Fachgebieten und Themen oder Stich
Natürlich möchten – wir faulen Programmierer – nicht wieder alles neu programmieren. Wir haben ja schon eine super programmierte und getestete Klasse für normale Bücher.
Auf der Basis dieser Bücher-Klasse erstellen wir nun die Fachbuch-Klasse.

```
…
       Buch.BuecherZahl-=1
```

```
class FachBuch(Buch):
    BuecherZahl=0

    def
__init__(self,titel,autor,verlag,isbn,preis,fachbereich,stichwort):
        # Buch.__init__(self,titel,autor,verlag,isbn,preis)
        super().__init__(titel,autor,verlag,isbn,preis)
        self.Fachbereich=fachbereich
        self.Stichwort=stichwort
        FachBuch.BuecherZahl+=1

    def zeigeBuchInfo(self):
        print("Fach-",end='')
        # Buch.zeigeBuchInfo(self)
        super().zeigeBuchInfo()
        print("Fachbereich: ",self.Fachbereich,"   Stichwort:
",self.Stichwort)

    def __del__(self):
        FachBuch.BuecherZahl-=1
        Buch.BuecherZahl-=1
```

```
# Main
buch1=Buch("Das Leben der Z","Silp","Universal","1234567890X",24)
buch1.zeigeBuchInfo()
print("Preis= ",buch1.Preis, "Euro")
print("--> akt. Buchbestand: ",Buch.BuecherZahl,"   davon: ",
      FachBuch.BuecherZahl," Fachbücher")
print()
print()
buch2=FachBuch("LB Informatik","Meier","Fachbuchverlag",
           "1234567890X",30,"Progammierung","Python")
buch2.zeigeBuchInfo()
print("Preis= ",buch2.Preis, "Euro")
print("--> akt. Buchbestand: ",Buch.BuecherZahl,"   davon:
",FachBuch.BuecherZahl," Fachbücher")
print()
print()
print("Entfernen Buch2 (Fachbuch)")
del(buch2)
print("--> akt. Buchbestand: ",Buch.BuecherZahl,"   davon:
",FachBuch.BuecherZahl," Fachbücher")
print()
print("Entfernen Buch1")
…
```

```
>>>
Buch-Info:
Autor:  Silp    Titel:  Das Leben der Z
Verlag.  Universal    ISBN:  1234567890X
Preis=  24 Euro
--> akt. Buchbestand:  1    davon:  0  Fachbücher


Fach-Buch-Info:
Autor:  Meier    Titel:  LB Informatik
Verlag.  Fachbuchverlag    ISBN:  1234567890X
Fachbereich:  Progammierung    Stichwort:  Python
Preis=  30 Euro
--> akt. Buchbestand:  2    davon:  1  Fachbücher


Entfernen Buch2 (Fachbuch)
--> akt. Buchbestand:  1    davon:  0  Fachbücher

Entfernen Buch1
--> akt. Buchbestand:  0    davon:  0  Fachbücher
Ende
>>>
```

Beim genauen Betrachten des Quelltextes kann man einige Spezialitäten erkennen. Zum Ersten kann man innerhalb jeder Objekt-Ebene gleich namige Attribute / Variablen-Namen nutzen (hier: BuecherZahl). Sie beziehen sich immer auf die jeweilige Ebene, die als Objektnamens-Teil (vor dem Punkt) mit angegeben werden muss.
Dann können wir mit dem allgemeinen Namen **super** für die übergeordnete Klasse zurückgreifen. Das ist z.B. dann praktisch, wenn sich solche Namen öfter ändern oder der Quelltext mehrfach genutzt werden soll. **Super** ist somit strukturell dem **self** äquivalent.
In Python lässt sich auch über die Methoden hinweg z.B. eine spezielle Ausgabe realisieren. Im Fall eines Fachbuches wird nur das Wörtchen "Fach-" zur Anzeige (zeigeBuchInfo()) gebracht und dann direkt die Anzeige-Methode von Buch aufgerufen Die Anzeige-Methode von Fachbuch ergänzt dann noch die speziellen Attribute von Fachbuch.
Interessant ist auch, dass die Methoden den Objekten entsprechend ihrer Klasse zugeordnet werden. In beiden Bücherklassen gibt es die gleichlautende Methode zeigeBuchInfo(). Beim Aufruf von einem Fachbuch aus wird zuerst die Fachbuch-Methode genutzt. Diese verwendet dann – in unserem Beispiel – die gleichnamige Methode aus der Buch-Klasse.
Gibt es keine zeigeBuchInfo()-Methode in der Fachbuch-Klasse, dann wird dies aus der übergeordneten Klasse genutzt. Natürlich fehlen dann die zusätzlichen Fachbuch-Informationen.

8.11.x.y. Tips und Tricks zu Objekt-orientierten Programmen / Klassen-Definitionen

um eine Klassen-Defintion mit einem kleinen Test-Programm auch als Modul benutzen zu können, gibt es den folgenden Konstrukt, der nur dann den THEN-Zweig ausführt, wenn der Quelltext als Haupt-Programm (MAIN) ausgeführt wird. Ist der Quelltext ein Modul wird dieser Zweig nicht ausgeführt.

```
in __name__ == "__main__":
    # hier stehen die Anweisungen für die Nutzung als (Haupt-)Programm
```

8.11.x. OOP-Programmbeispiele

```
# -*- coding: utf8 -*-

  # Klasse zur Verwaltung von Personen
  class Person(object):
    # Konstruktor/Initialisierer
    def __init__(self, alter, groesse, name = None):
      self.alter = alter
      self.groesse = groesse
      self.name = name

    # String-Repräsentation einer Person erstellen
    def __repr__(self):
      return repr((self.alter, self.groesse, self.name))

    # einfache String-Repräsentation einer Person erstellen
    def __str__(self):
      return '%s/%s/%s' % (self.alter, self.groesse, self.name)

    # Person altern lassen, also Alter um n Jahre erhöhen
    def altern(self, n = 1):
      self.alter += n

    # mittels eines Dekorators eine Property mytuple erzeugen
    @property
    def mytuple(self):
      # das ist der Getter; den Namen lassen wir hier aus
      return self.alter, self.groesse
    # alternativ:
    #   def mytuple(self): return self.alter, self.groesse
    #   mytuple = property(mytuple)

    # einen Setter für die Property definieren
    @mytuple.setter
    def mytuple(self, t):
      if t[0] > 10 and t[1] > 150:
        self.alter, self.groesse = t[:2]
        if len(t) > 2:
          # wenn t einen Namen enthält, dann diesen auch setzen
          self.name = t[2]

  # Personenliste erstellen
  personen = [Person(39, 172, 'ABC'), Person(88, 165), Person(15, 181),
Person(88, 175)]

  # Ausgabe der Personenliste
  print personen
  print '==='

  # Iteration über der Personenliste und Ausgabe der einzelnen Personen
  for pers in personen:
    print pers, '==>', repr(pers)
  print '==='

  # alle Personen altern lassen
  for pers in personen:
    pers.altern(3)

  # nochmal ausgeben
```

```
print 'nach dem Altern'
print personen
print '==='

# nochmal altern lassen, diesmal funktional
map(lambda x: x.altern(3), personen)
print 'nach dem 2. Altern'
print personen
print '==='

# nochmal funktional altern lassen, diesmal mit Vorzugswert n
map(Person.altern, personen)
print 'nach dem 3. Altern'
print personen
print '==='

# Ausgabe der sortierten Personenliste
print sorted(personen, key = lambda pers: (pers.alter, pers.groesse))
print '==='

# dito mit benannter Funktion statt einer anonymen lambda-Funktion
def pers_key(pers):
  return pers.alter, pers.groesse

print sorted(personen, key = pers_key)

# Attribute sind public, man kann von außen zugreifen
print personen[0].alter
print personen[0].groesse
print personen[0].name

p = personen[0]
print p.alter + p.groesse

# Nutzung der Property mit Getter
print p.mytuple
p.alter += 100
print p.mytuple

# Nutzung des Setters der Property
p.mytuple = 1, 2 # wird vom Setter stillschweigend ignoriert
print repr(p.mytuple)

p.mytuple = 11, 155
print repr(p.mytuple)

# nochmal, aber mit Name
p.mytuple = 11, 155, 'Pumuckl'
print repr(p.mytuple)

for p in personen:
  print p

print 'maximales Element einer Personen-Liste bestimmen'
print max((person.alter, person.groesse) for person in personen)

# alternativ nutzbar wären
print max(map(lambda elem: (elem.alter, elem.groesse), personen))
print max(personen, key = lambda elem: (elem.alter, elem.groesse))

print

# Größe von außen ändern
personen[1].groesse += 5
```

```
for p in personen:
  print p

print

# neues Attribut setzen
personen[1].name2 = 'XYZ'
for p in personen:
  print p # __str__() wird für die String-Darstellung gerufen

print

# hier sieht man das neue Attribut
for p in personen:
  print vars(p)

print

print 'Maximum der Property mytuple'
print max(person.mytuple for person in personen)

# das Tupel der letzten Person der Liste ändern
personen[-1].mytuple = 110, 190

# Maximum erneut ausgeben
print 'Maximum der Property mytuple nach Zuweisung'
print max(person.mytuple for person in personen)

# nochmal alle Attribute mit vars()
print
for p in personen:
  print vars(p)
```

Q: https://www-user.tu-chemnitz.de/~hot/PYTHON/

8.12. GUI-Programme mit Tkinter

Tkinter ist nicht etwa eine Fortsetzung ode Erweiterung der Turtle-Graphik. Nein, es ist genau anders herum – die Turtle-Graphik basiert auf dem mächtigen Graphik-Modul Tkinter.
Aber die Turtle-Graphik ist einfach der bessere Einstieg in die graphische Programmierung. Es macht richtig Laune, der Schildkröte zuzusehen.
Echte graphische Aufgaben löst man dann eher mit Tkinter.
Tkinter ist die Python-Schnittstelle zur Graphik-GUI "Tcl/Tk" (GUI … Graphical User Interface)

Tk ist das GUI-Erweiterung für Tcl
Tcl ist eine Scriptsprache die 1991 von John OUSTERHOUT entwickelt wurde
zu Tk und Tcl gobt es Schnittstellen für die verschiedensten Programmiersprachen (z.B. Perl, Ruby, Common LISP, Ada, R , …)

Tk stellte Widgets (Steuerelemente, Bedienelemente) für die Erstellung / Zusammenstellung und Funktionalisierung von graphischen Programmen zur Verfügung

Für die in graphischen Oberflächen weniger bewanderten folgt hier eine Vorstellung / Zusammenstellung von Objekten, die eben Tk – wie andere Programmier-Systeme eben auch – bereitstellt.

Tk-Widgets

- **button** Schaltfläche
- **canvas** Graphik-Fläche
- **checkbutton** Options-Feld
- **combobox** Auswahl-Box
- **entry** Eingabefeld
- **frame** Fenster-Bereich; Container für andere Objekte
- **label** Beschriftung(sfeld)
- **labelframe**
- **listbox** Listen-Feld
- **menu** Menü
- **menubutton** Menü-Eintrag
- **message** Text-Feld
- **notebook**
- **panedwindow**
- **progressbar**
- **radiobutton** Auswahl-Feld
- **scale** Gleiter
- **scrollbar** (Bild-)Laufleiste
- **separator** Fenster-Teiler
- **sizegrip**
- **spinbox**
- **text** Text-Feld
- **treeview**
- **tk_optionMenu**

Für die Integration in das Windows-System werden die klassischen System-Fenster bereitgestellt. Dazu gehören:

Tk-System-Fenster:

- **tk_chooseColor**
- **tk_chooseDirectory**
- **tk_dialog**
- **tk_getOpenFile**
- **tk_getSaveFile**
- **tk_messageBox**
- **tk_popup**
- **toplevel**

Geometrie-Manager organisieren die Anordnung der Bedien-Elemente im Fenster / auf der Fensterfläche

Tk-Geometrie-Manager

- **pack** einfache Geometrie, Objekte werden vorrangig untereinander angeordnet
- **grid** Gitter- bzw. Tabellen-orientierte Geometrie; Objekte werden an Gitter-Plätzen innerhalb des Fensters angeordnet (Spalten: 0 bis x; Zeilen: 1 .. y)
- **place** genaue (absolute) oder relative Platzierung der Objekte

Mit Tkinter kann man auch auf der Konsolen-Ebene arbeiten. Bei manchen Aktionen ist sogar sehr sinnvoll. In der Praxis sind abe eher nachnutzbare Programme interessant. Deshalb werden wir hier fast ausschließlich zusammenhängende Quell-Texte schreiben und diese dann ausprobieren.

8.12.1. … und der erste Programmierer sprach: "Hallo Welt!"

Wir gehen mal ganz klassisch vor. Also …

Aufgabe:
Geben Sie das folgende Programm ein! Die vielen Leerzeilen sind nicht wirklich notwendig, sie dienen nur der Strukturierung für die Erläuterungen rechts neben dem Quelltext.

```
from tkinter import *

fenster=Tk()




elem=Label(fenster,text="Hallo Welt!")
elem.pack()


fenster.mainloop()
```

import des Tkinter-Moduls

Erzeugen eines Root-Objektes namens **fenster** vom Typ Tk

Erzeugen eines untergeordneten Label-Objektes namens **elem** und einem Text

mit der Pack-Methode wird das untergeodnete Objekt integriert

realisiert die Anzeige der Objekte und erzeugt die Ereignis-Abfrage-Schleife

Die etwas ungewöhnliche Notierung ist ein typisches Beiwerk der sogenannten Objektorientierten Programmierung. Das irgnorieren wir hier einfach und zwingen uns zu dieser Schreibweise. Später werden wir sie verstehen, hier ist sie erst mal als gegeben / notwendig zu akzeptieren.
Das Ergebnis sieht natürlich richtig Windows-like aus, aber provoziert sofort die Fragen:
Geht der Text auch in farbig?
Kann man den Text größer oder in einer anderen Schriftart darstellen?

Bevor aber dazu kommen, schnell ein paar Hinweise zur Notierung der Import-Anweisung und er sich daraus ergebenen Notierung im weiteren Quell-Text.
Das gleiche "Hallo Welt!-Programm kann auch mit der Import-Zeile:

```
import tkinter as tk
```

beginnen. Das Programm würde dann so aussehen:

```
import tkinter as tk

fenster=tk.Tk()

elem=tk.Label(fenster, text="Hallo Welt!")
elem.pack()

fenster.mainloop()
```

Diese Notierung wird man gegegebenfalls auch in verschiedenen Beispielen aus Büchern oder dem Internet finden. Man muss allerdings jetzt vor jedem benutztem Objekt noch die Herkunft von tk (als solches haben wir das Modul Tkinter ja jetzt importiert) mit angeben. Das bedeutet nur deutlich mehr Schreibaufwand. Nur bei Kombinationen und Überschneidungen mit anderen Objekten / Modulen ist diese Schreibung sinnig.
Ebenfalls funktioniert die vereinfachte Import-Anweisung:

import tkinter

Kommen wir zu Gestaltung / Formatierung eines Label zurück. Natürlich können wir Farben und Schriften verändern und es auch nicht wirklich schwer. Wegen der ungewöhnlich Text-aufwändigen Notierung, sollten wir uns gleich an eine übersichtlichere Strukturierung des Quell-Textes gewöhnen. Hier sit es allerdings nicht so, dass diese – wie bei Verzweigungen Schleifen und Funktionen – notwendig ist.

```
from tkinter import *

fenster=Tk()

elem=Label(fenster,
           text="Hallo Welt!",
           fg="blue",
           bg="yellow",
           font="Times 12 bold"
           ).pack()

fenster.mainloop()
```

auch möglich:
font=('Times','12','bold')

Schaut man sich nebenstehendes Resultat des obigen Quelltextes an, dann werden die einzelnen Eigenschaften(-Kürzel) schnell klar.
Typische Schriftarten sind **"Courier", "Arial", "Comic Sans MS", "Verdana", "System", "Fixedsys", "MS Sans Serif", "Symbol", "Helvetica"** und **"ansi"**.
Die anderen Schriftstil-Bezeichnungen sind **"normal"**, **"italic"** für kursiv, **"roman"** für ???, **"underline"** für unterstrichen und **"overstrike"** für durchgestrichen.
Einige Worte noch zu der seltsamen **mainloop()**-Funktion am Ende der meisten hier gezeigten Programme. Mainloop startet die Ereignis-Abfrageschleife für das gestartete Programm. Irgenwie soll es ja auf Maus-Klicks oder Tastatur-Befehle reagieren. Genau das realisiert die mainloop()-Funktion. Die **mainloop()**-Funktion wird mit dem Zerstören des (Haupt-)Programm-Fensters durch das destroy-Kommando beendet.
Tkinter-Programme ohne mainloop lassen sich nur in bzw. mit IDLE benutzen. Dort übernimmt der Interpreter die Ereignis-Verarbeitung bzw. übergibt sie zeitweilig an das gerade benutzte Programm.
Programme, die eigenständig funktionieren sollen – also auch direkt aus dem Arbeitsplatz oder dem Windows-Explorer heraus gestartet und benutzt werden sollen – müssen am Schluss die **mainloop()**-Funktion enthalten. Sie können und sollten dann auch als **pyw**-Datei gespeichert werden. In diesem Dateityp lassen sich die Programme direkt unter Windows etc. ausführen

8.12.2. Nutzung verschiedener Bedienelemente

Die Vielfalt der verfügbaren Bedienelemente ist in Windows und in Tk recht groß. Viele sind für eine moderne Interaktion mit den Programmen toll, aber nur wenige Elemente sind für rein funktionelle Programme wirklich notwendig. Diese werden wir hier vorstellen.
Wer mit diesen klar kommt, kann sich dann in die höheren Sphären der GUI-Programmierung begeben.

8.12.2.1. Button's - Schaltflächen

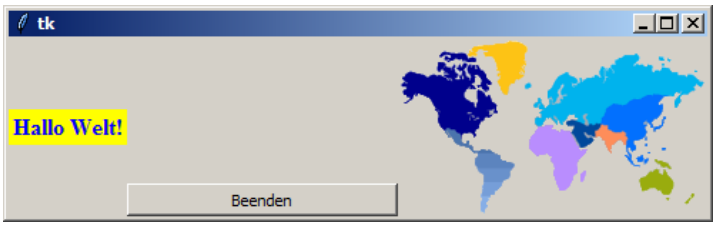
Unser erstes "Hello Welt!"-Programm ließ sich nur über die Fenster-Schaltflächen schließen. Für einfache Programme ist das auch ok, aber wir wollen ja später doch ein bisschen professioneller arbeiten. Also müssen Schaltflächen in die Programme rein.
Beginnen wir mit einem Beenden-Button, der natürlich auch die passende Funktion verpasst bekommen soll.

```
from tkinter import *

fenster=Tk()

elemLabel=Label(fenster,
                   text="Hallo Welt!",
                   fg="blue",
                   bg="yellow",
                   font="Times 12 bold"
            ).pack()
elemButton=Button(fenster,
                  text='Beenden',
                  width=30,
                  command=fenster.destroy
            ).pack()
fenster.mainloop()
```

Die Fenster-Elemente hätte man alle auch nur mit elem bezeichnen können. Später wollen wir aber doch mal das eine oder andere Detail eines Bedien-Elementes ändern. Deshalb bekommt jedes Objekt einen eigenen Namen. Ein einfaches Durchzählen ist natürlich auch möglich.
Nun soll noch ein kleines Bildchen – ein Icon – mit angezeigt werden. Zuerst einmal soll das Bildchen rechts nebendem "Hallo Welt!"-Text erscheinen. Dazu muss der verfügbare Raum verteilt werden. Es gibt ein Label-Objekt links und ein weiteres Label-Objekt – mit der Bildchen – rechts. Die Bildchen müssen als (nicht-animierte) GIF-Datei vorliegen (alternativ gehen auch PGM- bzw. PPM-Dateien).

```
from tkinter import *

fenster=Tk()

elemLabel1=Label(fenster,
                   text="Hallo Welt!",
                   fg="blue",
                   bg="yellow",
                   font="Times 12 bold"
            ).pack(side="left")

bildchen=PhotoImage(file="erde50.gif")
elemLabel2=Label(fenster,
                   image=bildchen
            ).pack(side="right")

elemButton=Button(fenster,
                  text='Beenden',
                  width=30,
                  command=fenster.destroy
            ).pack(side="bottom")

fenster.mainloop()
```

Wer ein bisschen mit den side-Parametern in der Pack-Methode rumspielt, wird schnell merken, dass das Positionieren der Widgets (Bedienelemente) nicht so ganz ohne ist.



Ein Label kann nun auch dafür benutzt werden, um Daten auszugeben. Für einen ersten Versuch soll im Hintergrund ein Zähler laufen, der nach einer Sekunde den Text des Labels ändert. Als Text wird der aktuelle Zähler-Stand genutzt.

```
from tkinter import *

fenster=Tk()

zaehler=0
def zaehlerLabel(label):
  def zaehlen():
    global zaehler
    zaehler+=1
    label.config(text=str(zaehler))
    label.after(1000, zaehlen)
  zaehlen()


fenster.title("Zähler")

elemLabel=Label(fenster, font="Times 20 bold")
elemLabel.pack()

zaehlerLabel(elemLabel)

elemButton=Button(fenster,
                  text='Beenden',
                  width=20,
                  command=fenster.destroy)
elemButton.pack()

fenster.mainloop()
```

Zähl-Funktion

hier wird der Label-Text neu festgelegt

Beschriftung / Titel des Fensters

Label, der für die Anzeige des Zählers genutzt werden soll

starten der Zähler-Funktion

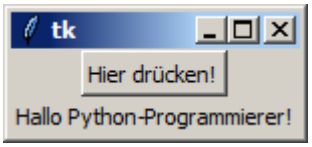
Nach dem Programm-Start beginnt der Zähler zu zählen und nach jeweils 1000 ms (= 1 s) ändert die Funktion den Text des Labels mit dem neuen Zähler-Wert.

8.12.2.1.1. eine eigene Button-Aktion erstellen

Wollen wir nun einem Button eine eigene Aktion programmieren. Dabei müssen zwei Dinge erledigt werden. Einmal müssen wir eine Aktion als Funktion definieren und zum Anderen muss diese Aktions-Funktion der Betätigung der Schaltfläche zugeordnet werden.
Letzteres passiert in den Optionen des erstellten Button als command-Attribut. Der Name der Funktion sollte eindeutig sein, damit später bei vielen Aktionen eine klare Zuordnung möglich ist.
Die Funktion selbst wird ganz klassisch angelegt und in unserem Fall einfach mit einem neu anzulegenen Label versehen.

```
from tkinter import *

def button_aktion():
    elem=Label(fenster,text="Hallo Python-Programmierer!")
    elem.pack()


fenster=Tk()

elem=Button(fenster,
            text="Hier drücken!",
            command=button_aktion)
elem.pack()

fenster.mainloop()
```

Beim Starten des Programm bekommen wir zuerst das obere Fenster angezeigt. Sobald die Schaltfläche gedrückt wird, erscheint der untere Text.



Rücksetzen des Zählers über einen weiteren Button

8.12.2.1.2. Button gestalten / formatieren

bitmap="questhead"

bitmap="error"

bitmap="warning"

bitmap="error"

bitmap="hourglass"

Neben reinem Text lassen sich Button auch mit verschiedensten anderen Details darstellen.
Ersetzt man z.B. das Text-Argument durch ein Bitmap-Argument, dann erscheinen statt dem Text je nach Option verschiedenen kleine Icon's.
Nebenstehend sind die wichtigsten Beispiele aufgezeigt. Solche kleinen Schaltflächen eignen sich z.B. zum Aufrufen von kleinen Hilfestellungen usw.

```
from tkinter import *

def button_aktion():
    elem=Label(fenster,text="Super gemacht!")
    elem.pack()

fenster=Tk()

elem=Button(fenster,
            text="Drücke jetzt!",
            bitmap="warning",
            command=button_aktion)
elem.pack()

fenster.mainloop()
```

Text und Bild lassen sich aber nicht direkt nebeneinander in einer Schaltfläche anzeigen. Die Bild-Option hat Vorrang und der Text wird ignoriert.

8.12.2.2. Nachrichten-Felder / Text-Felder

Label sind doch recht beschränkte Objekte. Für Beschriftungen sind sie völlig ausreichend. Längere Texte oder Nachrichten lassen sich damit nur sehr aufwändig darstellen. Ein etwas flexibleres Widget ist die message.
Mit ihr lassen sich längere, mehrzeilige mit zusätzlichen Hervorhebungen realisieren.

ausgewählte Optionen für Message-Widgets

- **background bg** Hintergrundfarbe; wenn keine angegeben wird, dann wird die Systemeinstellung genutzt
- **font** Schrift-Art, -Größe und -Stil; wenn nichts angegeben wird, dann wird die Systemeinstellung genutzt
- **foreground fg** Vordergrundfarbe / Textfarbe; wenn keine angegeben wird, dann wird die Systemeinstellung genutzt
- **text** anzuzeigender Text
- **textvariable** ist eine spezielle Textvariable, die im Hintergrund geändert werden kann → die Anzeige ändert sich entsprechend
- **takefocus** wird dieser auf True gesetzt, dann erhält die Message-Box den Eingabe- / Bedien-Focus; normalerweise ist der Wert: False

Weitere Optionen sind **anchor, aspect, borderwidth, cursor, highlightbackground, highlightcolor, highlightthickness, justify, padx, pady, relief** und **width**. Wenn für spezielle Gestaltungen von Message-Boxen Bedarf besteht, dann kann man sich in den üblichen Quellen informieren. Für Standard-Anwendungen sind sie nicht notwendig.

8.12.2.3. Eingabe-Felder / Eingabezeilen

Widmen wir uns nun der Eingabe von Texten, Zahlen usw. Dafür sind primär die Entry-Objekte gedacht. Ein erstes Programm soll die gewaltige Aufgabe lösen, aus eingegebenen Vor- und Nachnamen eine ordentliche Begrüßung zu erzeugen!

```
from Tkinter import *          ## Tkinter importieren
root=Tk()                      ## Wurzelfenster!
eingabe = Entry(root)          ## Eingabezeile erzeugen
eingabe.pack()                 ## und anzeigen
```

.get()
gibt Eingabezeile als Text (!) zurück

.delete(position)
löscht Zeichen an der Position (Zählung beginnt bei 0)

.insert(END, text)

```
e = Entry(master)
e.pack()

e.delete(0, END)
e.insert(0, "a default value")
```

```
s = e.get()
```

```
v = StringVar()
e = Entry(master, textvariable=v)
e.pack()

v.set("a default value")
s = v.get()
```

```
from Tkinter import *

master = Tk()

e = Entry(master)
e.pack()

e.focus_set()

def callback():
    print e.get()

b = Button(master, text="get", width=10, command=callback)
b.pack()

mainloop()
e = Entry(master, width=50)
e.pack()

text = e.get()
def makeentry(parent, caption, width=None, **options):
    Label(parent, text=caption).pack(side=LEFT)
    entry = Entry(parent, **options)
    if width:
        entry.config(width=width)
    entry.pack(side=LEFT)
    return entry

user = makeentry(parent, "User name:", 10)
password = makeentry(parent, "Password:", 10, show="*")
content = StringVar()
entry = Entry(parent, text=caption, textvariable=content)

text = content.get()
content.set(text)
```

mehr: http://www.wspiegel.de/tkinter/tkinter02.htm

8.12.2.4. Nachrichten-Boxen

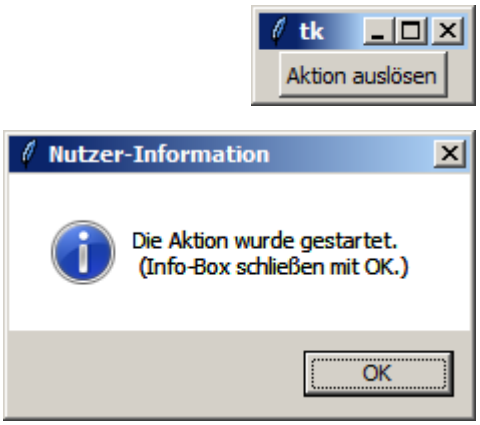
Vielfach sollen kleine Informationen, Fehlerhinweise usw. usf. auf dem Bildschirm gebracht werden. Diese Art der Nutzer-Information wird Message-Box genannt und
Windows – und die meisten anderen graphischen Betriebssysteme – kennen mehrere Arten von Message-Boxen. Diese unterscheiden sich praktisch vor allem hinsichtlich der eingeblendeten Icon's und / oder der Farbgebung.
Klassisch unterscheidet man zwischen "Fehler"-, "Warnung"-, "Frage"- und "Information"-s-Box.
Von der Ebene des Programmierers aus sind alle gleich.
Die Message-Boxen von Python sind etwas breiter aufgestellt. Zumindestens scheint es so. Die Abfrage- / Frage-Box kann mit unterschiedlichen Schaltflächen (Button's) versehen werden. Aus den verschiedenen Varianten ergeben sich unterschiedliche Box-Typen:

Tk-Message-Boxen

- **showinfo** Message-Box mit einer Info-Blase als Icon
- **showwarning** Message-Box mit Warn-Zeichen
- **showerror** Message-Box mit Fehler-Icon
- **askyesno** Message-Box mit Frage-Zeichen als Icon und den Schaltflächen [Ja] und [Nein]
- **askokcancel** Message-Box mit Frage-Zeichen als Icon und den Schaltflächen [OK] und [Abbrechen]
- **askretrycancel** Message-Box mit Frage-Zeichen als Icon und den Schaltflächen [Fortsetzen] und [Abbrechen]

```
from tkinter import *

def antwort():
    titel="Nutzer-Information"
    nachricht="Die Aktion wurde gestartet.\n
               (Info-Box schließen mit OK.)"
    messagebox.showinfo(titel,nachricht)


fenster=Tk()

button1=Button(fenster,
                text="Aktion auslösen",
                command=antwort)
button1.pack()

fenster.mainloop()
```

die Texte für Titel und Nachricht können natürlich auch direkt in den Aufruf der MessageBox notiert werden
\n steht für einen Zeilenumbruch



Wenn bei Ihnen die Message-Box-Fenster anders aussehen, dann ist das dem benutzten Betriebssystem geschuldet. Diese Message-Boxen werden nämlich direkt von Windows (oder dem jeweils benutzten Betriebssystem) direkt zur Verfügung gestellt.

Beide Fenster übergeben sich immer gegenseitig den Focus.
Erst ein "Schließen" des Haupt-Fensters ("tk") beendet das Wechselspiel.

Ein Nachteil der Message-Boxen ist sicher, dass sie immer gleichartig aussehen. Dafür kann man Informationen, Warnungen usw. usf. schnell und effektiv programmieren.
Einfache Message-Boxen (klassischerweise Info-Boxen) lassen sich temporär in Programme integrieren, um sich Zwischenwerten usw. anzeigen zu lassen. Dann muss man nicht jedes Mal das Layout des Programm-Fenster bemühen.

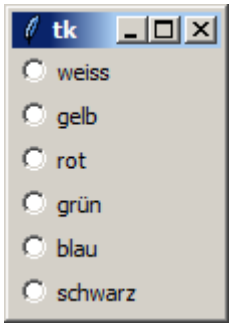
8.12.2.5. Checkbutton-Wdget's – Options-Felder

```
from Tkinter import *

master = Tk()

var = IntVar()

c = Checkbutton(master, text="Expand", variable=var)
c.pack()

mainloop()
```

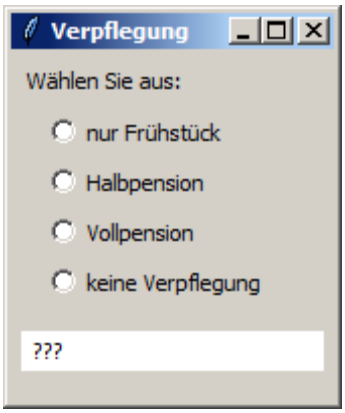
```
var = StringVar()
    c = Checkbutton(
        master, text="Color image", variable=var,
        onvalue="RGB", offvalue="L"
        )


v = IntVar()
    c = Checkbutton(master, text="Don't show this again", variable=v)
    c.var = v
```

8.12.2.6. Radiobutton-Widget – Options-Auswahl

```
from tkinter import *

fenster=Tk()

auswahl=IntVar()

Radiobutton(fenster, text="weiss",
            variable=auswahl, value=1).pack(anchor=W)
Radiobutton(fenster, text="gelb",
            variable=auswahl, value=2).pack(anchor=W)
Radiobutton(fenster, text="rot",
            variable=auswahl, value=3).pack(anchor=W)
Radiobutton(fenster, text="grün",
            variable=auswahl, value=4).pack(anchor=W)
Radiobutton(fenster, text="blau",
            variable=auswahl, value=5).pack(anchor=W)
Radiobutton(fenster, text="schwarz",
            variable=auswahl, value=6).pack(anchor=W)

mainloop()
```



```
from tkinter import *

# Ereignisverarbeitung
def gewaehlt():
    if wahl.get()==1:
        anzeige="Sie wählen nur Frühstück"
    elif wahl.get()==2:
        anzeige="Sie wählen Halbpension"
    elif wahl.get()==3:
        anzeige="Sie wählen Vollpension"
    elif wahl.get()==4:
        anzeige="Sie wählen keine Verpflegung"
    else:
        anzeige="Sie haben noch nicht gewählt"
    ergebnis.config(text=anzeige)

# Erzeugung des Fensters
tkFenster = Tk()
tkFenster.title('Verpflegung')
tkFenster.geometry('160x175')
# Aufforderungslabel
aufforderung=Label(master=tkFenster, text="Wählen Sie aus:", anchor='w')
aufforderung.place(x=5, y=5, width=140, height=20)
# Kontrollvariable
wahl=IntVar()
# Radiobutton Optionsauswahl
rb1=Radiobutton(master=tkFenster, anchor='w', text='nur Frühstück',
                value=1, variable=wahl, command=gewaehlt)
rb1.place(x=15, y=30, width=140, height=20)
rb2=Radiobutton(master=tkFenster, anchor='w', text='Halbpension',
                value=2, variable=wahl, command=gewaehlt)
rb2.place(x=15, y=55, width=140, height=20)
rb3=Radiobutton(master=tkFenster, anchor='w', text='Vollpension',
                value=3, variable=wahl, command=gewaehlt)
rb3.place(x=15, y=80, width=140, height=20)
rb4=Radiobutton(master=tkFenster, anchor='w', text='keine Verpflegung',
                value=4, variable=wahl, command=gewaehlt)
rb4.place(x=15, y=105, width=140, height=20)
# ev. Vorauswahl
# radiobutton3.select()
# Ergebnislabel
ergebnis=Label(master=tkFenster, bg='white', anchor='w', text=" ??? ")
ergebnis.place(x=5, y=140, width=150, height=20)
# Aktivierung des Fensters
tkFenster.mainloop()
```



Erzeugen einer Radiobutton-Auswahl aus einer Liste (von Tupeln) heraus:

```
MODES = [
        ("Monochrome", "1"),
        ("Grayscale", "L"),
        ("True color", "RGB"),
        ("Color separation", "CMYK"),
    ]

    v = StringVar()
    v.set("L") # initialize

    for text, mode in MODES:
        b = Radiobutton(master, text=text,
                        variable=v, value=mode)
        b.pack(anchor=W)
```

8.12.2.7. Text-Fenster / Text-Widget

```
from Tkinter import *             ## Tkinter importieren
root=Tk()                         ## Wurzelfenster!
textfenster = Text(root)          ## Ein Textfenster erzeugen
textfenster.pack()                ## und anzeigen
```

Zeilen von 1 bis y durchgezählt; Spalten von 0 bis x

.get('anfangzeile.anfangspalte','endezeile.endespalte')
.get('anfangzeile.anfangspalte','endezeile.end')
.get('anfangzeile.anfangspalte',END)

.insert(END, text)
.insert('einfügezeile.einfügespalte', text)
.insert('einfügezeile.end', text)

.delete('anfangzeile.anfangspalte','endezeile.endespalte')
.delete ('anfangzeile.anfangspalte','endezeile.end')
.delete ('anfangzeile.anfangspalte',END)

.see(indexzeile)
.see(END)
scrollt den Text, bis angegebene Zeile sichtbar ist

.yview(indexzeile)
.yview(END)
scrollt den Text, bis angegebene Zeile sichtbar ist
Index wandert nach oben (???)

.mark(markierungsname, 'zeile.spalte')
erstellt eine Markierung an der Position

.index(markierungsname)
gibt den Index einer Markierung zurück

.names()
liefert die Namen der verfügbaren Markierungen zurück

.search(suchtext, 'endezeile.endespalte')
.search(suchtext, 'endezeile.end')
.search(suchtext, END)

.tag_add(text, 'anfangzeile.anfangspalte','endezeile.end')
.tag_config(text, foreground=farbe)
.tag_names()

```
from Tkinter import *
root = Tk()
textfenster = Text(root)
textfenster.pack()
eingabe = Entry(root,width=60)
eingabe.pack(side=LEFT)
def hole():
     textfenster.insert(END, '\n' + eingabe.get())

but = Button(root,text='Hole', command = hole)
but.pack(side = LEFT)
root.mainloop()
```

weitere Teile für ein Chat-Programm (zusätzlich zu obigen Quelltext, bzw. Änderungen)

```
…
root = Tk()
def ende():
    root.destroy()

root.title('Chatten mit Python')
…

…
textfenster = ScrolledText(root,width=90)
textfenster.pack()
…
```

mehr: http://www.wspiegel.de/tkinter/tkinter02.htm

8.12.2.8. Frames – Group-Box's – Gruppen-Boxen

Frame-Widget
besser Container genannt,
beinhalten andere Bedien-Elemente, können so gruppiert angeordnet oder z.B. ein- und ausgeschaltet werden

```
from Tkinter import *

master = Tk()

Label(text="one").pack()

separator = Frame(height=2, bd=1, relief=SUNKEN)
separator.pack(fill=X, padx=5, pady=5)

Label(text="two").pack()

mainloop()
```

```
frame = Frame(width=768, height=576, bg="", colormap="new")
frame.pack()

video.attach_window(frame.window_id())
```

8.12.2.9. Menüs / Menu-Widget

```
from Tkinter import *

def callback():
    print "called the callback!"

root = Tk()

# create a menu
menu = Menu(root)
root.config(menu=menu)

filemenu = Menu(menu)
menu.add_cascade(label="File", menu=filemenu)
filemenu.add_command(label="New", command=callback)
filemenu.add_command(label="Open...", command=callback)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=callback)

helpmenu = Menu(menu)
menu.add_cascade(label="Help", menu=helpmenu)
helpmenu.add_command(label="About...", command=callback)

mainloop()
```

```
root = Tk()

def hello():
    print "hello!"

# create a toplevel menu
menubar = Menu(root)
menubar.add_command(label="Hello!", command=hello)
menubar.add_command(label="Quit!", command=root.quit)

# display the menu
root.config(menu=menubar)
```

```
root = Tk()

def hello():
    print "hello!"

menubar = Menu(root)

# create a pulldown menu, and add it to the menu bar
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Open", command=hello)
filemenu.add_command(label="Save", command=hello)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu)

# create more pulldown menus
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Cut", command=hello)
editmenu.add_command(label="Copy", command=hello)
editmenu.add_command(label="Paste", command=hello)
menubar.add_cascade(label="Edit", menu=editmenu)

helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="About", command=hello)
menubar.add_cascade(label="Help", menu=helpmenu)

# display the menu
root.config(menu=menubar)
```

```
root = Tk()

def hello():
    print "hello!"

# create a popup menu
menu = Menu(root, tearoff=0)
menu.add_command(label="Undo", command=hello)
menu.add_command(label="Redo", command=hello)

# create a canvas
frame = Frame(root, width=512, height=512)
frame.pack()

def popup(event):
    menu.post(event.x_root, event.y_root)

# attach popup to canvas
frame.bind("<Button-3>", popup)
```

```
counter = 0

def update():
    global counter
    counter = counter + 1
    menu.entryconfig(0, label=str(counter))

root = Tk()

menubar = Menu(root)

menu = Menu(menubar, tearoff=0, postcommand=update)
menu.add_command(label=str(counter))
menu.add_command(label="Exit", command=root.quit)

menubar.add_cascade(label="Test", menu=menu)

root.config(menu=menubar)
```

8.12.2.9.2. eine Tool-Bar einbauen

besteht aus einem Frame und Button's

```
from Tkinter import *

root = Tk()

def callback():
    print "called the callback!"

# create a toolbar
toolbar = Frame(root)

b = Button(toolbar, text="new", width=6, command=callback)
b.pack(side=LEFT, padx=2, pady=2)

b = Button(toolbar, text="open", width=6, command=callback)
b.pack(side=LEFT, padx=2, pady=2)

toolbar.pack(side=TOP, fill=X)

mainloop()
```

8.12.2.9.3. eine Status-Zeile (Status-Bar) einbauen

besteht aus einem Frame und Button's

```
class StatusBar(Frame):

    def __init__(self, master):
        Frame.__init__(self, master)
        self.label = Label(self, bd=1, relief=SUNKEN, an-
chor=W)
        self.label.pack(fill=X)

    def set(self, format, *args):
        self.label.config(text=format % args)
        self.label.update_idletasks()

    def clear(self):
        self.label.config(text="")
        self.label.update_idletasks()
```

```
status = StatusBar(root)
status.pack(side=BOTTOM, fill=X)
```

8.12.2.10. Umgang mit Standard-Dialogen

kommen direkt aus dem Betriebssystem
eigentlich gehören auch die Message-Boxen (→) mit dazu

Aufruf immer über Fehler-Behandlung empfehlenswert

```
try:
        fp = open(filename)
    except:
        tkMessageBox.showwarning(
            "Open file",
            "Cannot open this file\n(%s)" % filename
        )
        return
```

8.12.2.11. Listbox-Widget – Auswahl-Listen – List(en)-Boxen

```
from Tkinter import *

master = Tk()

listbox = Listbox(master)
listbox.pack()

listbox.insert(END, "a list entry")

for item in ["one", "two", "three", "four"]:
    listbox.insert(END, item)

mainloop()
```

```
listbox.delete(0, END)
listbox.insert(END, newitem)
```

```
lb = Listbox(master)
b = Button(master, text="Delete",
           command=lambda lb=lb: lb.delete(ANCHOR))
```

```
self.lb.delete(0, END) # clear
for key, value in data:
    self.lb.insert(END, key)
self.data = data
```

```
items = self.lb.curselection()
items = [self.data[int(item)] for item in items]
```

8.12.2.12. Options-Menüs – Auswahl-Schaltflächen

```
from tkinter import *

fenster=Tk()

auswahl=StringVar(fenster)
auswahl.set("weiss") # Vorgabe

optionsButton=OptionMenu(fenster,
                         auswahl,
                         "weiss", "gelb", "rot", "grün",
"blau", "schwarz")
optionsButton.pack()

def uebernehmen():
    print("Die Auswahl lautet(e): ", auswahl.get())
    fenster.quit()

buttonOK=Button(fenster, text="OK", command=uebernehmen)
buttonOK.pack()

mainloop()
```

Die print-Anweisung wird im IDLE-Fenster realisiert.

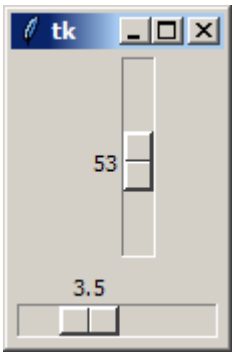
erstellen eines Options-Menüs aus einer Liste von Optionen

```
from Tkinter import *

# the constructor syntax is:
# OptionMenu(master, variable, *values)

OPTIONS = [
    "egg",
    "bunny",
    "chicken"
]

master = Tk()

variable = StringVar(master)
variable.set(OPTIONS[0]) # default value

w = apply(OptionMenu, (master, variable) + tuple(OPTIONS))
w.pack()

mainloop()
```

8.12.2.13. Scale-Widget – Gleiter / Regler

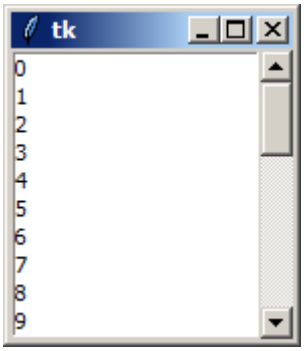
auch slider gnannt

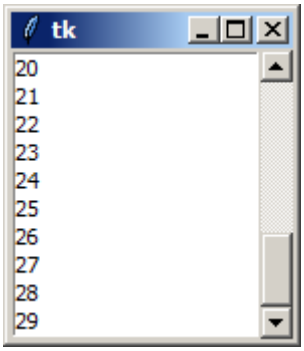
```
from tkinter import *

fenster=Tk()

schieber1=Scale(fenster, from_=0, to=100)
schieber1.pack()

schieber2=Scale(fenster, from_=0, to=12,
                resolution=0.5, orient=HORIZONTAL)
schieber2.pack()

mainloop()
```



8.12.2.14. Scrollbar-Widget - Bildlaufleisten

```
from tkinter import *

fenster = Tk()

laufleiste=Scrollbar(fenster)
laufleiste.pack(side=RIGHT, fill=Y)

auswahlListenBox=Listbox(fenster,
                         yscrollcommand=laufleiste.set)
for nummer in range(30):
    auswahlListenBox.insert(END, str(nummer))
auswahlListenBox.pack(side=LEFT, fill=BOTH)

laufleiste.config(command=auswahlListenBox.yview)

mainloop()
```





8.12.2.15. Widget x

8.12.x. Tkinter – stark, stärker, noch stärker Objekt-orientiert.

Alles was wir bisher mit Tkinter gemacht haben, war schon Objekt-orientiert. Wir benutzten Objekte, wie z.B. Tkinter selbst oder Labels und Buttun und die dazugehörigen Methoden. Wenn man es von Anfang an so macht, dann ist es auch irgendwie gar kein Problem. Wird man aber richtig Objekt-orientiert, dann ist die Welt für den Einsteiger-Programmierer schon schwerer zu durchschauen. Wer die nachfolgenden Programme und Erklärungen nicht gleich versteht, kann vielleicht erst einmal die Grundlagen der Objekt-orientierten Programmierung konsumieren. Ein Rücksprung hierher ist dann gut möglich und macht dann auch wieder Spaß. Graphische Oberflächen machen eben einfach die schöneren Programme.

8.12.x.1. nochmal "Hello Welt!"

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.pack()
        self.createWidgets()

    def createWidgets(self):
        self.hi_there = tk.Button(self)
        self.hi_there["text"] = "Hello World\n(click me)"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.QUIT = tk.Button(self, text="QUIT", fg="red",
                                            command=root.destroy)
        self.QUIT.pack(side="bottom")

    def say_hi(self):
        print("hi there, everyone!")

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

einfache Dialoge

```
from Tkinter import *

class MyDialog:

    def __init__(self, parent):

        top = self.top = Toplevel(parent)

        Label(top, text="Value").pack()

        self.e = Entry(top)
        self.e.pack(padx=5)

        b = Button(top, text="OK", command=self.ok)
        b.pack(pady=5)

    def ok(self):

        print "value is", self.e.get()

        self.top.destroy()


root = Tk()
Button(root, text="Hello!").pack()
root.update()

d = MyDialog(root)

root.wait_window(d.top)
```

```
import tkSimpleDialog

class MyDialog(tkSimpleDialog.Dialog):

    def body(self, master):

        Label(master, text="First:").grid(row=0)
        Label(master, text="Second:").grid(row=1)

        self.e1 = Entry(master)
        self.e2 = Entry(master)

        self.e1.grid(row=0, column=1)
        self.e2.grid(row=1, column=1)
        return self.e1 # initial focus

    def apply(self):
        first = int(self.e1.get())
        second = int(self.e2.get())
        print first, second # or something
  ...
    def apply(self):
        first = int(self.e1.get())
        second = int(self.e2.get())
        self.result = first, second

d = MyDialog(root)
print d.result
```

Überprüfung von Eingaben

```
...

    def apply(self):
        try:
            first = int(self.e1.get())
            second = int(self.e2.get())
            dosomething((first, second))
        except ValueError:
            tkMessageBox.showwarning(
                "Bad input",
                "Illegal values, please try again"
            )

    ...

    def validate(self):
        try:
            first= int(self.e1.get())
            second = int(self.e2.get())
            self.result = first, second
            return 1
        except ValueError:
            tkMessageBox.showwarning(
                "Bad input",
                "Illegal values, please try again"
            )
            return 0

    def apply(self):
        dosomething(self.result)
```

```
from Tkinter import *
import os

class Dialog(Toplevel):

    def __init__(self, parent, title = None):

        Toplevel.__init__(self, parent)
        self.transient(parent)

        if title:
            self.title(title)

        self.parent = parent

        self.result = None

        body = Frame(self)
        self.initial_focus = self.body(body)
        body.pack(padx=5, pady=5)

        self.buttonbox()

        self.grab_set()

        if not self.initial_focus:
            self.initial_focus = self

        self.protocol("WM_DELETE_WINDOW", self.cancel)

        self.geometry("+%d+%d" % (parent.winfo_rootx()+50,
                                  parent.winfo_rooty()+50))
```

```
        self.initial_focus.focus_set()

        self.wait_window(self)

    #
    # construction hooks

    def body(self, master):
        # create dialog body.  return widget that should have
        # initial focus.  this method should be overridden

        pass

    def buttonbox(self):
        # add standard button box. override if you don't want the
        # standard buttons

        box = Frame(self)

        w = Button(box, text="OK", width=10, command=self.ok, default=ACTIVE)
        w.pack(side=LEFT, padx=5, pady=5)
        w = Button(box, text="Cancel", width=10, command=self.cancel)
        w.pack(side=LEFT, padx=5, pady=5)

        self.bind("<Return>", self.ok)
        self.bind("<Escape>", self.cancel)

        box.pack()

    #
    # standard button semantics

    def ok(self, event=None):

        if not self.validate():
            self.initial_focus.focus_set() # put focus back
            return

        self.withdraw()
        self.update_idletasks()

        self.apply()

        self.cancel()

    def cancel(self, event=None):

        # put focus back to the parent window
        self.parent.focus_set()
        self.destroy()

    #
    # command hooks

    def validate(self):

        return 1 # override

    def apply(self):

        pass # override
```

Check-Boxen

```
def __init__(self, master):
        self.var = IntVar()
        c = Checkbutton(
            master, text="Enable Tab",
            variable=self.var,
            command=self.cb)
        c.pack()

    def cb(self, event):
        print "variable is", self.var.get()
```

weiterführende und Quell-Links:
http://www.python-kurs.eu/python_tkinter.php (tolles Tutorial)
http://www.wspiegel.de/tkinter/tkinter_index.htm (kurzes, aber informatives Tutorial)

Tk-Geometrie-Manager

-
-
-

```
>>>
```

8.12.x. diverse Tkinter-Beispiele

aus verschiedenen Quellen:

```
#grafik1.py
from Tkinter import *   #alle Funktionen des Moduls Tkinter werden importiert
fenster = Tk()          #Ein Objekt der Klasse Tk mit Namen fenster wird eingerichtet
fenster.mainloop()      #Die Methode mainloop aktiviert ein Tk-Fenster
```

```
#grafik2.py
from Tkinter import *
fenster= Tk()
fenster.etikett= Label(master=fenster,text= 'Hallo!')      #Ein Objekt der Klasse Label
                                                           #mit Namen fenster.etikett wird erzeugt.
fenster.etikett.pack()                                     #Mit der Methode pack() wird das neue
                                                           #Objekt etikett in die Darstellung des
                                                           #Anwendungsfensters fenster eingebaut.
fenster.mainloop()
```

```
#grafik3.py
from Tkinter import *
fenster= Tk()
fenster.etikett= Label(master=fenster,text= 'Hallo!',
font=('Comic Sans MS',14),fg='blue')           #als Schrifttyp wird Comic Sans MS
                                               #in der Schriftgröße 14;
                                               #Schriftfarbe ist Blau
fenster.etikett.pack()
fenster.title ('Formen')                       #Ueberschrift
leinwand=Canvas(fenster,width=800,height=600,bg="yellow")  #Mithilfe von Canvas-Objekten
                                               #werden Kreise, Rechtecke, Linien
                                               #oder Textobjekte generiert
leinwand.pack()
rechteck=leinwand.create_rectangle(40,20,160,80,fill="Moccasin")
kreis=leinwand.create_line(270,290,450,350,width=10,fill="Lightblue")
vieleck=leinwand.create_polygon(500,80,500,120,600,120,500,80, fill ="white")
streckenzug=leinwand.create_line(270,290,450,350,300,200,
arrow = LAST, width =10, fill = "blue")        #andere Werte fuer arrow: #FIRST, BOTH
spruch=leinwand.create_text(300,50,text="Aller Anfang ist schwer!"),
font=('Arial',14), fill="green")
fenster.mainloop()
```

Q: ???

8.13. Internet

8.13.x. Python und das http-Protokoll

Variante 1

```
import requests

adresse="http://www.lsp-dre.de"

antwort = requests.get(adresse)

print(antwort.ststus_code)
print(antwort.headers['content-type'])
print(antwort.encoding)
print(antwort.text[:80])
```

Variante 2

```
import urllib.request
adresse="http://www.lsp-dre.de"
seite=urllib.request.urlopen(adresse)
seiteninhalt=seite.read()
print(seiteninhalt)
seite.close()
```

Wichtig ist es, zumindestens für das erste Ausprobieren, eine einfache Internetseite abzufragen. Ansonsten kann die Antwort etlige Seiten lang sein. Im Beispiel-Fall ist das eine ganz einfach gestrickte Umleitung auf eine andere Internetseite.

```
>>>
b'\xef\xbb\xbf<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">\n<HTML>\n<HEAD>\n  <TITLE>lern-
soft-projekt: drews</TITLE>\n</HEAD>\n\n<BODY BGColor=#3333FF
Text=#FFFF99 Link=#FFFFFF VLink=#FFFF66>\n<H2>Homepage lern-soft-
projekt: drews</H2><HR>\n<P>Derzeit wird diese Domain nicht bedient.
Nutzen Sie bitte:<P>\n<DIV align="center"><H2><A href="http://www.lern-
soft-projekt.de/">www.lern-soft-
projekt.de</A></H2></DIV>\n</BODY>\n</HTML>'
>>>
```

Natürlich können wir nun den zurückgelieferten HTML-Text auch weiterverwenden.
Wollten wir einen Browser programmieren, müssten wir jetzt nach und nach alle Tags auswerten und in eine Seiten-Darstellung umsetzen. Dabei sollte dann das herauskommen, was uns ein anderer Browser (Internet-Explorer, Firefox, Opera, Chrome, Safari, …) uns auch liefern würde.
Mit Text wird das vielleicht noch recht einfach gehen, aber spätestens bei Bildern, Videos usw. usf. sind dann schon erweiterte Programmierkenntnisse notwendig.
Wir wollen den HTML-Text einfach nach dem Seiten-Titel durchsuchen. Darüber sollte eigentlich jede Internet-Seite verfügen.

die Beispiel-Seite im Browser Firefox

Wer sich schon mit HTML beschäftigt hat, der weiss, dass der Seiten-Titel zwischen den Tags <TITLE> und </TITLE> zu finden ist. Genau danach wollen wir jetzt suchen.

```
…
startpos=0
while True:
    startpos=seiteninhalt.find("<TITLE>",startpos)
    if startpos == -1:
        break
    endepos=seiteninhalt.find("</TITLE>",startpos)
    if endepos == -1:
        break
    print("gefundener Text:
",seiteninhalt[(startpos+7):endepos])
    startpos=endepos
print("Suche beendet!")
```

```
>>>
gefundener Text:  lern-soft-projekt: drews
Suche beendet!
>>>
```

Wenn wir das Programm ein wenig umgestalten, dann kann auch nach jedem anderen beliebigen Begriffspaar gesucht werden. Ob das Tags sind oder andere Begriffe, ist dabei egal.

```
…
print("-----------")
print("Suche:")
print("")startpos=0
starttag="<H2>"
endetag="</H2>"
while True:
    startpos=seiteninhalt.find(starttag,startpos)
    if startpos == -1:
        break
    endepos=seiteninhalt.find(endetag,startpos)
    if endepos == -1:
        break
    print("Text zwischen ",starttag," und ",endetag,":
",seiteninhalt[(startpos+len(starttag)):endepos])
    startpos=endepos
print("Suche beendet!")
```

```
…
-----------
Suche:

gesuchter Text zwischen  <H2>  und  </H2> :  Homepage lern-soft-projekt: drews
gesuchter Text zwischen  <H2>  und  </H2> :  <A href="http://www.lern-soft-
projekt.de/">www.lern-soft-projekt.de</A>
Suche beendet!
>>>
```

8.13.x. einfacher Web-Server

```
from http.server import HTTPServer, CGIHTTPRequestHandler
import os

os.chdir("/tmp")

# CGIHTTP-Server auf Port 8080 starten
server = HTTPServer(("",8080), CGIHTTPRequestHandler)
server.serve_forever()
```

passendes CGI-script "cgi_test" unter /tmp abgespeichert

```
echo 'Content-Type: text/plain; charset=UTF-8'
echo
echo 'Hallo Welt!'
```

oder als Python-script "cgi_test.py" unter /tmp abgespeichert

```
print('''Content-Type: text/plain; charset=UTF-8
Hallo Welt!''')
```

8.13.x. Python und die eMail-Protokolle (smtp, pop3, imap)

```
# -*- coding: utf8 -*-

  # Mail-Versand mit dem Standard-Modul smtplib

  # Module smtplib und sys importieren
  import smtplib, sys

  # MIMEText aus dem Modul text des Sub-Pakets email.mime des Pakets email
  # importieren;
  # im Dateisystem z.B. unter /usr/lib64/python2.6/email/mime/text.py
  from email.mime.text import MIMEText

  # unser ASCII-Mailtext
  mail_text = '''
  Hello friends,
  this is a simple ASCII mail.
  '''

  # eine MIMEText-Nachricht erstellen
  msg = MIMEText(mail_text)

  # Header setzen
  msg['Subject'] = 'test mail'
  me = msg['From'] = 'otto@hrz.tu-chemnitz.de'
  you = msg['To'] = 'hot@hrz.tu-chemnitz.de'

  # Mail senden
  s = smtplib.SMTP()
  if len(sys.argv) > 1 and sys.argv[1] == 'd':
    # Kommandozeilenargument 1 lautet "d", daher Debug einschalten
    s.set_debuglevel(1)
  #s.connect(host = 'mailbox.hrz.tu-chemnitz.de')
  s.connect()
  s.sendmail(me, [you], msg.as_string())
  s.close()
```

Q: https://www-user.tu-chemnitz.de/~hot/PYTHON/

8.14. besondere mathematische Möglichkeiten in Python

8.14.1. imaginäre Zahlen

Notierung
2 + 3j
(2 + 3j)
complex(2,3**)**

jede Variable kann auch eine imaginäre Zahl beinhalten:
img_zahl = 2.5 - 1.5j
img-zahl**.real** liefert den Real-Teil, also hier 2,5
img_zahl**.imag** liefert den Imaginär-Teil, also hier -1,5 ➔ -1,5i

8.14.2. Matrizen (Matrixes)

Ob es nun Matrizen oder Matrixes heißt, wollen wir hier nicht vertiefen. Ich benutze Matrix für die Einzahl und Matritzen für die Mehrzahl. Das liesst sich aus meiner Sicht einfacher und jeder halbwegs (mathematisch) Eingeweihte, weiss, worum es geht.

Realisierung und Bearbeitung z.B. über geschachtelte Listen (s.a. kurze Einführung: → 8.4. Listen, die I. – einfache Listen)

```
def transponiere(matrix, bisIndex):
    for i in range(bisIndex):
        for j in range(i+1, bisIndex):
            matrix[i][j],matrix[j][i] = matrix[j][i],matrix[i][j]
    return matrix


def testeTransponieren(n):
    matrix= range(n)
    for i in range(n):
        matrix[i]=range(n)
    print("(Original-)Matrix")
    for i in range(n):
        print(matrix[i])
    transpoMatrix=transponiere(matrix)
    print("")
    print("transponierte Matrix")
    for i in range(n):
        print(transpoMatrix[i])
```

```
def multipliziere(matrix1, matrix2):
    laengeM1=len(matrix1)
    multMatrix=range(laengeM1)
    for i in range(laengeM1):
        multMatrix[i]=range(laengeM1)
        for j in range(laengeM1):
            multMatrix[i][j]=0
    for i in range(laengeM1):
        for j in range(laengeM1):
            summme=0
            for k in range(laengeM1):
                summe+=matrix1[i][k]*matrix2[k][j]
            multMatrix[i][j]=sum
    return multMatrix


def testeMultipizieren(laengeMatrix):
    matrixA=range(laengeMatrix)
    matrixB=range(laengeMatrix)
    print(matrixA)
    print(matrixB)
    for i in range(laengeMatrix):
        matrixA[i]=range(laengeMatrix)
        matrixB[i]=range(laengeMatrix)
        for j in range(laengeMatrix):
            matrixA[i][j]=i
            matrixB[i][j]=i
    print(matrixA)
    print(matrixB)
    matrixC=multipliziere(matrixA,matrixB)
    print(matrixC)
```

```
def berechneDeterminante(matrix):
    laengeM=len(matrix)
    if laengeM<=0:
        return 1
    else:
        if laengeM==1:
            return matrix[0][0]
        else:
            summe=0
            neg=-1
            for i in range(laengeM):
                neg=(-1)*neg
                matrixH=matrixcopy(matrix)
                for j in range(laengeM):
                    matrixH.pop(0)
                matrixH-pop(i)
                sum+=neg*matrix[i][0]*berechneDeterminante(matrixH)
    return summe
```

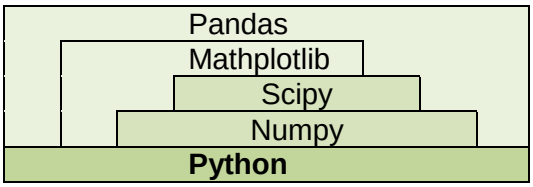
dieser Algorithmus hat eine Laufzeit von $O(2^n)$, es existiert aber auch einer mit $O(n^3)$

8.14.3. Python numerisch, Python für Big Data

auch für Data science, Maschinelles Lernen, Künstliche Intelligenz, …

numerisches Programmieren umfasst einen breiten Teil der Mathematik und meint das Arbeiten mit stetigen Variablen, numerische Analysen, Approximations-Algorithmen, …

in vielen Punkten Ersatz für das kostenpflichtige Matlab
hier genannte Module alle kostenfrei
keine Einschränken durch prohibitive / proprietäre Lizenzen bei der Weiterverwendung



Numpy

stellt die grundlegenden Daten-Typen für numerische Arbeiten und das Händling von Big Data zur Verfügung
dazu gehören mehrdimensionale Array's und Matrizen

Scipy

benutzt die Daten-Typen aus Numpy
bietet vor allem Funktionalitäten für Analysen usw. usf. an, wie z.B. Regression, FOURIER-Transformation, …

Matplotlib

bietet Möglichkeiten der graphischen Darstellung von Daten an

```
1 import mathplotlib.pyplot as zeichnung
2
3 x = [1,2,3,4,5,6]
4 y = [1,4,9,16,25,36]
5 zeichnung.plot(x,y)
6 zeichnung.title("quadratische Funktion")
7 zeichnung.show()
```

Pandas

nutzt alle genannten Module
erweitert diese für Tabellen und Zeit-Reihen

8.15. Behandlung von Laufzeitfehlern – Exception's

try … except … else

Bsp: Zahlenraten
Computer wählt zufällig eine Zahl aus einem Zahlenbereich aus, hier 1 bis 100
der Nutzer soll die Zahl raten; der Computer gibt bein nicht-zutreffen zurück, ob die Zahl zu groß ode zu klein ist; Ziel sind besonders wenige Rate-Vorgänge zu brauchen.

```
from random import randint

geraten=False
SuchZahl=randint(1,100)
print("Der Computer hat eine Zahl erwürfelt?")
print()
zaehler=0
while not geraten:
    # Eingabe
    try:
        eing=int(input("Welche Zahl vermutest Du?: "))
    except ValueError:
        print("")
        continue    # --> Eingabe wiederholen
    # Auswertung
    zaehler+=1
    if eing > Suchzahl:
        print("vermutete Zahl ist zu groß!")
    elif ein < Suchzahl:
        print("vermutete Zahl ist zu klein!")
    else:
        geraten=True
print("Richtig!  ",zaehler," Versuche gebraucht")
print("Spiel-Ende")
```

Die Auswertung könnte auch im optionalen ELSE-Zweig stehen können.

```
>>>

>>>
```

try … except … finally

Bsp: Zahlenraten
Computer wählt zufällig eine Zahl aus einem Zahlenbereich aus, hier 1 bis 100
der Nutzer soll die Zahl raten; der Computer gibt bein nicht-zutreffen zurück, ob die Zahl zu groß ode zu klein ist; Ziel sind besonders wenige Rate-Vorgänge zu brauchen.

```
from random import randint

geraten=False
SuchZahl=randint(1,100)
print("Der Computer hat eine Zahl erwürfelt?")
print()
zaehler=0
while not geraten:
    # Eingabe
    try:
        eing=int(input("Welche Zahl vermutest Du?: "))
    except ValueError:
        print("")
        continue    # --> Eingabe wiederholen
    # Auswertung
    zaehler+=1
    if eing > Suchzahl:
        print("vermutete Zahl ist zu groß!")
    elif ein < Suchzahl:
        print("vermutete Zahl ist zu klein!")
    else:
        geraten=True
print("Richtig!  ",zaehler," Versuche gebraucht")
print("Spiel-Ende")
```

```
>>>

>>>
```

try … finally

raise

pass

leere Anweisung; z.B. als Platzhalter in definierten, aber noch nicht implementierten Funktionen / Klassen / …

8.16. Sortieren – eine Wissenschaft für sich

Dieses Kapitel könnte genauso gut unter dem Abschnitt (→) eingeordnet werden. Was durch den Anfänger vielleicht als überzogene, abgehobene, akademische Auseinandersetzung abgetan wird, ist in der Informatik ein Kernproblem: Wie bekommt man schnell und mit möglichst wenig Speicher-Aufwand eine Liste / ein Feld von Daten sortiert.
Die Algorithmik liefert viele Lösungen mit unterschiedlichen Vor- und Nachteilen. Einige Sortier-Verfahren wollen wir hier vorstellen und unter bestimmten Kriterien bewerten.
Für Anfänger-Listen-Größen von vielleicht maximal einigen hundert Werten machen die Unterschiede meist nicht viel aus. Bei großen - ev. sogar mehrdimensionalen Daten-Strukturen – bekommen die Kriterien dann schon eine ganz andere Bedeutung.
Wir wollen hier versuchen die einzelnen Algorithmen nicht nur zu nennen, sondern auch zu erklären und an einer Beispiel-Datenreihe anzuwenden.
Wenn es geht und wenn es sinnvoll ist, dann werden wir auch die Zwischen-Zustände mittels eines abgewandelten Programm anzuzeigen, um das Verfahren auch in der Praxis zu erleben. Solche Zwischen-Anzeigen bieten sich auch an, wenn man einen Algorithmus auf die eigenen Daten-Strukturen anpasst. Selten geht alles beim ersten Mal glatt.
Anfängern sei empfohlen, sich zuerst einmal den Algorithmus herauszusuchen, bei dem man den Eindruck hat, man versteht ihn und die Umsetzung ins Programm. Dadurch wird die Fehlersuche vereinfacht. Später kann man sich dann den höheren Verfahren zuwenden.

weiterführende Links:
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html (Visualisierung von Algorithmen)

8.16.x. Bubble-Sort

```
1  def bubblesort(liste):
2      laenge = len(liste)
3      for i in range(laenge):
4          geaendert = False
5          for j in range(laenge-i-1):
6              if liste[j] > liste[j+1]:
7                  liste[j],liste[j+1] = liste[j+1],liste[j]
8                  geaendert = True
9          if not geaendert:
10             break
11     return liste
```

8.16.x. Selection-Sort

8.16.x. Quick-Sort

Anwendung des "Teile und herrsche"-Prinzips ("divide and conquer")
algemeines Prinzip zum Lösen von Problemen: Zerteile das Problem in kleinere und löse diese. Dabei darf das prinzip immer wieder angewendet werden, wir arbeiten also rekursiv
irgendwann sind die Teil-Probleme so klein, dass sie schon gelöst sind (Rekursions-Abbruch) oder einfach zu lösen sind

Quicksort besteht aus drei Elementen *(noch nicht perfekt!)*

- **Herrsche / Beauftragen / Befehlen** — wenn die Liste länger als ein Element ist, dann wird sie nach Teilen-Prinzip bearbeitet, ansonsten ist die Liste sortiert
  es kann das Zusammenfügen / Ausgeben erfolgen
- **Teilen** — aus der Liste wird (zufällig) ein Element elem ausgewählt und die Liste in zwei Teillisten zerlegt, wobei Liste1 alle kleineren Elemente als elem enthält und Liste 2 alle größeren oder gleichgroßen (/ anderen)
  die Teillisten werden dem Herrsche-Prinzip zur Prüfung übergeben
- **Zusammenfügen** — die sortierte Liste wird nun zusammengesetzt aus der sortierten Liste der kleineren Elemente, dem Element elem und der sortierten Liste der größeren Elemente

der Algorithmus stammt von HOARE 1962 ist einer der effektiven Sortier-Verfahren
besonders herausragend ist die Zeit-Effektivität

es werden durchschnittlich n log n Vergleiche benötigt

```
def quicksort(liste):

    def teile(links, rechts):
        i = links
        j = rechts - 1
        pivot = liste[rechts]

        while True:
            while liste[i] <= pivot and i < rechts:
                i+=1
            while liste[j] >= pivot and j > links
                j-=1
            if i < j:
                liste[i], liste[j] = liste[j], liste[i]
            else:
                break
        if liste[i] > pivot:
            liste[i], liste[rechts] = liste[rechts], liste[i]
        return i

    def sortieren(links,rechts):
        if links < rechts:
            teiler = teile(links,rechts)
            sortieren(links, teiler-1)
            sortieren(teiler+1, rechts)

    sortieren(0, laenge-1)
    return liste
```

ein Quick-Sort mit Anzeige

```
def quicksort(liste):
    if len(liste)>0:
        print("es wird sortiert: ", liste')
    if len(liste)<=1:
        return liste
    else:
        return quicksort([i for i in liste[1:] if i < liste[0])\
        + [liste[0]]\
        + quicksort([j for j in liste[1:] if j >= s[0]])
```

etwas kryptisch , aber auch so geht es:

```
def quicksort(liste):
    if len(liste) <= 1:
        return liste
    wahlelement = liste.pop()
    links  = [element for element in liste if element < wahlelement]
    rechts = [element for element in liste if element >= wahlelement]
    return quicksort(links) + [wahlelement] + quicksort(rechts)
```

8.16.x. Tree-Sort

8.16.x. Merge-Sort

```
def mergesort(liste):
    def mische(links, rechts):
        gemischt = []
        laengeLinks = len(links)
        laengeRechts = len(rechts)
        while laengeLinks != 0 and laengeRechts != 0:
            if links[0] <= rechts[0]:
                gemischt.append(links[0])
                links = links[1:]
            else:
                gemischt.append(rechts[0])
                rechts = rechts[1:]
        while laengeLinks !=0:
            gemischt.append(links[0])
            links = links[1:]
        while laengeRechts !=0:
            gemischt.append(rechts[0])
            rechts = rechts[1:]
        return gemischt

    def sortieren(liste):
        laenge = len(liste)
        if laenge<=1:
            return liste
        else:
            haelfte=laenge/2
            links = liste[0:haelfte]
            rechts = liste[haelfte:]
            links = sortieren(links)
            rechts = sortieren(rechts)
        return mische(links, rechts)

    return sortieren(liste)
```

8.16.x. Selection-Sort

```
def selectionsort(liste):
    laenge = len(liste)
    for i in range(laenge-1):
        minimum = i
        for j in range(i,laenge):
            if liste[j] < liste[minimum]:
                minimum = j
        liste[minimum],liste[i] = liste[i],liste[minimum]
    return liste
```

8.16.x. Insertion-Sort

```
def insertionsort(liste):
    laenge = len(liste)
    for i in range(1,laenge):
        wert = liste[i]
        j = i
        while j > 0 and liste[j-1] > wert:
            liste[j] = liste[j-1]
            j-=1
        liste[j] = wert
    return liste

```

8.16.x. Gnome-Sort

```
def gnomesort(liste):
    pos = 0
    laenge = len(liste)
    while pos < laenge-1:
        i = pos
        if liste[i] <= liste[i+1]:
            pos+=1
        else:
            liste[i], liste[i+1] = liste[i+1], liste[i]
            if pos !=0:
                pos-=1
            else:
                pos+=1
    return liste
```

8.16.x. Counting-Sort

```
def countingsort(liste):
    laenge = len(liste)
    if laenge == 0:
        return []
    listeA = [0] * (max(liste)+1)
    listeB = [""] * laenge
    for elem in liste:
        listeB[elem]+=1
    for i in range(1,len(listeB)):
        listeB[i]+=listeB[i-1]
    for elem in liste[::-1]:
        listeA[listeB[elem]-1] = elem
        listeB[elem]-=1
    return listeA
```

8.16.x. Radix-Sort

```
def radixsort(liste, k=10, d=0):
    laenge = len(liste)
    if laenge == 0:
        return []
    elif d == 0:
        d = max(map(lambda x: len(str(abs(x))), liste)
    for x in range(d):
        listeA = [[] for i in range(k)]
    for elem in liste:
        listeA[(elem / 10**x) % k].append(elem)
    liste = []
    for bereich in listeA:
        liste.extend(bereich)
    return liste
```

8.16.x. Tim-Sort

8.16.x. Heap-Sort

```
def heapsort(liste):

    return liste
```

8.16.x. Bucket-Sort

8.16.x. -Sort

8.16.x. Vergleich ausgewählter Sortier-Algorithmen

Algorithmus / Name	in place	stabil			Laufzeit-Verhalten B Best-Case	AVG durchschnittlich	W Worst-Case			
Selection-Sort	ja	nein			$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$			
Bubble-Sort	ja	ja			$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$			
Insertion-Sort	ja	ja			$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$			
Quick-Sort	ja	nein			$\Theta(n * \log(n))$	$\Theta(n * \log(n))$	$\Theta(n^2)$			
Heap-Sort	ja	nein			$O(n * \log(n))$	$O(n * \log(n))$	$O(n * \log(n))$			
Merge-Sort	nein, (ja)	ja			$\Theta(n * \log(n))$	$\Theta(n * \log(n))$	$\Theta(n * \log(n))$			
Tim-Sort	nein	ja			$\Theta(n)$	$O(n * \log(n))$	$O(n * \log(n))$			
Radix-Sort	nein	ja					$O(d * (n+k))$			
Counting-Sort	nein	ja					$O(n+k)$			
Bucket-Sort										

interessante Links:
https://www.toptal.com/developers/sorting-algorithms (Animationen zu den verschiedenen Sortier-Algorithmen)
https://www.youtube.com/watch?v=t8g-iYGHpEA (Sortierungen optisch und akustisch veranschaulicht)

(c,p) 2015 - 2022 lern-soft-projekt: drews

8.16.x. das Häufigste Element finden – der Modus

```
def mode(L):
    for i in range(0,100):
        for i in L:
            frequency[i] += 1
                return i
            if frequency[i] == max(frequency):
                frequency=[0]*10
```

```
>>>
```

interessante Links:
http://www.sortierkino.de (zum Zuschauen beim Sortieren; viele Algorithmen im Vergleich)

Beispiel-Implementierung

Q: https://github.com/MartinThoma/algorithms/blob/master/sorting.py

8.17. Nutzung weiterer (/ besonderer) graphischer Benutzer-Oberflächen

8.18. (die hohe Kunst der) Spiele-Programmierung

Nachdem wir einiges dazu schon beim Modul "pygame" besprochen haben (→ 8.9. das Modul "pygame") dringen wir nun noch etwas tiefer in den Sachvehalt ein.

interessante Links:
http://inventwithpython.com/inventwithpython_3rd.pdf (online-Version des Buches: AL SWEIGART: Invert Your Own Computer Games with Python 3rd Edition)
http://inventwithpython.com/makinggames.pdf (online-Version des Buches: AL SWEIGART: Making Games with Python & Pygame)

8.19. Python im Geheimen - Kryptologie

Begeben wir uns in die Welt von Alice und Bob, den beiden Haupt-Agenten in der Kryptologie.

8.19.0. Grundlagen

8.19.0.1. Codierung

"geheime" Codierungen

8.19.0.2. Chiffrierung

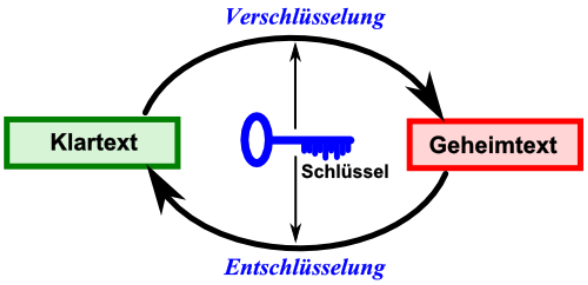
In den folgenden Kapiteln werden wir die Klartexte (unverschlüsselte Texte) grün oder grünlich hinterlegt darstellen. Wenn die Buchstaben-Art keine Rolle spielt, dann werden die Klatrtexte mit Groß-Buchstaben geschrieben.
Die verschlüsselten Texte (Geheimtexte) werden dagegen in Klein-Buchstaben in rot oder rötlich hinterlegt notiert.
Das Standard-Alphabet sind die 26 deutschen Buchstaben ohne Umlaute und ß. Oft wird auch auf das Leerzeichen verzichtet und die Wörter einfach hintereinander geschrieben. Erweiterte Alphabete nutzen Leerzeichen und / oder Ziffern und / oder Satz-Zeichen mit dazu. So etwas definieren wir dann bei den einzelnen Verfahren. Viele Algorithmen sind so ausgelegt, dass sie Nicht-Alphabet-Zeichen einfach ignorieren oder direkt übernehmen.

I.A. geht es vor allem um das Demonstrieren des Verfahren's. Der wichtigste Grund für die Wahl der standardisierten Alphabete ist aber die Vergleichbarkeit der verschiedenen Verfahren. Dabei interessiert uns immer das Agieren der Gegenseite. Kann Sie das Verfahren knacken?

Nicht's ist unangenehmer als eine Geheimschrift, von der man glaubt, sie sein Bomben-sicher und jeder kann aber in der Praxis mit wenig Aufwand mitlesen.

Die Einteilung von Geheim-Schriften / -Verfahren ist ein unendliches Thema. Wir beschränkn uns hier auf zwei elementare Möglichkeiten.

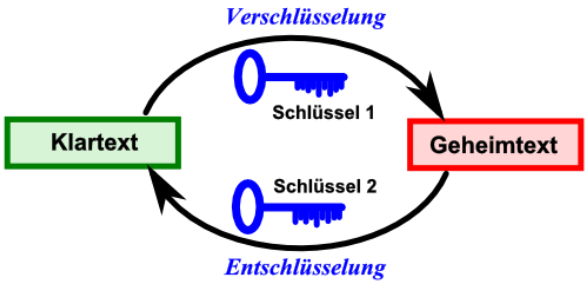
Für die erste Einteilung betrachtet man die Anzahl der verwendeten Schlüssel und das benutzte Verfahren. Wird nur ein Schlüssel und praktisch das gleiche verfahren für Ver- und Ent-Schlüsselung benutzt, dann sprechen wir von **symmetrischer Verschlüsselung**. Klassische Vertreter sind die CÄSAR-Chiffre (→) und (→).



symmetrische Verschlüsselung

Kommen dagegen zwei (zueinander gekoppelte) Schlüssel und praktisch auch zwei Verfahren zum Einsatz, dann handelt es sich um die **asymmetrische Verschlüsselung**.

Beispiele hierfür sind das RSA-Verfahren oder der DES-Algorithmus.



asymmetrische Verschlüsselung

Die zweite Einteilung bezieht sich auf die Art und Weise, wie der Geheimtext erzeugt wird.So kann man z.B. Geheimtexte durch Austauschen der Symbole erzeugen. Wir sprechen hier von **Substitution**. Typische Umsetzungen sind (→) und (→).

Eine weitere weitere Möglichkeit – Texte unleserlich zu machen – sind **Transposition**en. Hierbei bleiben die Symbole des Klartextes erhalten, aber ihre Positionen innerhalb des Textes werden verändert. (→) und (→) sind hier viel zitierte Chiffren.

Die dritte Art verändert die Klartext durch Hinzufügen von Symbolen. Dabei geht es zum Einen darum die Texte unleserlich (oder schwer leserlich) zu machen und zu Anderen sollen Häufigkeits-Analysen ausgetrickst werden. Kryptographen nennen diese Art der Geheimtext-Erzeugung **Erweiterung**. Als typische Vertreter dieser Gruppe können (→) und (→) genannt werden.

Aufgaben:

1. *Vergleichen Sie symmetrrische und asymmetrische Verschlüsselung! Nutzen Sie auch das Internet, um weitere typische Merkmale, Vor- und Nachteile zu erkunden!*
2. *Vergleichen Sie die Erzeugung von Geheimtexten durch Substitution, Transposition und Erweiterung anhand von jeweils mindestens 6 selbstgewählten Kriterien! Versuchen Sie gleichrangig Gemeinsamkeiten und Unterschiede zu finden!*
3. *Informieren Sie sich über weitere Einteilungs-Möglichkeiten und stellen Sie eine in Form eines kurzen Vortrages vor!*

Aufgaben:

4. *Erstellen Sie ein Stammbaum von Geheimsprachen und stellen Sie diese vor!*

8.19.1. symmetrische Verschlüsselung

Symmetrische Verschlüsselungen benutzen für die Ver- und Ent-Schlüsselung (Chiffrierung / Dechiffrierung) immer den gleichen Schlüssel. In praktischallen Fällen kann das gleiche – oder auch das umgekehrte (reverse) – Verfahren genutzt werden.
Das macht symmetrische Verfahren sehr effektiv. Mit Computern können sie sehr einfach umgesetzt werden. Das große Problem sind die Schlüssel. Sie müssen irgendwann ausgetauscht werden. Dieser Vorgang kann von Unbefugten mitgehört / manipuliert / … werden.
Viele der älteren symmetrischen Verfahren bieten durch eine recht geringe Schlüssel-Anzahl auch keine ausreichende Sicherheit mehr. Mit modernen Rechnern sind sie oft innerhalb weniger (milli-)Sekunden durch Brute-Force-Angriffe oder Häufigkeits-Analysen angreifbar.

8.19.1.x. CÄSAR-Verschlüsselung

Das Verfahren geht der Legende nach auf Gaius Julius CÄSAR (100 – 44 v.u.Z) zurück. Zu jener Zeit soll die Chiffre auch nicht gebrochen worden sein. Dazu gab es wahrscheinlich auch zu wenige Menschen, die sich mit Schrift und Alphabet auskannten.
CÄSAR's Verschlüsselung war einfach und effektiv. Er setzte dem Klartext-Alphabet ein zweites gegenüber, dass um 3 Positionen verschoben war. Buchstaben, die keine Entsprechung hatten, wurde an der anderen Seite angelegt.
Wahrscheinlich benutzte CÄSAR auch nur eine Möglichkeit – und zwar eben diese CÄSAR3-Verschiebung.

Beispiel: CÄSAR**3**

Symbole lfd. Nr.	1	2	**3**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Klaralphabet	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Geheimalph.	**y**	**z**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**

Dadurch entsteht ein Buchstaben-Ring. Für andere CÄSAR-Verschlüsselungen wird der Ring einfach weitergeschoben.

Beispiel: CÄSAR**7**

Symbole lfd. Nr.	1	2	3	4	5	6	**7**	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Klaralphabet	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Geheimalph.	**u**	**v**	**w**	**x**	**y**	**z**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**

Auch der Nachfolger von CÄSAR – der Kaiser AUGUSTUS – benutzte eine ähnliche Chiffre. Er benutzte die Verschiebung um einen Buchstaben und verwendete für das X (damals letzter Buchstabe im Alphabet) ein AA als Substituenten.
Die konkrete Umsetzung in Python schauen wir uns etwas später an. Zuerst diskutieren wir einige Programmier-Varianten bei einem speziellen Fall der CÄSAR-Verschlüsselung.

8.19.1.x. ROT13

ROT13 ist eine spezielle Variante der CÄSAR-Verschlüsselung. Genaugesagt handelt es sich um eine CÄSAR13-Chiffre.

Beispiel: CÄSAR**13**

Symbole lfd. Nr.	1	2	3	4	5	6	7	8	9	10	11	12	**13**	14	15	16	17	18	19	20	21	22	23	24	25	26
Klaralphabet	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Geheimalph.	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**

Durch die symmetrische Teilung des Alphabet's ergeben sich einige praktische Besonderheiten.

Es kommt zu einer festen Zuordnung der Buchstaben von Klar- und Geheimtext-Alphabet. Dadurch funktionieren Ver- und Ent-Schlüsselung mit dem exakt gleichen Algorithmus bzw. der gleichen Funktion.

A	B	C	D	E	F	G	H	I	J	K	L	M
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Ursprünglich wurde die ROT13 auch nicht wirklich als Verschlüsselung eingesetzt, sondern als Mittel der sehr einfach und effektiven Verschleierung von Texten. Ursprünglich sollte damit im usenet zweideutige Witze und Texte auf den ersten Blick versteckt werden.

Dabei zielte man auch auf den Effekt hin, dass ein Leser einen ROT13-Text bewußt entschlüsselt. Damit ist er auch für sich verantwortlich, wenn er mit bestimmten Obzönigkeiten, sexuellen Anspielungen usw. usf. nicht klar kommt.

Er hätte es lassen können.

ROT13-Verschlüsselungen sind, wie alle klassischen CÄSAR-Chiffren sehr leicht durch Brute-Force- Angriffe oder Häufigkeits-Analysen knackbar. Mittlerweise sind sie ein Sinnbild für sehr schlechte kryptographische Verfahren.

In den folgenden Programmier-Beispielen zu den verschiedenen Chiffren wollen wir ein grundlegendes Schema benutzen.

1. Eingabe des Klartextes
2. Umwandlung des Klartextes in die notwendige Form (Groß-Buchstaben, ev. ohne Leerzeichen)
3. Erstellen einer Häufigkeits-Analalyse (zu Vergleichszwecken)
4. Verschlüsseln der Klartextes
   a) ev. Anzeige von Zwischenschritten
   b) Anzeige des verschlüsselten Textes
5. Erstellen einer Häufigkeits-Analyse vom Geheimtext
6. Entschlüsseln des Geheimtextes

Entwickeln wir ein solches Programm nun schrittweise für die ROT13-Verschlüsselung.
Zuerst bauen wir uns ein einfaches Programm ohne Funktionen. Diese führen wir dann in einer zweiten Entwicklungs-Reihe ein.
Starten wir mit einem einfachen Eingabe-Ausgabe-Rahmen, in den wir dann als nächstes die Umwandlung der Eingabe in Groß-Buchstaben integrieren.
Die beiden Alphabete legen wir als feste Listen an. Gerade beim ROT13-Verfahren ist dies ja eine der Basis-Vereinbarungen.

```
# ROT13 Ver- und Ent-Schlüsselung
# L. Drews; 2020

# Definitionen
klarAlpha=["A","B","C","D","E","F","G","H","I",
           "J","K","L","M","N","O","P","Q","R",
           "S","T","U","V","W","X","Y","Z"]
geheimAlpha=["n","o","p","q","r","s","t","u","v",
             "w","x","y","z","a","b","c","d","e",
             "f","g","h","i","j","k","l","m"]
laengeKlarAlpha=26

# Eingabe
klarText=input("Klartext  : ")


# Verschlüsselung
geheimText=klarText

print("Geheimtext: ",geheimText)


# Entschlüsselung
dechiffKlarText=geheimText


# Ausgabe
print("Klartext  : ", dechiffKlarText)

input()
```

Mit der Funktion upper() erhalten wir einen Groß-Buchstaben-Text von einem Text-Objekt. (s.a. 8.1.1. Objekt-orientierte Nutzung von Strings)

```
…
# Eingabe
klarText=input("Klartext  : ")
klarText=klarText.upper()
print("KLARTEXT  : ",klarText)


# Verschlüsselung
…
```

In der Verschlüsselung selbst bestimmen wir zuerst die Länge des Klartextes. Desweiteren wird ein leerer Geheimtext angelegt, denn wird dann mit der i-Schleife Zeichen für Zeichen auffüllen wollen.
Bei jedem Schleifendurchlauf separieren wir ein KlarSymbol. Dieses wird mittel j-Schleife im Klar-Alphabet gesucht und sich die Position gemerkt.

```
…
# Verschlüsselung
print("=========> Verschlüsselung ==========>")
laengeKlarText=len(klarText)
geheimText=""
for i in range(laengeKlarText):
    klarSymbol=klarText[i]
    pos=-1
    for j in range(laengeKlarAlpha):
        if klarSymbol==klarAlpha[j]:
            pos=j
            break
    geheimSymbol=geheimAlpha[pos]
    if pos>=0:
        geheimText+=geheimSymbol
    else:
        geheimText+=klarSymbol
print("Geheimtext: ",geheimText)


# Entschlüsselung
…
```

Mit Hilfe der Position holen wir aus das passende GeheimSymbol aus dem geheim-Alphabet und hängen es an den bisher bearbeiteten Geheimtext an. Für den Fall, dass wir kein passende Symbol gefunden haben (pos ist dann immer noch -1), übernehmen wir das Nicht-Alphabet-Symbol.

Für die Entschlüsselung benutzen wir genau den gleichen Algorithmus, nur umgetauschten klar- und geheim-Variablen. Hier macht sich eine sprechende Benennung wieder einmal bezahlt.

```
# Entschlüsselung
print("=========> Entschlüsselung ========>")
laengeGeheimText=len(geheimText)
dechiffKlarText=""
for i in range(laengeGeheimText):
    geheimSymbol=geheimText[i]
    pos=-1
    for j in range(laengeKlarAlpha):
        if geheimSymbol==geheimAlpha[j]:
            pos=j
            break
    klarSymbol=klarAlpha[pos]
    if pos>=0:
        dechiffKlarText+=klarSymbol
    else:
        dechiffKlarText+=geheimSymbol

# Ausgabe
```

Aufgaben:

1. ***Übernehmen Sie das Programm und die ergänzenden Programm-Abschnitte! Testen Sie das Programm mit verschieden Eingaben!***
2. ***Erweitern Sie das Programm um die Möglichkeit weitere Klartexte einzugeben! Ein Abbruch soll mit der Eingabe eines leeren Textes erfolgen!***
3. ***Verändern Sie die Ausgabe "`======> Verschlüsselung …`" so, dass für jedes chiffrierte Symbol ein Gleichheitszeichen angezeigt wird! Übernehmen Sie dieses dann auch für die Entschlüsselung"***

Natürlich gibt es weitaus schönere und effektivere Daten-Strukturen für die Alphabete.

So kann man zwei einhache Strings benutzen und dann durch sie durch interieren.

```
klarAlpha="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
geheimAlpha="nopqrstuvwxyzabcdefghijklm"
…
```

Diese Variante erscheint mir z.B. sehr gut für die flexible Erzeugung von Klar- und Geheim-Text-Symbolen zu sein.

Eine weitere Möglichkeit ist die Verwendung von einer strukturierten / geschachtelten Liste aus Symbol-Paaren.

```
rot13=[["A","n"], ["B","o"], ["C","p"], …
       …
      ]
…
```

Auch Tupel in Form von Dictonary's sind denkbar. Besonders wenn man Alphabete aus einer (JSON-)Datei einlesen möchte, spricht einiges für diese Variante.

```
rot13=[["A","n"], ["B","o"], ["C","p"], …
       …
      ]
…
```

Es geht aber auch ohne Vordefinition der Alphabete. Man kann ja auch die ASCII-Tabelle der Rechner selbst nutzen.
Natürlich muss man dann die neuen ASCII-Symbole immer berechnen. Der Algorithmus ändert sich also entscheident.

Klar-Alphabet			
Symbol	**ASCII**	**Symbol**	**ASCII**
A	065	a	097
B	066	b	098
C	067	c	099
…		…	…
L	076	l	108
M	077	m	109
N	078	n	110
O	079	o	111
…		…	
X	088	x	120
Y	089	y	121
Z	090	z	122

Geheim-Alph..	
Symbol	**ASCII**
n	110
o	111
p	112
…	
y	121
z	122
a	097
b	098
…	…
k	107
l	108
m	109

Aufgaben:

1. *Entscheiden Sie sich für eine neue Art der Alphabet-Darstellung bzw. die "Alphabet"-frei Version und erstellen Sie ein neues ROT13-Programm!*
2. *Drucken Sie Ihr Programm aus und veröffentlichen Sie es für eine Diskussion an der Tafel oder einem schwarzen Brett od.ä.!*
3. *Ein Mitschüler vertritt die Auffassung, man durch mehrfache Anwendung von Verschlüsselungen aufeinander die Sicherheit deutlich erhöhen kann. Auch beim ROT13-verfahren soll dies so sein. Setzen Sie sich mit dieser These auseinander!*

für die gehobene Anspruchsebene:

4. *Verändern Sie eine Programm-Version so, dass eine Text-Datei mit einem längeren Klartext eingelesen und angezeigt werden kann! Dieser Text soll dann verschlüsselt und danach wieder entschlüsselt angezeigt werden!*

In der Registry sollen in bestimmten Schlüsseln die Verläufe (des Internet-Browsing) gespeichert sein, die mit ROT13 verschlüsselt sind und wohl auch nicht gelöscht werden, wenn man den Verlauf löscht!
`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\ [Unterverzeichnisse]`
Dieses Logging lässt sich abschalten, wenn man in dem Verzeichnis einen neuen Schlüssel (DWORD) "NoLog" anlegt und diesem den Wert auf 1 setzt. Mit einem weiteren Schlüssel "NoEncrypt" mit dem Wert 1 kann die "Verschlüsselung" ausgeschaltet werden (mit 0 eben wieder einegschaltet)

8.19.1.x.1. ROT13 mit einer Funktion

Die zwei praktisch identischen Algorithmen für die Ver- und Entschlüsselung sind natürlich ein Dorn im Auge eine ("faulen") Programmierer's. Findet man irgendwann einen Fehler oder will man den Algorithmus berändern, dann muss man immer an zwei Stellen im Programm korrigieren. Erfahrungs.gemäß geht das schief. Irgend eine Stelle vergißt man oder ändert diese anders. Da sind dann Folge-Probleme schon vorprogrammiert.

So etwas schreit ja förmlich nach der Benutzung einer Funktion, die den einen Text in den anderen umwandelt. Was dabei Klar- und was Geheim-Text ist, ist ja egal, weil das Verfahren so schön symmetisch ist.

Wir bleiben hier mal bei der oben besprochenen Form der Alphabet-Darstellung in zwei Listen. Die Variabalen, die sich auf den Klartext bezogen, werden in der Funktion nun in **rein**-gehende Variablen umbenannt. Dementsprechend die geheim-Variablen auf **raus**.

```
def rot13(textRein):
    laengeTextRein=len(textRein)
    textRaus=""
    for i in range(laengeTextRein):
        reinSymbol=textRein[i]
        pos=-1
        for j in range(laengeKlarAlpha):
            if reinSymbol==klarAlpha[j]:
                pos=j
                break
        geheimSymbol=geheimAlpha[pos]
        if pos>=0:
            textRaus+=geheimSymbol
        else:
            textRaus+=reinSymbol
    return textRaus
```

Nun müssen wir aber auch noch beachten, dass wir aus kosmetischen Gründen die Klar- und geheim-Texte mit anderen Buchstaben versehen haben.

Das Hauptprogramm verkürzt sich nun natürlich deutlich durch die Funktions-Aufrufe.

Das Programm wird so auch deutlich übersichtlicher und verständlicher.

Änderungen und Erweiterungen können wir nun auch sehr gut vornehmen.

```
…
abbruch=False
while not abbruch:
# Eingabe
    klarText=input("Klartext  : ")

    if klarText>"":
        klarText=klarText.upper()
        print("KLARTEXT  : ",klarText)

    # Verschlüsselung
        print("======> Verschlüsselung =====>")
        geheimText=rot13(klarText)
        print("ROT13-Fkt.: ",geheimText)

        geheimText=geheimText.upper()

    # Entschlüsselung
        print("======> Entschlüsselung =====>")

    # Ausgabe
        print("ROT13-Fkt.: ",
              rot13(geheimText))
        print("-----------------------------")
        print()
    else:
        abbruch=True

print("Programm-Ende")
```

Aufgaben:

1. ***Der Aufbau der Verschlüsselungs-Zeile mit den Gleichheits-Zeichen entsprechend der umgewandelten Symbole hat einem Kunden sehr gut gefallen. Bekommen Sie das auch mit rot13-Funktion hin? Realisieren Sie die Funktion entsprechend ODER begründen Sie, warum das so nicht geht!***
2. ***Wandeln Sie Ihr 2. ROT13-Programm (mit der geänderten Daten-Struktur für die Alphabete bzw. mit dem geänderten Algorithmus) in ein Programm mit einer passenden rot13-Funktion um!***

8.19.1.x.2. Häufigkeits-Analyse

Alle einfachen CÄSAR-Chiffren – und ganz besonders die ROT13-Chiffre – sind für Analysen der Buchstaben-Häufigkeit empfindlich. Wir wollen die Buchstaben-Häufigkeit vor allem dazu benutzen, um verschiedene Verfahren miteinander zu vergleichen und zu bewerten.
Hier werden wir eine Funktion erstellen, die sich auf die Zählung und Anzeige der Symbole beschränkt. Für vergleichende Zwecke müsste man sonst vielleicht auch die Ergebnisse wieder zurückgeben.

```
…
def symbolHaeufigkeit(alphabet, analyseText):
    print(".. Häufigkeits-Analyse ..")
    anzahlSymbole=len(alphabet)
    haeufigkeit=[]
    for i in range(anzahlSymbole):
        haeufigkeit.append(0)
    laengeText=len(analyseText)
    for i in range(anzahlSymbole):
        aktSymbol=alphabet[i]
        for j in range(laengeText):
            if analyseText[j]==aktSymbol:
               haeufigkeit[i]+=1
    for i in range(anzahlSymbole):
        print("   ",format(alphabet[i],"2s"),
              format(haeufigkeit[i],"3d"),
              fomat(haeufigkeit[i]/laengeText*100,"6.2f"),"%")
    #return
…
```

In der obigen Funktion wird Alphabet-bezogen gearbeitet. Das Ergebnis soll in der Liste hauefigkeit gespeichert werden. Für jedes Symbol aus dem Alphabet wird zuerst einmal eine Null als Anfangs-Wert eingespeichert.
Danach wird wieder für jedes Symbol der Text nach allen Vorkommen durchsucht und die Häufigkeit inkrementiert.
Zum Schluß wird die Häufigkeit für jedes Symbol mit Anzahl und prozentualem Anteil ausgegeben.

Aufgaben:

1. ***Vereinfachen Sie die Häufigkeits-Analyse dahingehend, dass nur noch eine Schleife (`for i in range(anzahlSymbole):`) benutzt wird!***
2. ***Erweitern Sie die Analyse um die Erfassung der Nicht-Alphabet-Symbole und einer nahtloschen Ausgabe als "???"!***
3. ***Da die Zeilen für die Symbole nicht wirklich ausgenutzt werden, möchte der Kunde eine Ausgabe in mehreren Spalten, wobei die Spalten-Anzahl variabel gehalten werden soll!***

Hier eine mögliche Umsetzung einzelner Aspekte in einer erweiterten Funktion.

```
def symbolHaeufigkeit(alphabet, analyseText):
    print(".. Häufigkeits-Analyse ..")
    anzahlSymbole=len(alphabet)
    haeufigkeit=[]
    anzahlGefunden=0
    laengeText=len(analyseText)
    for i in range(anzahlSymbole):
        haeufigkeit.append(0)
        aktSymbol=alphabet[i]
        for j in range(laengeText):
            if analyseText[j]==aktSymbol:
               haeufigkeit[i]+=1
               anzahlGefunden+=1
    spalten=3
    aktSpalte=0
    for i in range(anzahlSymbole):
        print((8-spalten)*" ",format(alphabet[i],"2s"),
              format(haeufigkeit[i],"3d"),
              format(haeufigkeit[i]/laengeText*100,"6.2f"),"%",end="")
        aktSpalte+=1
        if aktSpalte==spalten:
            aktSpalte=0
            print()
    print((7-spalten)*" ",format("???","3s"),
          format(laengeText-anzahlGefunden,"3d"),
          format((laengeText-anzahlGefunden)/laengeText*100,"6.2f"),"%")
    #return
```

Aufgaben:

1. ***Erweitern Sie die Buchstaben-Häufigkeits-Analyse um kleine Histogramme (in Balken-Form)! Für jeweils gerundete 10 % könnte z.B. eine Raute (#) und für 5 % ein senkrechter Strich (\|) gesetzt werden.***
2. ***Verändern Sie die Funktion so, dass beim Funktions-Aufruf auch mit angegeben werden kann, in wievielen Spalten die Ausgabe erfolgen soll!***
3.

8.19.1.x. Umsetzung der CÄSAR-Verschlüsselung

Mit den Kenntnissen aus der Umsetzung der ROT13-Verschlüsselung können wir nun auch effektiv eine universelle CÄSAR-Ver- und Entschlüsselung programmieren.
Will man das Konzept der zwei Alphabete, wie vorne beschrieben, weiter benutzen. Dann muss man sich nach der Festlegung der Verschiebung das Geheim-Alphabet (neu) zusammenstellen. Dann kann man viele Funktionen mit wenigen Veränderungen übernehmen. Dafür spricht z.B., dass man diese Funktionen z.B. für CÄSAR13 ja schon getestet hat. Wie wir später bei der Umsetzung einer CÄSAR-Verschlüsselung mit einem Schlüssel noch sehen werden, ist diese zuerst etwas altbacken wirkende Methode, dann doch wieder sehr praktisch.
Aus meiner Sicht spricht hier aber mehr für eine flexible Umsetzung auf der Basis der ASCII-Zeichen.
Geht man von einem Ring-förmigen Alphabet aus, dann beschreiben die Funktionen:

$chiffriert_S(K) = (K + S) \bmod 26 = G$	K .. Position Klar-Alphabet-Symbol S .. Schlüssel-(Nummer) / Verschiebung
$dechiffriert_S(G) = (G - S) \bmod 26 = K$	G .. Position Geheim-Alphabet-Symbol

das grundsätzliche Vorgehen.
Bei den ASCII-Zeichen haben wir aber kein geschlossenes Alphabet. Leider sind auch die Kleinbuchstaben nicht direkt an die Groß-Buchstaben angeschlossen, was uns hier super helfen würde.
So bleibt nur eine Entscheidungs-bezogene Umsetzung entsprechend der Lage des / der Buchstaben zum Geheim-Buchstaben, welcher der CÄSAR-Chiffre entspricht.

```
# CÄSAR-Verschlüsselung
# 01.2020

# Eingabe Klartext
klarText=input("Klartext:  ")
verschiebung=eval(input("CÄSAR-Code (-26 .. 26): "))

# Vorbereitung
klarText=klarText.upper()
print("KLARTEXT:  ",klarText)
obenGeheimCodeGrenze=90-verschiebung
geheimText=""

# Verschlüsselung
laengeKlarText=len(klarText)
for i in range(laengeKlarText):
    aktKlarSymbol=ord(klarText[i])   #ASCII-Code
    if aktKlarSymbol>=65 and aktKlarSymbol<=90: # Großbuchstabe
        aktGeheimSymbol=aktKlarSymbol+verschiebung
        if aktGeheimSymbol>90:  # zu großer ASCII-Code
            aktGeheimSymbol-=26
        if aktGeheimSymbol<65:  # zu kleiner ASCII-Code
            aktGeheimSymbol+=26
        geheimText+=chr(aktGeheimSymbol) # Symbol aus Code
    else: # Nicht-Alphabet-Symbol
        geheimText+=chr(aktKlarSymbol)

print("Geheimtext: ",geheimText.lower())
```

Nach der Prüfung, ob es sich um ein Klartext-Symbol (also Großbuchstabe) handelt, wird die Verschiebung im ASCII-Code vorgenommen. Dabei können auch ASCII-Code's jenseits der Großbuchstaben entstehen. Hier korrigienen wir dann um die Alpahbet-Länge (hier 26).

Der aufmerksame Leser wird gleich bemerkt haben, dass ich dieses Mal nicht in die Kleinbuchstaben chiffriert habe. Da steckt einfach schon der Hintergedanke drin, später eine universelle Funktion aus dem Prototypen zu machen. Die Darstellung mit Klein- und Groß-Buchstaben ist ja nur Kosmetik und die möchte ich den Python-String-Funktionen überlassen.

Aufgaben:

1. ***Ergänzen Sie das obige Programm um eine Wiederholungs-Schleife, die solange chiffriert, bis ein Leertext eingegeben wird!***
2. ***Verändern Sie das Programm so, dass eine direkte Chiffrierung in die Kleinbuchstaben erfolgt!***
3. ***Planen Sie eine universelle Funktion für die CÄSAR-Chiffrierung und –Dechiffrierung! Welche Parameter braucht eine solche Funktion?***
4. ***Realisieren Sie eine universelle CÄSAR-Funktion!***
5. ***Erweitern Sie das Programm nun noch um die Dechiffrierung und eine Buchstaben-Häufigkeits-Analyse!***

8.19.1.x. moderne CÄSAR-Verschlüsselung mit Schlüssel

Natürlich ist der Begriff modern nicht wirklich Ernst gemeint. Durch das große Leistungs-Potential von Computern und vielen sehr cleveren Analyse-Methoden ist die CÄSAR-Chiffrierung nicht mehr Stand der Zeit. Aber mit ein paar Tricks kann man noch so einiges aus dieser "einfachen" Verschlüsselung herausholen.

Beispiel: CÄSAR7 mit Schlüsselwort BUCHLISTE

Symbole lfd. Nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Klaralphabet	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Geheimalph.	**r**	**v**	**w**	**x**	**y**	**z**	**b**	**u**	**c**	**h**	**l**	**i**	**s**	**t**	**e**	**a**	**d**	**f**	**g**	**j**	**k**	**m**	**n**	**o**	**p**	**q**

Beispiel: CÄSAR7 auf ein Zufalls-Alphabet

Symbole lfd. Nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Klaralphabet	**K**	**T**	**G**	**E**	**P**	**S**	**W**	**Y**	**J**	**Q**	**H**	**Z**	**I**	**A**	**V**	**O**	**R**	**C**	**M**	**N**	**U**	**X**	**B**	**D**	**F**	**L**
Geheimalph.	**u**	**x**	**b**	**d**	**f**	**l**	**k**	**t**	**g**	**e**	**p**	**s**	**w**	**y**	**j**	**q**	**h**	**z**	**i**	**a**	**v**	**o**	**r**	**c**	**m**	**n**

8.19.1.x. POLYBIOS-Verschlüsselung

auch unter Polybius zu finden
beschrieben vom griechischen Geschichtsschreiber POLYBIOS VON MEGALOPOLIS etwa um 200 bis 120 v.u.Z.

A	B	C	D	E	1
F	G	H	I	K	2
L	M	N	O	P	3
Q	R	S	T	U	4
V	W	X	Y	Z	5
1	2	3	4	5	

Basis ist die sogenannte Polybios-Matrix (Polybios-Quadrat), in der die Klar-Buchstaben Zeilen-weise notiert sind
Die Zeilen und Spalten werden durchnummeriert.
Die Zeilen- und Spalten-Nummern (Buchstaben-Koordinaten) werden zur Substitution genutzt.
Nehmen wir an, unser Klartext lautet:

SEHR GEHEIM

A	B	C	D	E	1
F	G	H	I	K	2
L	M	N	O	P	3
Q	R	**S**	T	U	**4**
V	W	X	Y	Z	5
1	2	**3**	4	5	

Praktisch wird nun jeder zu verschlüsselnde Buchstabe in der Matrix gesucht und dann zuerst immer die Zeilen- und dann die Spalten-Nummer notiert.
Aus dem **S** wird so **43** usw. usf.
Die Nummer konnten dann z.B. über auf dem Burg-Türmen oder –Mauern angeordneten Fackeln signalisiert werden.
Die Empfänger benutzten die gleiche Tabelle und praktisch das gleiche Verfahren (nur umgekehrt), um den Klartext aus den Geheimzeichen zu dechiffrieren.
Aus den Koordinaten **43** erhalten wir dann den entschlüsselten Buchstaben **S**.

A	B	C	D	E	1
F	G	H	I	K	2
L	M	N	O	P	3
Q	R	**S**	T	U	**4**
V	W	X	Y	Z	5
1	2	**3**	4	5	

In Python können wir für die POLYBIOS-Matrix ein mehrdimensionales Feld (Array, Vektor) nutzen (→ 6.6. Vektoren, Felder und Tabellen).

```
polybios=array(["A","B","C","D","E"],
               ["F", …],
               …)
```

Aufgaben:

1. *Verschlüsseln Sie den obigen Text bis zum Ende!*
2. *Überlegen Sie sich, wie das Verfahren verbessert werden! Machen Sie Vorschläge und erklären Sie, welche Veränderungen sich ergeben würden!*
3. *Setzen Sie die POLYBIOS-Verschlüsselung in ein Python-Programm um! Der zu verschlüsselnde Text soll als Eingabe in das Programm einfließen! Als Basis-Quadrat verwenden wir eine 6x6-Matrix mit allen Buchstaben und den Ziffern.*

Ein der praktischen Umsetzungen erfolgte als Klopf-Code in Gefängnissen, um z.B. über Rohrleitungen oder Wände Nachrichten zu übertragen.
Beim bifid-Verfahren (→) wird an eine POLYBIOS-Verschlüsselung noch eine Transposition angehängt, um die Koordinaten zu trennen.
Eine noch weitere Verbesserung erfolgte durch das ADFGX-Verfahren (→). Dieses wurde noch bis in den 1. Weltkrieg hinein verwendet.

8.19.1.x. VIGENÈRE-Verschlüsselung

Das große Problem der einfachen Substitutions-Verfahren ist immer die mögliche Krypto-Analyse über die Buchstaben-Häufigkeit.
Johannes TRITHEMIUS (1462 – 1516) erstellte für sein Verschlüsselungs-Verfahren eine sogeannte Transpositions-Tabelle – diese nannte er **Recta transpositionis tabula** oder kurz auch **Tabula recta**. In der originalen Tabelle fehlen die Buchstaben **j** und **v**, da im Mittelalter in der deutschen Sprachen **u** und **v** sowie **i** und **j** nicht unterschieden wurden. Wir nehmen hier eine an unser heutiges Alphabet angepasste Tabelle:

		G	**E**	**H**	**E**	**I**	**M**	**B**	**L**	**E**	**I**	**B**	**T**	**G**	**E**	**H**	**E**	**I**	**M**	X	X	X	A	B	C	D
A	1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	26	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

h	**g**	**k**	**i**	**n**	**s**	**i**	**t**	**n**	**s**	**m**	**f**	**t**	**s**	**w**	**u**	**z**	**e**	q	r	s	w	y	a	c

Man kann gut erkennen, dass ein Buchstabe jedes Mal ein neues Geheim-Zeichen bekommt. Obwohl wir fünfmal ein E im Klartext hatten, ergibt sich jedes Mal ein anderes eheim-Symbol. Anders herum kann man aus dem mehrfachen Auftreten eines Geheim-Symbol's nicht auf den Klartext-Buchstaben zurückschließen. Wir haben im Geheimtext z.B. zweimal ein i. Jedes Mal war es aber ein anderer Klartext-Buchstabe, der verschlüsselt wurde.
Im Prinzip benutzte TRITHEMIUS als Schlüssel den Klartext. Mit anderen Worten: er verschlüsselte einen Text mit sich selbst.
Daraus ergibt sich ein Problem: ein Buchstabe wird niemals durch sich selbst kodiert. Dies bietet eine Chance, den Code zu knacken.
Die Verwendung von Füllzeichen ist bei TRITHEMIUS eher ungünstig. Benutzt man das gleiche Füllzeichen (s. oben: XXX), dann ergibt sich eine alphabetische Symbolfolge. Die Verwendung von fortlaufenden Buchstaben (s. oben: ABCD) erzeugt auch eine charakteristische (springende) Symbol-Folge.
Der Franzose Blaise DE VIGINÉRE (sprich: de wischineer,) entwickelte eine ähnliche Chiffre. Er ordnete den Buchstaben in Abhängigkeit von ihrer Position im Klartext unterschiedliche Chiffren zu. Nehmen wir z.B. das Schlüsselwort: **geheim**, dann wird der erste Buchstabe des

Klartextes mit CÄSAR**7** verschlüsselt, weil **g** an der **7**. Position im Alphabet steht. Der zweite Buchstabe wird dann mit CÄSAR5 (e ist an der 5. Position) usw. verschlüsselt. Am Ende des Schlüsselwortes beginnt man wieder von vorn.

		K	**R**	**Y**	**P**	**T**	**O**	**G**	**A**	**F**	**I**	**I**	**S**	**T**	**S**	**C**	**H**	**O**	**N**	**T**	**O**	**L**	**L**	X	X	X
A	1	**G**	**E**	**H**	**E**	**I**	**M**	G	E	H	E	I	M	G	E	H	E	I	M	G	E	H	E	I	M	G
B	2	H	F	I	F	J	N	H	F	I	F	J	N	H	F	I	F	J	N	H	F	I	F	J	N	H
C	3	I	G	J	G	K	O	I	G	J	G	K	O	I	G	J	G	K	O	I	G	J	G	K	O	I
D	4	J	H	K	H	L	P	J	H	K	H	L	P	J	H	K	H	L	P	J	H	K	H	L	P	J
E	5	K	I	L	I	M	Q	K	I	L	I	M	Q	K	I	L	I	M	Q	K	I	L	I	M	Q	K
F	6	L	J	M	J	N	R	L	J	M	J	N	R	L	J	M	J	N	R	L	J	M	J	N	R	L
G	7	M	K	N	K	O	S	M	K	N	K	O	S	M	K	N	K	O	S	M	K	N	K	O	S	M
H	8	N	L	O	L	P	T	N	L	O	L	P	T	N	L	O	L	P	T	N	L	O	L	P	T	N
I	9	O	M	P	M	Q	U	O	M	P	M	Q	U	O	M	P	M	Q	U	O	M	P	M	Q	U	O
J	10	P	N	Q	N	R	V	P	N	Q	N	R	V	P	N	Q	N	R	V	P	N	Q	N	R	V	P
K	11	Q	O	R	O	S	W	Q	O	R	O	S	W	Q	O	R	O	S	W	Q	O	R	O	S	W	Q
L	12	R	P	S	P	T	X	R	P	S	P	T	X	R	P	S	P	T	X	R	P	S	P	T	X	R
M	13	S	Q	T	Q	U	Y	S	Q	T	Q	U	Y	S	Q	T	Q	U	Y	S	Q	T	Q	U	Y	S
N	14	T	R	U	R	V	Z	T	R	U	R	V	Z	T	R	U	R	V	Z	T	R	U	R	V	Z	T
O	15	U	S	V	S	W	A	U	S	V	S	W	A	U	S	V	S	W	A	U	S	V	S	W	A	U
P	16	V	T	W	T	X	B	V	T	W	T	X	B	V	T	W	T	X	B	V	T	W	T	X	B	V
Q	17	W	U	X	U	Y	C	W	U	X	U	Y	C	W	U	X	U	Y	C	W	U	X	U	Y	C	W
R	18	X	V	Y	V	Z	D	X	V	Y	V	Z	D	X	V	Y	V	Z	D	X	V	Y	V	Z	D	X
S	19	Y	W	Z	W	A	E	Y	W	Z	W	A	E	Y	W	Z	W	A	E	Y	W	Z	W	A	E	Y
T	20	Z	X	A	X	B	F	Z	X	A	X	B	F	Z	X	A	X	B	F	Z	X	A	X	B	F	Z
U	21	A	Y	B	Y	C	G	A	Y	B	Y	C	G	A	Y	B	Y	C	G	A	Y	B	Y	C	G	A
V	22	B	Z	C	Z	D	H	B	Z	C	Z	D	H	B	Z	C	Z	D	H	B	Z	C	Z	D	H	B
W	23	C	A	D	A	E	I	C	A	D	A	E	I	C	A	D	A	E	I	C	A	D	A	E	I	C
X	24	D	B	E	B	F	J	D	B	E	B	F	J	D	B	E	B	F	J	D	B	E	B	F	J	D
Y	25	E	C	F	C	G	K	E	C	F	C	G	K	E	C	F	C	G	K	E	C	F	C	G	K	E
Z	26	F	D	G	D	H	L	F	D	G	D	H	L	F	D	G	D	H	L	F	D	G	D	H	L	F
		q	**v**	**f**	**t**	**b**	**a**	**m**	**e**	**m**	**m**	**q**	**e**	**z**	**w**	**j**	**l**	**w**	**z**	**z**	**s**	**s**	**p**	**f**	**j**	**d**

Die Stärke dieses Verfahrens wird schon bei den Füllzeichen am Ende sichtbar. Kein X wurde gleichartig oder mit einer Buchstabenfolge verschlüsselt. Noch besser wäre es natürlich, ganz auf die Füllzeichen zu verzichten, da sie ein guter Angriffs-Punkt für eine Krypto-Analyse sind. Wenn man weiss, dass am Ende sehr wahrscheinlich Xe stehen, dann kann bei genügend Geheimtexten das Passwort teilweise geknackt werden.
Heute wissen wir, wenn man einen zum Klartext gleichlangen Schlüssel verwendet und diesen nur ein einziges Mal benutzt, dann ist die VIGINÉRE-Chiffre unknackbar. Außer natürlich man versucht es mit einem Brute-Force-Angriff.
Man braucht also auch heute keine komplizierte Technik oder gar Computer, um absolut sicher Texte zu verschlüsseln. Das einzige Problem ist der Transport der Schlüssel und die Absprache, welcher Schüssel genau benutzt werden soll.
Werden allerdings kürze Schlüsselwörter benutzt, dann kann der Geheimtext ev. entschlüsselt werden. Dabei ermittelt man zuerst mit dem KASISKI-Test die wahrscheinliche Schlüssellänge. Dann zerlegt man den Text in die Teile, die mit dem gleichen Schlüssel-Zeichen codiert wurden. Sie werden einer Häufigkeits-Analyse unterzogen. Ab hier ist es dann nur noch Rechen- oder Such-Aufwand. Gute Code-Knacker erschließen daneben noch das verwendeten Schüsselwort.

(!Aufgabe für die händische Arbeit!)

Aufgaben:

1. ***Verschlüssele den folgenden Text mittels VIGENÉRE-Verfahren und dem Schlüsselwort "DAMENSCHUH"!***

   Mein Geheimnis ist: Ich mag gerne Tee.

2. ***Denke Dir nun ein neues Schlüsselwort mit mindestens 8 Zeichen aus und verschlüssele damit einen Text von maximal 25 Zeichen! Die nicht benötigten Zeichen werden mit X aufgefüllt.***
3. ***Gebe den Geheimtext und den Schlüssel an Deinen Nachbarn weiter! Dechiffriere den Geheimtext Deines Nachbarn!***

für Experten und zum Knobeln:

4. ***Der folgende Text wurde mittels VIGENÉRE-Verfahren verschlüsselt. Die letzten – nicht gebrauchten – Zeichen wurden mit X aufgefüllt. Wie lautet das Passwort und wie der Klartext?***

Verbesserungen

Giovan Battista BELLASO (~ 1505 ~ 1568/81) benutzte statt der klassischen (sortierten Alphabete z.T. gewürfelte Symbol-Listen (1555). So z.B. für die Buchstaben A und R die folgende Liste.

A R	→	r	m	d	a	c	n	e	u	p	s	b	t	d	f	g	e	h	l	x	o	y	z

(!Aufgabe für die händische Arbeit!)

Aufgaben:

1. ***Erstelle Dir eine eigene Liste von 5 gewürfelten deutschen Alphabeten! Ordne Sie den möglichen Schlüssel-Buchstaben zu, so dass eine private Verschlüsselungs-Tabelle entsteht!***
2. ***Verschlüssele nun mit Deiner Tabelle und einem Schlüsselwort einen kurzen Text!***
3. ***Tausche den Geheimtext und das Schlüsselwort auf zwei verschiedenen Wegen (schriftlich, mündlich, per eMail, …) mit einem Kursteilnehmer!***
4. ***Entschlüssele die getauschte Nachricht!***

8.19.1.x.y. Krypto-Analyse der VIGENÈRE-Verschlüsselung

praktisch Positions-bezogene CÄSAR-Verschlüsselung
das sich die Verschiebung mit der Länge des Schlüssel's wiederholt kann man auf gleiche Verschlüsselung für gleiche Buchstaben-Folgen setzen
Zuerst versucht man die Länge des Schlüssel's zu ermitteln

KASISKI-Test

Suche nach gleichen Buchstaben-Folgen und Ermittlung des Abstände zwischen den Wiederholungen
Schlüssel-Länge könnte nun einer dieser Abstand oder einer der (gemeinsamen) Teiler sein (→ Prim-Faktoren-Zerlegung)

je länger die untersuchten Buchstaben-Folgen sind, umso größer ist die Wahrscheinlichkeit für die richtige Schlüssel-Länge

Analyse mittels Auto-Korrelation
Zuerst wird der Text 2x hintereinander notiert und gegenseitig verschoben (*? wie*)
für jede Verschiebung die Anzahl gleicher Buchstaben ermitteln
Suche nach Verschiebungen mit möglichst vielen Übereinstimmungen

Nun für jede Buchstaben-Gruppe des Schlüssel's (?woher bekannt) die Buchstaben-Häufigkeit ermitteln und ev. ein Diagramm erstellen
typische Häufigkeits-Analyse → Prüfen ob Verschiebung der ermittelten Häufigkeit zur typischen Verteilung in der Sprache stimmt

8.19.1.x. bifid-Verschlüsselung

Bei der bifid-Chiffrierung wird die klassische POLYBIOS-Chiffrierung (Substitution) durch eine Faktionierung und eine Rück-Chiffrierung ergänzt.
Dazu werden die Koordinaten nicht hintereinander weg geschrieben, sondern in zwei Zeilen:
Nehmen wir an, unser Klartext lautet:

A	B	C	D	E	1
F	G	H	I	K	2
L	M	N	O	P	3
Q	R	**S**	T	U	**4**
V	W	X	Y	Z	5
1	2	**3**	4	5	

SEHR GEHEIM

Praktisch wird nun jeder zu verschlüsselnde Buchstabe in der Matrix gesucht und dann zuerst immer die Zeilen- und dann die Spalten-Nummer notiert.
Aus dem **S** wird so:

4
3

usw. usf.
Die beiden Zeilen:

4124 212123
3532 253542

wird dann die Symbolfolge:

41242121233532253542

Mit dieser Zifferfolge wird nun eine Rück-Verschlüsselung vorgenommen. D.h. die ersten zwei Ziffern sind die Koordninaten

41 24 21 21 23 35 32 25 35 42

für den ersten Geheim-Buchstaben:

Daraus ergibt sich ein Geheimtext, der auch Häufigkeits-Analysen stand hält:

qiffhpmkpr

a	b	c	d	e	1
f	g	h	i	k	2
l	m	n	o	p	3
q	r	s	t	u	4
v	w	x	y	z	5
1	2	3	4	5	

Wie in symmetrischen verfahren üblich läßt sich die Entschlüsselung durch das Umkehren des Verfahrens erreichen.
Zuerst wandeln wir den Geheimtext wieder in die Koordinaten um:

41 24 21 21 23 35 32 25 35 42

Dann wird die reine Ziffernkette

41242121233532253542

in der Mitte getrennt, um de Fraktionierung rückgängig zu machen:

4124 212123
3532 253542

Die Koordinaten oben und unten in den zwei Zeilen sind nun wieder die Basis für die Rück-Verschlüsselung in den Klartext.
Aus **43** wird so wieder der ursprüngliche Klartext-Buchstabe **S**.
In dieser Form gehen wir die nächsten Paare durch und erhalten den vollständigen Klartext zurück:

A	B	C	D	E	1
F	G	H	I	K	2
L	M	N	O	P	3
Q	R	S	T	U	4
V	W	X	Y	Z	5
1	2	3	4	5	

SEHR GEHEIM

Bei der Umsetzung in Python sollten wir jetzt deutlich planvoller vorgehen.
Da wir nun für Chiffrieren und Dechiffrieren immer jeweils eine POLYBIOS-Chiffrierung und –Dechiffrierung brauchen, ist die Nutzung von Funktion fast nicht mehr zu umgehen.

```
function chiffPolybios(polybios, zeichen):
    …
    return koord

function dechiffPolybios(koord, polybios):
    …
    return zeichen
```

Aufgaben:

1. *Planen Sie ein Programm zur Chiffrierung und Dechiffrierung nach dem bifed-Verfahren als Grob-Struktogramm!*
2. *Leiten Sie aus dem Grob-Struktogramm ein Funktions-orientiertes Python-Programm ab, in dem Sie dann Schritt-weise die Funktionen mit Leben füllen!*
3.

8.19.1.x. ADFGX-Verschlüsselung

in einer erweiterten 6x6-Form auch ADFGVX
basiert auf der POLYBIOS-Chiffre

der offizielle Name war "Geheimschrift der Funker 1918" oder kurz "GedeFu 18"

der Name ADFGX stammt von den Allierten, die die auffälligen Funksprüche nach den verwendeten Buchstaben charakterisierten
damals natürlich immer manuell durchgeführt
die Auswahl der Buchstaben wurde so gewählt, dass die MORSE-Zeichen sich besonders gut voneinander unterscheiden ließen

Bst.	MORSE-Zeichen
A	·—
D	—··
F	··—·
G	——·
V	···—
X	—··—

entwickelt vom deutschen Nachrichten-Offizier Fritz NEBEL (1891 - 1977)

verwendet wird als Basis das POLYBIOS-Quadrat
wieder ohne das J
auch die Reihenfolge der Buchstaben wird gedreht

Z	Y	X	W	V	a
U	T	S	R	Q	d
P	O	N	M	L	f
K	I	H	G	F	g
E	D	C	B	A	x
a	d	f	g	x	

desweiteren wird ein Schlüsselwort verwendet
als Beispiel hier: **Verschlüsselung**

dieses sollte optimaler-weise schön lang sein und alle Buchstaben nur einfach enthalten
sollten Buchstaben doppelt vorkomm,en werden sie im Einsatz einfach weggelassen
damit bleibt **VERSCHLUNG** übrig
das restliche Alphabet wird dann dahinter geschrieben

V	E	R	S	C	a
H	L	U	N	G	d
Z	Y	X	W	T	f
Q	P	O	M	K	g
I	F	D	B	A	x
a	d	f	g	x	

als Nächstes erfolgt die POLYBIOS-typische Substitution durch Beschriftung der Zeilen und Spalten (Koordinaten in der Matrix)
Soll z.B. der Klartext:

SEHR GEHEIM

verschlüsselt werden, dann wird aus dem **S** der Zwischen-Code **ag**.
Nachdem der Klartext so codiert wurde, erhalten wir:

ag ad da af dx ad da ad xa gg

Um die Angreifbarkeit gegen Häufigkeits-Analyse zu verbessern wird nun noch eine zweite Stufe der Verschlüsselung genutzt.
Dazu wird ein zweites Schlüsselwort (hier: **Krypto**) verwendet. Dieses Mal wird üblicherweise auf das Weglassen doppelter Buchstaben verzichtet. Das Verfahren funktioniert aber auch mit dem Weglassen. Wir verwenden hier jetzt nur ein kurzes Wort, da ja auch unser Zwischtext relativ kurz ist. Übliche Schlüsselwortlängen sind hier 15 bis 22 Zeichen.

Den Schlüsselwort-Buchstaben wird nun ihrer Position im klassischen Alphabet entsprechent eine Reihenfolge zugeordnet. Da im Alphabet das K aus Krypto der erste Buchstabe ist, erhalt es die Spalten-Nummer 1 usw. usf.
In die neue Tabelle wird nun der Zwischen-Code Zeilenweise notiert.
Die fehlenden Zeichen werden ausgefüllt. Hier die Buchstaben des Geheim-Alphabetes in umgekehrter Reihenfolge.

K	R	Y	P	T	O
1	**4**	**6**	**3**	**5**	**2**
a	g	a	d	d	a
a	f	d	x	a	d
d	a	a	d	x	a
g	g	x	g	f	d

Das Erzeugen des zu sendenden Geheimtextes erfolgt nun durch Auslesen der Spalten entsprechend der (aus dem 2. Schlüsselwort) abgeleiteten Reihenfolge.
Also wird zuerst die Spalte K und dann O usw. usf. hintereinander notiert.

aadg adad dxdg gfag daxf adax

In typischer Funker-Manier werden die Buchstaben in Fünfer-Gruppen übertragen, was fehlende oder falsch erkannte Zeichen leichter erkennen läßt.

aadga daddx dggfa gdaxf adax

Auch hier können die fehlenden Buchstaben beliebig ergänzt werden. Z.B. kännten wir noch ein a ranhängen. Damit sendet der Funken dann 25 Zeichen:

aadga daddx dggfa gdaxf adaxa

Zur Dechiffrierung geht man den umgekehrten Weg durch das Verfahren.
Zuerst werden die Fünfer-Gruppen aufgelöst und die Zeilen-Anzahl für die Transpositions-Tabelle aus der Buchstaben-Anzahl und der Schlüsselwort-Länge berechnet. Bei 25 Zeichen Geheimtext und der Schlüsselwort-Länge von 6 Zeichen ergeben sich 4 Zeilen (6 x 4 = 24 < 25).
Somit wird aus dem ungruppierten Geheimtext jetzt einer, der 4er Gruppen enthält:

aadg adad dxdg gfag daxf adax a

Das Schlüsselwort muss jetzt natürlich bekannt sein, damit die richtige Spalten-Reihenfolge ermittelt werden kann.
Die
Dabei bleiben die überzähligen Buchstaben in der letzten Spalte hängen und werden einfach ignoriert.

K	R	Y	P	T	O
1	**4**	**6**	**3**	**5**	**2**
a	g	a	d	d	a
a	f	d	x	a	d
d	a	a	d	x	a
g	g	x	g	f	d
		a			

Im zweiten Schritt werden wieder die Koordinaten rekonstruiert. Dazu wird die Hilfs-Tabelle wieder Zeilen-weise in 2er Gruppen ausgelesen:

ag ad da af dx ad da ad xa gg xg fd

Zum Schluß rekonstruieren wir aus den Koordinaten wieder die ursprünglichen Buchstaben. Aus dem **ag** wird so wieder das **S** usw. usf.

V	E	R	S	C	a
H	L	U	N	G	d
Z	Y	X	W	T	f
Q	P	O	M	K	g
I	F	D	B	A	x
a	d	f	g	x	

gebrochen durch die Krypto-Analyse des fanzösischen Georges PAINVIN (1918)

Aufgaben:

1. *Verändern Sie die Verschlüsselung so, dass mit einem 6x6-Quadrat für alle Buchstaben und Ziffern gearbeitet werden kann! Probieren Sie es einmal mit einem Partner un gegenseitiger Nachrichten-Übertragung aus!*
2. *Entwickeln Sie ein Python-Programm, welches das ADFGVX-Verfahren umsetzt! Der Klartext und die beiden Schlüsselwörter sollen eingebbar sein! Alle Zwischen-Schritte bzw. –Tabellen sollen anzeigbar sein. Im fertigen Programm sollten diese dann abschaltbar sein oder auskommentiert werden!*
3.

8.19.1.x. trifid-Verschlüsselung

von Franzosen Felix DELASTELLE (1840 - 1902) entwickelt
1902 beschrieben

arbeitet mit drei kleineren Polybios-ähnlichen Tabellen
dadurch ergeben sich für jeden Buchstaben 3 Koordinaten (Matrix, Zeile, Spalte)
diese Trigramme
Das Klartext-Alphabet kann hier schon mit einem Schlüsselwort verteilt eingetragen werden. Der Übersichtlichkeit verwe hier ein unverschlüsseltes Alphabet.

Matrix **1**			
	1	**2**	**3**
1	A	B	C
2	D	E	F
3	G	H	I

Matrix **2**			
	1	**2**	**3**
1	J	K	L
2	M	N	O
3	P	Q	R

Matrix **3**			
	1	**2**	**3**
1	S	T	U
2	V	W	X
3	Y	Z	+

Durch die drei 3x3 Tabellen kommen wir nun auch auf 27 mögliche Symbole. Da verwenden wir das vollständige Alphabet und ein Sonderzeichen, was z.B. als Leerzeichen dienen könnte.
Das Vorgehen ist wieder äquivalent zur POLYBIOS-Chiffre. Nur erhalten wir außer den beiden Koordinaten auch noch die Matrix-Nummer.
Verschlüsseln wir dieses Mal:

RICHTIG GEHEIM

Nun wird das **R** durch das Trigramm **233** codiert. Dieses wird wieder gleich auf drei Zeilen verteilt:

Matrix **1**			
	1	**2**	**3**
1	A	B	C
2	D	E	F
3	G	H	I

Matrix **2**			
	1	**2**	**3**
1	J	K	L
2	M	N	O
3	P	Q	R

Matrix **3**			
	1	**2**	**3**
1	S	T	U
2	V	W	X
3	Y	Z	+

2
3
3

Mit den anderen Symbolen gehen wir genauso vor. Das Leerzeichen ersetzen wir durch ein **+**. Insgesamt ergibt sich dann:

21113131111212
33131333232132
33322131222331

Als nächstes sollen Blöcke erstellt werden. Üblich sind 5 bis 7 Spalten als ein Block. Wir wählen hier die 5 als Block-Größe. Damit ergibt die folgende Struktur:

21113 13111 1212
33131 33323 2132
33322 13122 2331

Der letzte Block wird durch ein beliebiges Zeichen erweitert. Ich wähle hier das – eigentlich ungüstige – Leerzeichen.

21113 13111 12123
33131 33323 21323
33322 13122 23313

Der nächste Schritt ist spezifisch für das trifed-Verfahren. Die Blöcke werden – jeder für sich – Zeilen-weise in neue Trigramme zerlegt. Im ersten Block habe ich das durch unterschiedliche Farben gekennzeichnet:

21113 13111 12123
33131 33323 21323
33322 13122 23313

Diese neuen Trigramme verteilen wir wieder auf 3 Zeilen:

21313 11331 12133
13132 31312 23321
13332 13232 12233

und verschlüsseln mit den ursprünglichen Tabellen zurück:

jiuiw gczue dqhxu

Matrix 1			
	1	**2**	**3**
1	a	b	c
2	d	e	f
3	g	h	i

Matrix 2			
	1	**2**	**3**
1	j	k	l
2	m	n	o
3	p	q	r

Matrix 3			
	1	**2**	**3**
1	s	t	u
2	v	w	x
3	y	z	+

Dieser Geheimtext hält einer Häufigkeits-Analyse gut stand.

Die Dechiffrierung dreht das Verfahren einfach um.
Zuerst ermitteln wir die Koordinaten der Geheimtext-Buchstaben, sortieren sie innerhalb von Blöcken wieder um und verschlüsseln zurück.

Matrix 1			
	1	**2**	**3**
1	a	b	c
2	d	e	f
3	g	h	i

Matrix 2			
	1	**2**	**3**
1	j	k	l
2	m	n	o
3	p	q	r

Matrix 3			
	1	**2**	**3**
1	s	t	u
2	v	w	x
3	y	z	+

21313 11331 12133
13132 31312 23321
13332 13232 12233

→

21113 13111 12123
33131 33323 21323
33322 13122 23313

→

21113 13111 12123
33131 33323 21323
33322 13122 23313

→

211131311112123
331313332321323
333221312223313

Am Schluß folgt nun die Rück-Verschlüsselung mittels der Matrizen zum Klartext:
Aus dem Trigramm **233** wird so wieder der Klartext-Buchstabe **R**.
Der so rekonstruierte Klartext:

Matrix **1**			
	1	**2**	**3**
1	A	B	C
2	D	E	F
3	G	H	I

Matrix **2**			
	1	**2**	**3**
1	J	K	L
2	M	N	O
3	P	Q	R

Matrix **3**			
	1	**2**	**3**
1	S	T	U
2	V	W	X
3	Y	Z	+

RICHTIG GEHEIM+

unterscheidet sich nur durch zusätzlich angefügte Plus-/Leer-Zeichen.

Für ein Python-Programm könnte man sicher wieder mehrdimensionale Felder benutzen (→ 6.6. Vektoren, Felder und Tabellen).
Hier scheint mir eine Chiffren-Tabelle – bzw. zwei – auf der Basis von Dictonary's vielleicht günstiger.

```
trifedChiff={
    "A": 111,
    "B": 112,
    …
    }

trifedDechiff={
    111: "A",
    112: "B",
    …
    }
```

alternativ mit Tupeln der drei Koordinaten

```
trifedChiff={
    "A": [1,1,1],
```

Eine strukturierte Liste wäre ebenfalls möglich. Hier spart man sich dann auch die doppelte Darstellung in zwei Dictonary's.
Hier sind mehrere Varianten denkbar.

```
trifed={
    {"A",[1,1,1]},
    {"B",[1,1,2]},
    …
    }
```

```
trifed={
    {"A",111},
    {"B",112},
    …
    }
```

```
trifed={
    {"A",1,1,1},
    {"B",1,1,2},
    …
    }
```

Aufgaben:

1. ***Vereinbaren Sie im Kurs ein reichlich langes Schlüsselwort für die Dictory-Struktur!***
2. ***Erstellen Sie nun ein Programm, dass die trifed-Ver- und Entschlüsselung zeigt! Als Block-Größe verwenden wir die 3, um auch Zahlen-Operationen für die Transposition zu ermöglich!***
3.

für das gehobene Anspruchsniveau:

4. ***Setzen Sie das trifed-Verfahren ohne Einschränkungen um! Es sollen sowohl das Schlüsselwort für die drei Matrizen sowie die Block-Größe frei gewählt werden können!***

8.19.1.x. Four-Square-Verschlüsselung

von Franzosen Felix DELASTELLE (1840 - 1902) entwickelt
benutzt als Basis-Alphabet Buchstaben-Paare (Digraphen, Bigramme)
Substitutions-Chiffre
damit werden aus den 26 Monographen (Monogramme) 676 Digraphen, die nun deutlich schwierger durch Häufigkeits-Analysen angreifbar sind
hierfür wären auch sehr lange Texte notwendig

Zur Ver- und Entschlüsselung werden 4 Quadrate (daher der Name) verwendet. Jeweils diagonal sind die Klartext- sowie die Geheimtext-Alphabete notiert. Im Beispiel wird, wie bei DELASTELLE auf das Q verzichtet. Alternativ wäre ein Verzicht auf das J möglich.
Modernere Verfahren nutzen 6x6-Felder. Dann passen auch noch die Ziffern mit hinein.
Zur Erhöhung der Verschlüsselung werden zwei Schlüsselwörter an den Anfang der Geheimtext-Alphabete gesetzt.

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	R	S	T	U
V	W	X	Y	Z

v	e	r	s	c
h	u	l	n	g
a	b	d	f	i
j	k	m	o	p
t	w	x	y	z

f	o	u	r	s
a	e	b	c	d
g	h	i	j	k
l	m	n	p	t
v	w	x	y	z

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	R	S	T	U
V	W	X	Y	Z

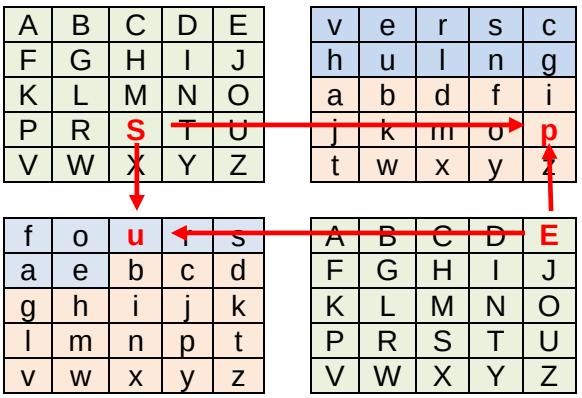
Doppelte Buchstaben werden weggelassen. Hinter den Schlüsselwörtern folgt das restliche Alphabet.
Der zu verschlüsselnde Klartext

z.B.: SEHR GEHEIM

wird zuerst in Digraphen zerlegt:

SE HR GE HE IM

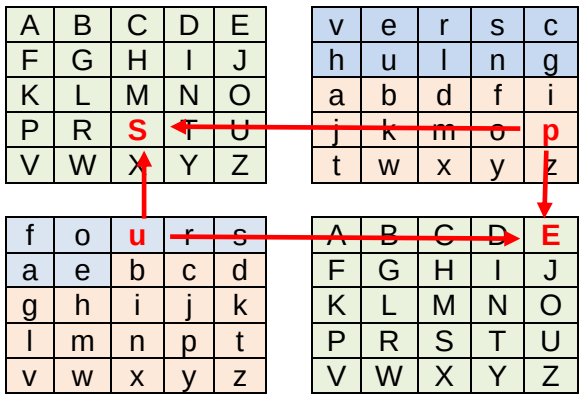
Nun wird der erste Buchstabe im oberen Klartext-Quadrat gesucht und der zweite aus dem unteren. Nun werden Waagerechten und Senkrechten in die Geheimtext-Quadrate gezogen und dort die Geheimtext-Zeichen abgelesen.
Aus **SE** wird so **pu**.
Das Verfahren wird nun Digraph für Diagraph fortgesetzt.



Aufgaben:

1. *Verschlüsseln Sie den Resttext!*
2. *Vereinbaren Sie mt einem Partner aus dem Kurs ein Schlüsselwort-Paar und verschlüsseln Sie damit eine kurze Nachricht!*
3.

Da es sich um ein symmetrisches Verfahren handelt verwenden wir die gleichen Schlüsselwörter und das inverse Verfahren für die Dechiffrierung.
Der empfangene Geheimtext wird wieder in Digraphen zerlegt:



z.B.: **pu** …

und dann der erste Buchstabe im oberen und der zweite im untereb Geheim-Alphabet gesucht. Die Klartext-Buchstaben ergeben sich wieder über die Waagerechten und Senkrechten.
So bekommen wir **SE** aus dem Klartext zurück.

In Python können wir hier mehrdimensionale Felder (Array's, Vektoren) nutzen (→ 6.6. Vektoren, Felder und Tabellen).
Die Geheimtext-Felder können natürlich erst nach Eingabe der Schlüsselwörter belegt werden.

```
klaroben=array(["A","B","C","D","E"],
               ["F", …],
               …)
geheimoben=array(["v","e","r","s","c"],
                 ["", …],
                 …)
…
```

Aufgaben:

1. ***Tauschen Sie die verschlüsselten Nachrichten (vom ersten Aufgabenblock) und entschlüsseln Sie diese!***
2. ***Schreiben Sie ein Programm, das einen einzgebenen Text ver- bzw. entschlüsselt! Die Schlüsselwörter sollen ebenfalls jeweils einzugeben sein! Als Alphabet benutzen wir alle Buchstaben und die Ziffern.***
3.

Aber auch Lösungen über verkettete Liste oder Tupel sind denkbar.

mögliche Listen

```
klaroben={ {"A","B","C","D","E"},
           {"F", …},
            …
         }
geheimoben={ {"v","e","r","s","c"},
             {"", …},
              …
           }
…
```

eine andere Listen-Variante mit weniger Such-Aufwand könnte so aufgebaut sein

```
klaroben={ {"A",0,0}, {"B",0,1}, …

geheimoben= { {"v",0,0}, {"e",0,1}, …
```

8.19.2. asymmetrische Verschlüsselung

interessante Links:

http://inventwithpython.com/hackingciphers.pdf (online-Version des Buches: AL SWEIGART: Hacking Secret Ciphers with Python)

8.20. Code verbessern und optimieren

Laufzeit wird durch viele Faktoren beeinflusst

Leistungs-Parameter des Computers / der ausführenden Maschine

Datenmenge
im Allgemeinen steigt der Berechnings-Aufwand nicht linear, meist expotentiell, selten überexportentiell

Lade-Technik
Laden und Arbeiten mit Daten im Speicher schneller als auf Fetsplatte
lokales Netzwerk ist noch langsamer
am langsamsten ist das Internet

Programmier-Stil / -Paradigmen
unnötige / ungünstige Befehle

Algorithmen / Algorithmen-Analyse
ungünstige Lage von nicht gebrauchten Funktions-Aufrufen inSchleifen
stattdessen speichern des Wertes in einer Variable und dann Nutzung der Variable in der Schleife

8.21. Test-gestütztes Programmieren mit Python

```
1  def test(algorithmus, testBedingung = True):
2
3      assert algorithmus([]) == []
4      assert algorithmus([2]) == [2]
5      assert algorithmus([4,2,2]) == [2,2,4]
6      assert algorithmus([1,2,4,6,7,9]) == [1,2,4,6,7,9]
7      assert algorithmus([9,8,5,5,3,2,1]) == [1,2,3,5,5,8,9]
8      if testBedingung:
9          assert algorithmus([0,-3,5,-9,13] == [-9,-3,0,5,13]
10
11 if __name__ == "__main__":
12     test(bubblesort, False)
13     test(bubblesort)
       test(quicksort)
```

9

9. Python, informatisch – Datenstrukturen, Klassen, Automaten, …

Ähnlich wie Mathematiker leben Informatiker in einer eigenen Modell-Welt. Nicht umsonst gelten sie als Nerds oder Guru's oder was es sonst auch noch für (böse) Bezeichnungen für die liebsten Menschen der Welt gibt (;-).
Beim genauen Hinschauen sind die Modell-Objekte genau so clever, wie die unzähligen mathematischen Operationen und Techniken. Wenn man irgendwelche Dinge am Computer tut, dann werden uns viele benutzte Informatik-Modelle gar nicht so recht bewusst. Kaum einer weiss, dass die Druck-Aufträge in einer Warteschlange verwaltet werden. Der Druck-Auftrag, der zuerst kommt, wird auch zuerst ausgedruckt. Erst wenn einer der Aufträge den Drucker blockiert, dann werden wir uns vielleicht die Warteschlange ansehen (Doppelklicken auf das Drucker-Symbol in der Task-Leiste) und den störenden Auftrag dort löschen.
Mit der Datenstruktur Baum haben wir dagegen alle schon zu tun gehabt, zumindestens, wenn wir einen Computer mehr als einmal praktisch genutzt und Daten gespeichert haben. Die Ordner in einem Laufwerk sind genau so eine Baum-Struktur. Aber auch die Laufwerke selbst sind wieder eine Baum-Struktur. Sie haben die gemeinsame Wurzel "Computer".
Keller und Ringe sind wieder eher verborgene Objekte. Aber auch sie werden für das ordnungsgemäße bzw. gewohnte Funktionieren eines PC gebraucht.

Definition(en): Datenstruktur
Eine Datenstruktur ist der Informatik eine Vereinbarung zur Organisation und Speicherung von Daten.

Definition(en): Datenstruktur
Eine Datenstruktur ist der Informatik eine Vereinbarung zur Organisation und Speicherung von Daten.

Einteilung nach (maximalen) Anzahl der Nachfolger auf ein Objekt möglich

maximal ein Nachfolger:
Liste

maximal zwei Nachfolger:
Binär-Baum

beliebig viele Nachfolger:
(allgemeiner) Baum
Netz

Zuerst werden wir allerdings etwas genauer auf die sogenannten Tupel eingehen. Sie sind keine klassische Datenstruktur oder gar ein Informatiker-Modell. Sie sind eher eine Spezialität von Python.

9.1. Tupel

Tupel sind Aufzählungen von Daten(-Objekten)
können, müssen aber nicht, den gleichen Typ haben

man kann sie als Paare oder Gruppen verstehen
beim Lesen des Quelltextes erscheinen viele Anweisungen kryptisch oder falsch
Anflug von Trick-Programmierung; meist aber elegante und effektive Lösungen, die in anderen Programmiersprachen viele Anweisungen oder eine etwas aufwändigere Programmierung erfordert hätten.
deshalb sollte die Anweisungen mit Tupel gut kommentiert werden

Tupel-Elemente werden in **runde Klammern** notiert
im Prinzip fest definierte und unveränderliche Listen
Zugriff aber – wie üblich bei Indixies – in eckigen Klammern

praktisch fast alle Operationen, wie bei Listen (→ 8.2.3. Listen, die I. – einfache Listen und 9.7. Listen, die II. – objektorientierte Listen) möglich
Tupel lassen sich aber nicht ergänzen oder ändern, nach ihrer Erzeugung sind sie unveränderlich
lassen sich aber aneinanderreihen

ein-elementige Tupel (Singleton) müssen nach dem (ersten) Element noch ein Komma aufweisen!

Können auch geschachtelt sein
geschachtTupel = (1,2,3,(4,5,6,(7,8,9))) hier dreifach geschachtelt: 1. Tupel ist (1,2,3 und ein Tupel (hier der Klammer-Ausdruck), dito für das nächste / innerste Tupel

ein Zerlegen solcher Tupel ist z.B. so möglich:

```
for wert1 in geschachTupel:
    if type(wert1)== int:
        print(wert1)
    else:
        for wert2 in wert1:
            if type(wert2)==int:
               print("\t", wert2)
            else:
                for wert3 in wert2:
                    print("\t\t", wert3)
```

Hinweis:
`\t erzeugt Tabulator`

```
…
# Vertauschen von Werten
if kleinereZahl > groessereZahl:
    kleinereZahl, groessereZahl = groessereZahl, kleinereZahl
…
```

Fast jede andere Programmiersprache

braucht eine Hilfsvariable oder einen Kellerplatz zum zeitweisen Abspeichern der einen Zahl, damit diese für die Übernahme der anderen bereitsteht, erst dann kann die zweite mit dem Wert aus der Hilfsvariable versehen werden

```
//Tauschen in PASCAL
hilfVar:=kleinereZahl;
kleinereZahl:=groessereZahl;
groessereZahl:=hilfVar;
```

9.2. Mengen

keine echte Informatik-Daten-Struktur, aber wichtiges Element in Python

Mengen sind Sammlungen von Objekten in denen jedes Objekt nur einmal vorkommt, eine Reihenfolge oder Ordnungs-Struktur gibt es nicht
in der Mathematik weden Mengen in geschweiften Klammern notiert
eine Menge ohne ein Element ist eine leere Menge

9.2.1. Mengen – einfach

9.2.1.1. Mengen-Erstellung

in Python gibt es die veränderlichen Mengen, die mit **set()** erstellt werden und es gibt unveränderliche Mengen für deren Erstellung die Funktion **frozenset()** zuständig ist

```
>>> menge1=set([1,3,5,4,2,3])
>>> menge2=set([2.1, 0.0, 5.3, 7.9])
>>> menge3=set("Farbenspiel")
>>> menge4=set(["gelb", "grün", "rot", "blau", "blau", "blau"])
>>> menge1
{1, 2, 3, 4, 5}
>>> print(menge1)
{1, 2, 3, 4, 5}
>>> menge2
{0.0, 5.3, 7.9, 2.1}
>>> menge3
{'p', 'l', 's', 'F', 'r', 'b', 'a', 'i', 'n', 'e'}
>>> menge4
{'rot', 'gelb', 'grün', 'blau'}
>>>
```

Die Ausgabe erfolgt immer schön ordentlich in geschweiften Klammern. Ev. mehrfach auftauchende Objekte werden eliminiert

```
>>> alphabet=frozenset("abcdefghijklmnopqrstuvwxyz")
>>> print(alphabet)
frozenset({'l', 'k', 'o', 'v', 'i', 'p', 't', 'f', 'a', 'z', 'j',
'n', 'g', 'm', 's', 'w', 'b', 'q', 'u', 'x', 'r', 'h', 'y', 'd', 'c',
'e'})
>>> alphabet
frozenset({'l', 'k', 'o', 'v', 'i', 'p', 't', 'f', 'a', 'z', 'j',
'n', 'g', 'm', 's', 'w', 'b', 'q', 'u', 'x', 'r', 'h', 'y', 'd', 'c',
'e'})
>>>
```

Interessant ist hierbei, dass frozenset's scheinbar anders zusammengestellt werden, als normale Mengen (set's). Die sind sortiert, während die Elemente im frozenset scheinbar willkürlich auftauchen, obwohl sie im Ursprungs-Objekt sortiert vorkamen.
Die Ausgabe verdeutlicht uns immer, dass wir es hier mit einer feststehenden / unveränderlichen Menge zu tun haben.

9.2.1.2. Mengen-Operationen

Ein existierendes Frozenset kann in Python niemals das Ergebnis einer Mengen-Operation werden, da mit ihnen die Unveränderlichkeit verbunden ist. Aber sie können natürlich Argument bzw. Operant sein.

einfache Operationen

len(menge)
gibt die Anzahl der Elemente in der Menge zurück

min(menge)

max(menge)

elem **in** menge

elem **not in** menge

teilmenge **<=** menge
ist True, wenn teilmenge eine Teilmenge von der Menge menge ist

teilmenge **<** menge
ist True, wenn teilmenge eine echte Teilmenge von der Menge menge ist

menge1 **|** menge2
erzeugt neue (Vereinigungs-)Menge von menge1 und menge2; die alle Elemente von beiden Mengen enthält

menge1 **&** menge2
erzeugt neue (Schnitt-)Menge von menge1 und menge2, die nur gemeinsame Elemente enthält

menge **-** teilmenge
erzeugt neue Menge, die alle Elemente von menge enthält, außer sie kommen in teilmenge vor
Differenz-Bildung

menge1 **^** menge2

erzeugt neue (Vereinigungs-)Menge von menge1 und menge2, außer den Elementen, die in beiden mengen enthalten sind
ergibt symmetrische Differenz oder auch Vereinigungs-Menge – Schnitt-Menge

typischen Mengen-Operationen

Zum Veranschaulichen der Mengen-Operationen erstellen wir uns zwei einfache Mengen:

```
>>> menge1=set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> menge2=set([0, 2, 4, 6, 8, 10, 12])
>>> menge1
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> menge2
{0, 2, 4, 6, 8, 10, 12}
>>>
```

Aus der Mathematik kennen wir als typische Mengen-Operationen die Vereinigung, den Durchschnitt und die Differenz.

Vereinigung

Unter der Vereinigung von Mengen versteht man die Gesamt-Menge aus den Teilmengen. In beiden Mengen mehrfach vorkommende Elemente sind in der Ergebnis-Menge natürlich nur einmal vorhanden.

```
>>> menge1 | menge2
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}
>>>
```

Durchschnitt

Der Durchschnitt zweier Mengen beschreibt die Menge der gemeinsamen Elemente aus beiden Mengen. In der Ergebnis-Menge kommen diese Elemente nur einmalig vor.

```
>>> menge1 & menge2
{0, 8, 2, 4, 6}
>>>
```

Differenz

Bei der Differenz von Mengen werden aus der ersten Menge, die Elemente entfernt, die auch in der subtrahierten Menge vorkommen, entfernt.
Die Differenzen zweier Mengen sind i.A. nicht symmetrisch bzw. kommutativ. D.h. normalerweise ist Menge1 – Menge2 ≠ Menge2 – Menge1 (alternative Notierung: Menge1 \ Menge2 ≠ Menge2 \ Menge1).

```
>>> menge1 - menge2
{1, 9, 3, 5, 7}
>>> menge2 - menge1
{10, 12}
>>>
```

Durchschnitt und Vereinigung sind dagegen kommutativ.

Bearbeitung in Schleifen etc.

Mengen lassen sich ebenfalls mit Schleifen durchlaufen. Wir benötigen wieder einen Interator (eine Laufvariable), um auf die einzelnen Elemente zuzugreifen.

iter(menge)
liefert einen Interator für die Menge

9.2.1.x. automatische Mengen-Generierung

Ähnlich, wie bei den Listen (→ 8.4.0.5. Listen-Erzeugung – fast automatisch) lassen sich auch Mengen automatisch generieren. Die Konstrukte unterscheiden sich praktisch nicht. Lediglich die Verwendung der Schlüsselwörtchen set bzw. frozenset kommt hinzu.

```
>>> quadrate=set(i**2 for i in range(16))
>>> quadrate
{0, 1, 64, 225, 4, 36, 100, 196, 9, 169, 16, 49, 81, 144, 25, 121}
>>>
```

Natürlich hätte man die Mengen für die Veranschaulichung der Mengen-Operationen (s.a.w.v.) auch mittels Generator (i for i in range(10)) erzeugen können.

```
>>> menge1=set(i for i in range(10))
>>> menge2=set(i*2 for i in range(7))
>>> menge1
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> menge2
{0, 2, 4, 6, 8, 10, 12}
>>>
```

Weitere Möglichkeiten können gerne in der Listen-Besprechung nachgeschlagen werden (→ 8.4.0.5. Listen-Erzeugung – fast automatisch).

9.2.2. Mengen – objektorientiert

menge1.**add**(element | menge2)
fügt das Element oder eine Menge2 der Menge1 hinzu

menge.**clear**()
löscht alle Elemente aus der Menge
es entsteht eine leere Menge

menge.**discard**(element)
das Element wird aus der Menge entfernt, wenn es dann dort enthalten ist

menge.**pop**()
liefert ein zufällig gewähltes Element aus der Menge zurück. Das Element selbst wird aus der Menge entfernt!

menge.**remove**(element)
das Element wird aus der Menge entfernt, wenn es dann dort enthalten ist, wenn es nicht vorhanden ist, gibt es eine Fehler-Meldung (KeyError)

die aufgezählten Operationen gelten nur für normale Mengen (set's), da sie Veränderlichkeit unterstellen
alle nachfolgenden Operationen gelten für set's bzw. frozenset's

menge.**copy**()
erstellt eine (flache) Kopie der Menge

menge1.**difference**(menge2)
berechnet die Differenz von Menge1 und Menge2 (menge1 – menge2 bzw. menge1 \ menge2)

menge1.**intersection**(menge2)
erstellt den Durchschnitt aus beiden Mengen

menge1.**union**(menge2)(
vereint die beiden Mengen

menge1.**issubset**(menge2)
prüft, ob Menge1 eine Teilmenge von Menge2 ist
liefert True bzw. False zurück

menge1.**issuperset**(menge2)
prüft, ob Menge1 die Obermenge von Menge2 ist
liefert True bzw. False zurück

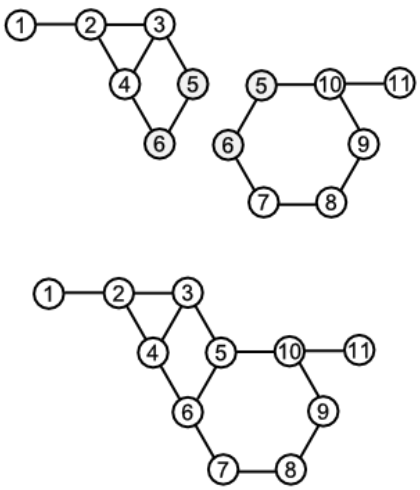
9.2.3. Anwendung von Mengen

9.2.3.1. ein bißchen Graphen

Graphen sind geometrische Objekte, die durch Knoten und Kanten beschrieben werden. Jeder Knoten (hier nummeriert) hat mindestens eine Verbindung (/ Kante) zu einem anderen.

Graphen werden z.B. zur Beschreibung von Wege- oder Raum-Plänen benutzt. Die Einmündungen bzw. Kreuzungen oder eben die Räume entsprechen den Knoten. Die möglichen Verbindungen (Wege oder Türen) zwischen den Knoten sind die Kanten.

Im nachfolgenden Programm werden neben "normalen" Mengen (set's) auch feste Menge (frozenset's) und Tupel (→ 9.1. Tupel) verwendet.



zwei vorgegebene Graphen (oben)
und der gemeinsame Graph (unten)

```
def findeNachbarKnoten(graph, knoten):
# Funktion zum Durchsuchen des Graphen nach den Nachbarknoten
# zu einem vorgegebenem Knoten
    alleKnoten, alleKanten = graph
    KantenDesKnoten = set(k for k in alleKanten if knoten in k)
    NachbarKnoten = set()
    for k in KantenDesKnoten:
        NachbarKnoten = NachbarKnoten | k
        NachbarKnoten= NachbarKnoten - set([knoten])
    return NachbarKnoten

def vereinigeGraphen(graph1, graph2):
# Funktion zur Verbindung von zwei Graphen (gemeinsame Knoten
# müssen gleiche Bezeichnung in beiden Graphen haben (mind. einer notw.!)
    return (graph1[0] | graph2[0], graph1[1] | graph2[1])

# =======Beispiel-Graphen (Daten)

Graph1Knoten={1,2,3,4,5,6}
Graph1Kanten= set(frozenset(k)
                  for k in [(1,2), (2,3), (2,4), (3,4), (3,5), (4,6), (5,6)])
Graph1=(Graph1Knoten, Graph1Kanten)

Graph2Knoten={5,6,7,8,9,10,11}
Graph2Kanten= set(frozenset(k)
                  for k in [(5,6), (5,11), (6,7), (7,8), (8,9), (9,10), (10,11)])
Graph2=(Graph2Knoten, Graph2Kanten)

# =======Hauptprogramm (Beispiel)
GesamtGraph=vereinigeGraphen(Graph1, Graph2)
print("Gesamtgraph: ...")
print("     Knoten: ", end='')
for i in GesamtGraph[0]:
    print(i, end='; ')
```

```
print()
print("     Kanten: ", end='')
for j in GesamtGraph[1]:
    print(tuple(j), end='; ')
print()
print("---------------------------------")
eingabe=1
while eingabe>0:
    eingabe=eval(input("Für welchen Knoten werden die Nachbarn gesucht?"
                       +" (Abbruch mit 0) ?: "))
    if eingabe>0:
        NachbarKnoten=findeNachbarKnoten(GesamtGraph, eingabe)
        print("Der Knoten",eingabe,"hat die / den Nachbarknoten: ",end='')
        for n in NachbarKnoten:
            print(n, end=', ')
        print()
    print()
input()
```

```
>>>
Gesamtgraph: ...
     Knoten: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
     Kanten: (2, 4), (1, 2), (5, 6), (6, 7), (3, 5), (8, 9), (3, 4),
(8, 7), (2, 3), (4, 6), (10, 11), (11, 5), (9, 10),
---------------------------------
Für welchen Knoten werden die Nachbarn gesucht? (Abbruch mit 0) ?: 4
Der Knoten 4 hat die / den Nachbarknoten: 2, 3, 6,

Für welchen Knoten werden die Nachbarn gesucht? (Abbruch mit 0) ?: 1
Der Knoten 1 hat die / den Nachbarknoten: 2,

Für welchen Knoten werden die Nachbarn gesucht? (Abbruch mit 0) ?: 0

>>>
```

9.3. Dictonary's - Wörterbücher

übersetzt Verzeichnis, praktisch eine Sammlung von Daten-Paaren, einem Schlüsselwert (Key) und einem Datenwert oder Eintrag (Value)
entspricht also einem Wörterbuch, deshalb auch gerne für direkte Übersetzungen benutzt. In der gesprochenen Sprache aber eher nur für einzelne Worte geeignet.
Angabe in geschweiften Klammern
prinzipiell Listen-artige Struktur; kann durch eine Liste aus zwei-stelligen Listen ersetzt werden
für "mehr-sprachige" Wörterbücher ist u.U. eine "mehr-spaltige" ("mehr-stellige") Liste besser geeignet.
(dann benötigt man auch keine gespiegelten Dictonary's, wobei dass auch nur programm-technisch interessant ist)

Notierung auch mehrzeilig möglich, dann muss die erste geschweifte Klammer in der Defintions-Zeile stehen und die abschließende Klammer hinter der letzten Zeile.
Die Wörterbuch-Einträge sollten Zeilen-weise notiert werden.
Dictonary's eigenen sich auch gut zum Abspeichern und Einlesen aus einer Text-Datei.

ebenfalls keine typische Daten-Struktur, Mischung aus Daten-Strukturen für die interne Daten-Verwaltung und -Verarbeitung
besonders auch bei der Speicherung der Daten bedeutsam

Anhängen eines neuen Eintrags
dictonary_name[neuer_schlüssel] = neuer_eintrag

Bestimmen der Länge / Größe / Einträgezahl eines Dictonary's
len(dictonary_name**)**
jedes Paar zählt als ein Eintrag

Löschen eines Eintrages durch Angabe des Schlüssels
del dictonary_name[schlüssel]

ist die Position bekannt, dann kann auch ein Löschen über den Index erfolgen
del(dictonary_name[index]**)**

weitere Objekt-orientierte Möglichkeit des Löschens eines Eintrages über **.pop()**

```
Koordinaten = {
"Erlangen": [56,23],
"Berlin": [34,51],
"München": [12,23],
"Rostock": [11,45]
}

for key in Koordinaten:
      print( key, Koordinaten[key])
```

```
>>>
München [12, 23]
Rostock [11, 45]
Erlangen [56, 23]
Berlin [34, 51]
```

```
Hauptstädte={
    "Baden-Württemberg":"Stuttgart",
    "Bayern":"München",
    "Berlin":"Berlin",
    "Brandenburg":"Potsdam",
    "Bremen":"Bremen",
    "Hamburg":"Hamburg",
    "Hessen":"Wiesbaden",
    "Mecklenburg-Vorpommern":"Schwerin",
    "Nordrhein-Westfalen":"Düsseldorf",
    "Niedersachsen":"Hannover",
    "Rheinland-Pfalz":"Mainz",
    "Saarland":"Saarbrücken",
    "Sachsen":"Dresden",
    "Sachsen-Anhalt":"Magdeburg",
    "Schleswig-Holstein":"Kiel",
    "Thüringen":"Erfurt"
}

for k in hauptstädte.items():
    print(k[0]+ " hat die Hauptstadt " + k[1])
```

```
>>>
Baden-Würtemberg hat die Hauptstadt Stuttgart
Bayern hat die Hauptstadt
Berlin hat die Hauptstadt Berlin
Brandenburg hat die Hauptstadt Potsdam
Bremen hat die Hauptstadt Bremen
Hamburg hat die Hauptstadt Hamburg
Mecklenburg-Vorpommern hat die Hauptstadt Schwerin
Nordrhein-Westfalen hat die Hauptstadt Düsseldorf
Niedersachsen hat die Hauptstadt Hannover
Rheinland-Pfalz hat die Hauptstadt Mainz
Saarland hat die Hauptstadt Saarbrücken
Sachsen hat die Hauptstadt Dresden
Sachsen-Anhalt hat die Hauptstadt Magdeburg
Schleswig-Holstein hat die Hauptstadt Kiel
Thüringen hat die Hauptstadt Erfurt
```

```
def Stadtstaaten():
    StadtstaatenListe=[]
    for land in Hauptstädte.items():
        if land[0] == land[1]:
            StadtstaatenListe.append(land[0])
    return StadtstaatenListe


def spiegeln(Dict):
    SpiegelDict={}
    for eintrag in Dict.items():
        SpiegelDict[eintrag[1]]=eintrag[0]
    return SpiegelDict
```

Schlüssel und Werte eines Wörterbuch's spiegel

```
woerterbuch = {
    1: "uno"
    2: "due"
    3: "tres"
    '0': "zero"
}

print(woerterbuch)

getauschtesWoerterbuch = {}
for schluessel, wert in woerterbuch.items():
    if wert not in getauschtesWoerterbuch:
        getauschtesWoerterbuch[wert] = []
    getauschtesWoerterbuch[wert].append(schluessel)


print(getauschtesWoerterbuch)
```

```
def spiegeln(woerterbuch):
    gespiegelt={}
    for schluessel, wert in woerterbuch.items():
        if wert not in gespiegelt:
            gespiegelt[wert]=schluessel
        #gespiegelt[wert].append(schluessel)
    return gespiegelt

print(woerterbuch)
print(spiegeln(woerterbuch))
```

Aufgaben:

1. *Erstellen ein Wörterbuch und ein kleines Anzeige-Programm (für alle Einträge) für die nächstkleinere Verwaltungs-Einheit Ihres Bundeslandes / Stadtstaates!*
2. *Ergänzen Sie dann eine – sich wiederholende – Abfrage eines beliebigen Schlüssels aus Ihrem Dictonary mit Anzeige des zugehörigen Eintrages in Form eines vollständigen Satzes! Der Abbruch der Eingabe soll bei der Eingabe eines leeren Textes erfolgen, fehlerhafte Eingabe sollen als solche auf dem Bildschirm vermerkt werden!*
3. *Erstellen Sie ein Deutsch-Englisch- und Englisch-Deutsch-Wörterbuch-Programm, dass bekannte Wort-Paare anzeigt und unbekannte lernt! (Wir gehen von einer immer richtigen Eingabe der Vokabeln aus!) Das Anfangs-Vokabular soll mindestens 30 Wort-Paare bzw. die Vokabeln der letzten Unterrichts-Module enthalten.*

9.3.1. Erfassen von unbekannten Objekten und Zählen der Objekte in einem Wörterbuch

Da man Dictonary's mit beliebigen Schlüsseln betreiben kann und auch die Aufnahme-Men ge an unterschiedlichen Objekten nicht begrenzt ist, bieten sie sich für das Zählen von irgendwelchen Objekten an.
Damit das Dictonary sowohl in der Funktion – als auch im Haupt-Programm nutzbar ist, definiert man es vor der Notierung der Funktion (im Haupt-Programm).

```
Anzahl={}

def zaehle(objekt):
  Anzahl[objekt]+=1

print(Anzahl)
zaehle("Maus")
print(Anzahl)
```

Es gibt allerdings ein Problem: In unserem Anzahl-Wörterbuch gibt es gar keine Schlüsselwörter.
Habe ich eine definierte Menge, kann ich sie im Vorfeld gleich mit definieren. Universeller – aber auch nicht perfekt – ist die nebenstehende Version:
Jetzt wird immer dann, wenn kein passendes Objekt in dem Anzahl-Wörterbuch gefunden wird, ein neues mit dem Zähl-Wert 1 angelegt.
Überlegen wir uns nun noch ein kleines Test-Programm für unsere Wörterbuch-Struktur.
Dabei wird auch die Effektivität der Speicherung deutlich.
Statt in einem Feld von 100 Objekten, werden jetzt nur die Objekte erfasst und zählend gespeichert, die wirklich bei einem Zufalls-Erzeugungs-Verfahren entstehen.
Bei einer weiteren Nutzung des Wörterbuches muss man ev. beachten, dass es bestimmte Einträge eben nicht gibt. Um hier keinen Laufzeitfehler zu bekommen, muss man dann vor der Benutzung immer die Existenz abtesten!

```
Anzahl={}

def zaehle(objekt):
  if objekt in Anzahl:
    Anzahl[objekt]+=1
  else:
    Anzahl[objekt]=1

zaehle("Maus")
print(Anzahl)
```

```
from random import randint

Anzahl={}

def zaehle(objekt):
  if objekt in Anzahl:
    Anzahl[objekt]+=1
  else:
    Anzahl[objekt]=1

for _ in range(50):
  zaehle(randint(1,20))
print(Anzahl)
```

```
>>>
{1: 4, 2: 1, 3: 5, 5: 3, 6: 3, 7: 2, 8: 3, 9: 3, 10: 6,
11: 6, 12: 1, 13: 2, 14: 2, 15: 3, 16: 3, 17: 1, 19: 1,
20: 1}
>>>
```

Aufgaben:

1. ***Verbessern Sie das obige Programm zum Zählen der Zufallszahlen um eine verständliche Ausgabe!***
2. ***Erstellen Sie eine Programm-Version eines Würfel-Programms, das 200x würfelt und die Wurf-Anzahl hinterher als Balken-Diagramm aus Rauten darstellt!***
3.

9.3.2. Objekt-orientierte Operationen mit Dictonary's

Löschen eines Eintrages aus dem Dictonary über den Index
dictonary_name**.pop(**index**)**
die Funktion liefert ein reduziertes Dictonary zurück

dictonary_name**.clear()**
löscht den Inhalt des Dictonary

dictonary_name**.copy()**
erzeugt eine flache Kopie von dictonary_name
erzeugt Alias-Dictonary

dictonary_name**.items()**
gibt eine Liste aller gespeicherter Schlüssel mit ihren Werten im Dictonary (jeweils Tupel-weise) zurück

dictonary_name**.keys()**
gibt eine Liste aller gespeicherter Schlüssel im Dictonary zurück

dictonary_name**.key()**

dictonary_name**.values()**
gibt eine Liste aller gespeicherter Werte (Value's) im Dictonary zurück

dictonary_name**.values()**
gibt eine Liste aller gespeicherter Werte im Dictonary zurück

dictonary_name**.get(schluessel, alternativwert)**
liefert den Wert zum Schlüssel zurück (wenn dieser existiert); sonst den Alternativwert

dictonary_name**.setdefault(schluessel, inhalt)**
setzt im Schlüssel-Eintrag/Inhalt-Paar mit dem angegebenen Schlüssel (wenn dieser vorhanden ist) den Inhalt → dic[schluessel] = inhalt
wenn Eintrag mit Schlüssel schon vorhanden ist, dann wird dessen aktueller Inhalt zurückgeliefert

dictonary_name**.iteritems()**
zum Durchlaufen aller Schlüssel-Eintrag/Inhalt-Paare in einer for-Schleife

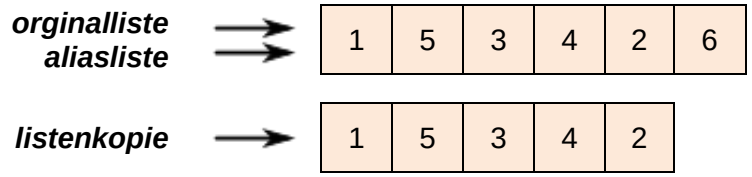
dictonary_name**.iterkeys()**
zum Durchlaufen aller Schlüssel in einer for-Schleife

dictonary_name**.itervalues()**
zum Durchlaufen aller Einträge einer for-Schleife

9.7. Listen, die II. – objektorientierte Listen

Im Abschnitt "Listen, die I. (→ 8.4. Listen, die I. – einfache Listen)haben wir die Listen ganz einfach betrachtet. Nun gehen wir zur Objekt-orientierten Nutzung über. Das hört sich vielleicht irgendwie kompliziert an, ist es aber gar nicht. Vom Objekt-orientierten Ansatz merken wir kaum etwas. Nur die übliche Objekt-orientierte Schreibung bzw. der Aufruf der Funktionen über die Punkt-Schribweise erinnert an sie. Ach ja, und jetzt heißen die Funktionen Methoden. Da werden wir uns aber schnell eingewöhnen.

Wir greifen auf auf das Listen-Beispiel aus dem ersten kapitel zurück. Wer will, kann ja noch mal schnell nachschlagen (→ 8.4. Listen, die I. – einfache Listen).



einige Methoden auf Listen lassen Operationen zu, die man eher den Keller- bzw. Warteschlangen-Datenstrukturen zuordnen würden
hie wird wieder die herausragende Rolle der Listen als Daten-Objekte (in Python und auch sonst) sichtbar

listenname.**remove**(element)
löscht aus der Liste das angegebene Element

??? listenname**.del(**index**)**
löscht aus der Liste das Element an der Index-Position

listenname**.count(**element**)**
zählt, wie häufig ein Elementes in einer Liste ist

listenname**.append(**element**)**
hängt ein Element an die Liste mit dem angegebenen Namen an
werden mehrere Elemente angehängt, dann erfolgt dies als ein Element, also als eigene Liste, das Ergebnis wäre eine Liste am Ende der (alten) Liste
Rückgabewert ist None, dieser muss aber nicht entgegengenommen werden

listenname**.extend()**
hängend ein oder mehrere Elemente einzeln an eine Liste an

listenname**.insert(**index, element**)**
fügt ein Element in eine Liste an der indizierten Position ein

element=listenname**.pop()**
liefert das letzte Element der Liste zurück, es wird aus der Liste entfernt!

einige Programmiersprachen kennen neben pop auch noch peek bei peek wird das letzte Element aber nicht entfernt, sondern nur gelesen; in python lässt sich hierfür liste[-1] benutzen

element=listenname**.pop(**index**)**
liefert das indizierte Element der Liste zurück, es wird aus der Liste entfernt!

listenname**.sort()**
sortiert die Elemente einer Liste aufsteigend
bietet als Argument noch die Schlüsselwörtchen key und reverse
mit reverse wird die Liste absteigend (also umgekehrt) sortiert

listenname.sort(reverse=True)

mit dem Wörtchen key kann man der Sortierfunktion noch eine einargumentige Funktion übergeben, die als Sortierkriterium dienen soll z.B. die Länge der Strings (→ len())

listenname.sort(key=len)

die Schlüsselwörtchen lassen sich auch gemeinsam verwenden

durch den Allias wird lediglich ein weiterer Zeiger (genannt Variable) auf die gleiche Liste gelegt

beim Verändern der einen Liste wird die "andere" Liste mit geändert

nur mit Deep-Kopie kann eine unabhängige Kopie erstellt werden

```
>>> a=[3,2,1]
>>> a
[3, 2, 1]
>>> b=a
>>> b
[3, 2, 1]
>>> a.sort()
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>>
```

auch bei Übernahme einer Alias-Liste in eine andere Liste wird nur der Zeiger übernommen

Veränderungen an der "originalen" Alias-Liste wirken sich auch eine weitere Alias-Nutzung aus

```
>>> a=[3,2,1]
>>> a
[3, 2, 1]
>>> b=["A", "B","C"]
>>> b
["A", "B", "C"]
>>> a.append(b)
>>> a
[1, 2, 3,["A", "B", "C"]]
>>> b.reverse
>>> b
["C", "B", "A"]
```

```
>>> a
[1, 2, 3, ["C", "B", "A"]]
>>>
```

listenname**.reverse()**
sortiert die Elemente einer Liste absteigend

listenkopie = copy.deepcopy(originalliste)

listenname**.split()**
zerlegt einen String in die Teile, die durch Leerzeichen voneinande getrennt sind

```
element_liste = text.split()
```

Erstellen eines Strings aus den Elementen einer Liste
listenname**.join()**

```
liste = ['a', 'b', 'c']]
print(''.join(liste))     ➔ 'abc'

print(' '.join(liste))     ➔ 'a b c'
print('_'.join(liste))     ➔ 'a_b_c'
```

Listen:
Vorteile:

- effektive Speichernutzung
- schnelles Einspeichern (Anhängen)
- einfache Algorithmen (suchen (, entfernen, einfügen an Position))

Nachteile:

- allgemein Arbeits-aufwändiger
- langsames Suchen

9.8. Keller

auch Stack (engl. = Stapel, Haufen)

was auf dem Stapel als letztes abgeladen wird, muss als erstes wieder entnommen werden, um z.B. an tiefer liegende / früher eingespeicherte Daten zu erreichen

bekannt z.B. aus der Rekursion (→ 8.4.2. Rekursion) dort ist Kellerspeicher zwingend notwendig, allerdings vom Nutzer unbemerkt

Größe einer KELLER_Datenstruktur wird im Wesentlichen vom verfügbaren / hierfür reservierten Speicher bestimmt
ansonsten Anzahl der Einträge beliebig

nur wenn Speicher des Rechners nicht mehr für die Größe des Kellerspeichers ausreicht (bei zu vielen Rekursionen), dann kommt es zum Fehler

LIFO-Speicher (Last-In-First-Out) oder Stack

alternatives Speicher-Prinzip ist die (Warte-)Schlange (→ 9.9. Warteschlangen) oder der FIFO-Speicher (First-In-First-Out)

Definition(en): Stack / Keller
Ein Keller- oder Stack-Datenstruktur ist eine lineare Anordnung von gleichartig zu bearbeitenden Daten-Obbjekten (Daten-Einträgen), die nach dem LIFO-Prinzip verwaltet werden.

zum Keller gehörenden Grund-Operationen:
nachsehen (top) → obersten Eintrag ansehen / auslesen ohne es zu entfernen
einlagern / einspeichern (push) → neues Element (oben) auf den alten Stapel legen
wegnehmen / ausspeichern (pop) → obersten Eintrag entnehmen

Stack	**Typ / Klasse**
liste: Liste (von Objekt)	Objekt
gibListe(): Liste setzListe(liste: Liste) istLeer(): Wahrheitswert einspeichern(eintrag: Objekt) ausspeichern(): Objekt lesen(): Objekt	Methoden

eine sehr einfache Implementierung eines Kellers (Stack's) über eine interne Liste:

```
def keller():
    liste = []

    def raus():
        if not istleer():
            return liste.pop()

    def rein(element):
        liste.append(element)

    def istleer():
        return len(liste)==0

    return raus, rein, istleer

REIN, RAUS, ISTLEER = keller()
```

Q und Fkt.W: ???; angelehnt an Objekt-orientierter Prog.

Listen-basiert
push = kellerliste.append
pop = kellerliste.pop

```
# Menü-System + Keller-Speicher mit Listen-Opp's

def anzeigenListe(Liste):
    for elem in Liste:
        print("[",elem,"]",end='  ')
    print("|<=")

def elementeZaehlen(Liste):
    anzahl=0
    for _ in Liste:
        anzahl+=1
    return anzahl

def istLeereListe(Liste):
    anzahl=elementeZaehlen(Liste)
    if anzahl==0:
        return True
    else: return False

def leerenListe(Liste):
    while not istLeereListe(Liste):
        Liste.pop()


# Main
KellerSpeicher=[]
maxMenuePunkte=3
auswahl=1
while auswahl>0 and auswahl<=maxMenuePunkte:
    print("")
    print("aktueller Keller-Speicher:")
    anzeigenListe(KellerSpeicher)
    print("")
    print("Auswahl-Menü")
    print("============")
    print("<1> .. Einspeichern (Push)")
    print("<2> .. Ausspeichern (Pop)")
    print("<3> .. Speicher leeren")
    print(".. ")
    print("<0> .. Programmende")
    auswahl=-1
    while auswahl<0 or auswahl>maxMenuePunkte:
         auswahl=eval(input("Ihre Wahl: "))
    if auswahl==1:
         eingabe=input("Was soll eingespeichert werden?: ")
         KellerSpeicher.append(eingabe)
    elif auswahl==2:
         if istLeereListe(KellerSpeicher):
             print("Keller-Speicher ist LEER.")
         else:
             ausgabe=KellerSpeicher.pop()
             print("Element: [", ausgabe, "] aus dem Speicher gelesen
und entfernt.")
    elif auswahl==3:
         leerenListe(KellerSpeicher)
    else:
        break

print("Ende...")
```

```
aktueller Keller-Speicher:
|<=

Auswahl-Menü
============
<1> .. Einspeichern (Push)
<2> .. Ausspeichern (Pop)
<3> .. Speicher leeren
<4> ..
<1> ..
..
<0> .. Programmende
Ihre Wahl: 1
Was soll eingespeichert werden?: 3

aktueller Keller-Speicher:
[ 3 ]  |<=
```

etwas aufwändigere Implementierung (Q: de.wikipedia.org)

```
class Stack(object):
    def __init__(self):
        self.maxindex=5
        self.topindex=0
        self.speicher = [0,0,0,0,0,0,0,0,0,0,0]

    def isEmpty(self):
        return self.topindex==0
    def isFull(self):
        return self.topindex==self.maxindex
    def push(self,element):
        if not self.isFull():
            self.topindex+=1
            self.speicher[self.topindex]=element
    def pop(self):
        if not self.isEmpty():
            self.topindex-=1
    def top(self):
        if not self.isEmpty():
            return self.speicher[self.topindex]

    def DisplayStack(self):
        M=self.topindex

        while M>0 :

            print("|     ",self.speicher[M],"         |")
            M-=1
        print("------------------|")


if __name__== "__main__":                                   #Beispiel
    myStack=Stack()
    print(myStack.isFull())
    print(myStack.isEmpty())
    myStack.push(5)
    myStack.push(3)
    myStack.DisplayStack()
    print(myStack.isEmpty())
    myStack.push(13)
    myStack.DisplayStack()
```

```
import random
# max operation on a stack

class Node:
  def __init__(self):
    self.data = None # contains the data

class StackNode:
  def __init__(self):
    self.maxNode = None # contains the data
    self.nextNode = None

class Stack:
  def __init__(self):
    self.head = None

  def push(self, node):

    toAdd = StackNode()
    if self.head:
      toAdd.nextNode = self.head
      if node.data > self.head.maxNode.data:
        toAdd.maxNode = node
      else:
        toAdd.maxNode = self.head.maxNode
    else:
      toAdd.maxNode = node
    self.head = toAdd


  def pop(self):
    toReturn = None
    if self.head:
      toReturn = self.head
      if self.head.nextNode:
        self.head = self.head.nextNode
      else:
        self.head = None

    return toReturn

  def max(self):
   return self.head.maxNode.data



stack = Stack()
for i in range (0,10):
  node = Node()
  node.data = random.randint(1,20)
  print "Pushing: " + str(node.data)
  stack.push(node)

print stack.max()
```

Q: http://pythonfiddle.com/max-operator-to-stack/

9.9. Warteschlangen

FIFO-Prinzip (First In First Out)

Queue (sprich: kju)

Beispiele:
Kasse im Supermarkt
Warten beim Frisör / Arzt / …
Abarbeitung von Überweisungen bei einer Bank

Definition(en): Warteschlange / Queue
Eine Warteschlangen- bzw. Queue-Datenstruktur ist eine lineare Anordnung von gleichartig zu bearbeitenden Daten-Objekten (Daten-Einträgen), die nach dem FIFO-Prinzip verwaltet werden.

zum Keller gehörenden Grund-Operationen:
nachsehen (front) → ersten Eintrag ansehen / auslesen ohne ihn zu entfernen
anhängen / einspeichern (enqueue) → neues Element (hinten) auf die alten Schlange anhängen / kontatenieren
entfernen / ausspeichern (dequeue) → ersten / vordersten Eintrag entnehmen

alternatives Speicher-Prinzip ist der Keller (→ 9.8. Keller) bzw. der Stack (Stapel) oder der LIFO-Speicher

Queue	**Typ / Klasse**
liste: Liste (von Objekt)	Objekt
gibListe(): Liste setzListe(liste: Liste) istLeer(): Wahrheitswert einspeichern(eintrag: Objekt) ausspeichern(): Objekt lesen(): Objekt	Methoden

für eine Implementierung über eine einfache Liste:

links-orientiertes Arbeiten:
insert(0,x) zum Einspeichern (neue Liste ::= x, alte Liste) (Erweiterung links)
pop() Ausspeichern / Entnehmen des letzten Element's (Verkürzung rechts)

rechts-orientiertes Arbeiten:
append(x) anhängen eines Eintrags an die Liste (einspeichern) (Erweiterung rechts)
pop(0) Ausspeichern / Entnehmen des 1. Element's (Verkürzung links)

```
# Menü-System + Warteschlangen-Speicher mit Listen-Opp's

def anzeigenListe(Liste):
    print("->|",end='  ')
    for elem in Liste:
        print("[",elem,"]",end='  ')
    print("|=>>")

def elementeZaehlen(Liste):
    anzahl=0
    for _ in Liste:
        anzahl+=1
    return anzahl
    # Alternative
    # return len(Liste)

def istLeereListe(Liste):
    anzahl=elementeZaehlen(Liste)
    if anzahl==0:
        return True
    else: return False
    # Alternative
    # if len(Liste)==0: return True
    # return False

def leerenListe(Liste):
    while not istLeereListe(Liste):
        Liste.pop()


# Main
WarteschlangenSpeicher=[]
maxMenuePunkte=3
auswahl=1
while auswahl>0 and auswahl<=maxMenuePunkte:
    print("")
    print("aktueller FIFO-Speicher (Warteschlange):")
    anzeigenListe(WarteschlangenSpeicher)
    print("")
    print("Auswahl-Menü")
    print("============")
    print("<1> .. Einspeichern (Push)")
    print("<2> .. Ausspeichern (Pop)")
    print("<3> .. Speicher leeren")
    print(".. ")
    print("<0> .. Programmende")
    auswahl=-1
    while auswahl<0 or auswahl>maxMenuePunkte:
         auswahl=eval(input("Ihre Wahl: "))
    if auswahl==1: # Einspeichern
         eingabe=input("Was soll eingespeichert werden?: ")
         WarteschlangenSpeicher.insert(0,eingabe)
```

```
    elif auswahl==2: # Ausspeichern
         if istLeereListe(WarteschlangenSpeicher):
             print("Keller-Speicher ist LEER.")
         else:
             ausgabe=WarteschlangenSpeicher.pop()
             print("Element: [", ausgabe, "] aus dem Speicher gelesen
und entfernt.")
    elif auswahl==3: # Speicher leeren
         leerenListe(WarteschlangenSpeicher)
    else: # Ende
        break

print("Ende...")
```

9.10. Bäume

9.11. Graphen

siehe auch bei Listen II
siehe auch bei Mengen → 9.2.3.1. ein bißchen Graphen

9.12. endliche Automaten

```
class DFA:
    current_state = None;
    def __init__(self, states, alphabet, transition_function, start_state,
accept_states):
        self.states = states;
        self.alphabet = alphabet;
        self.transition_function = transition_function;
        self.start_state = start_state;
        self.accept_states = accept_states;
        self.current_state = start_state;
        return;

    def transition_to_state_with_input(self, input_value):
        if       ((self.current_state,       input_value)       not       in
self.transition_function.keys()):
            self.current_state = None;
            return;
        self.current_state  =  self.transition_function[(self.current_state,
input_value)];
        return;

    def in_accept_state(self):
        return self.current_state in accept_states;

    def go_to_initial_state(self):
        self.current_state = self.start_state;
        return;

    def run_with_input_list(self, input_list):
        self.go_to_initial_state();
        for inp in input_list:
            self.transition_to_state_with_input(inp);
            continue;
        return self.in_accept_state();
    pass;



states = {0, 1, 2, 3};
alphabet = {'a', 'b', 'c', 'd'};

tf = dict();
tf[(0, 'a')] = 1;
tf[(0, 'b')] = 2;
tf[(0, 'c')] = 3;
tf[(0, 'd')] = 0;
tf[(1, 'a')] = 1;
tf[(1, 'b')] = 2;
tf[(1, 'c')] = 3;
tf[(1, 'd')] = 0;
tf[(2, 'a')] = 1;
tf[(2, 'b')] = 2;
tf[(2, 'c')] = 3;
tf[(2, 'd')] = 0;
tf[(3, 'a')] = 1;
```

```
tf[(3, 'b')] = 2;
tf[(3, 'c')] = 3;
tf[(3, 'd')] = 0;
start_state = 0;
accept_states = {2, 3};

d = DFA(states, alphabet, tf, start_state, accept_states);

inp_program = list('abcdabcdabcd');

print d.run_with_input_list(inp_program);
```

Q: http://pythonfiddle.com/dfa-simple-implementation/

9.13. Keller-Automaten

9.14. TURING-Automaten

Abbildungen und Skizzen entstammen den folgende ClipArt-Sammlungen:

/A/

andere Quellen sind direkt angegeben.

Alle anderen Abbildungen sind geistiges Eigentum:

lern-soft-projekt: drews (c,p) 1997 – 2022 lsp: dre
für die Verwendung außerhalb dieses Skriptes gilt für sie die Lizenz:

CC-BY-NC-SA

Lizenz-Erklärungen und –Bedingungen: http://de.creativecommons.org/was-ist-cc/
andere Verwendungen nur mit schriftlicher Vereinbarung!!!

verwendete freie Software:

- **Inkscape** von: inkscape.org (www.inkscape.org)
- **CmapTools** von: Institute for Human and Maschine Cognition (www.ihmc.us)

⊞- (c,p) 2015 - 2022 lern-soft-projekt: drews -⊟
⊞- drews@lern-soft-projekt.de -⊟
⊞- http://www.lern-soft-projekt.de -⊟
⊞- 18069 Rostock; Luise-Otto-Peters-Ring 25 -⊟
⊞- Tel/AB (0381) 760 12 18 FAX 760 12 11 -⊟