

Grundkurs: Programmieren

Einführung in grundlegende Programmierkonzepte mit Python

Maren Krafft

WS 18/19

Universität Passau

Einführung in die Programmierung

- Name
- Studiengang
- Programmiererfahrung allgemein
- Programmiererfahrung Python
- Erwartungen

- Anwesenheitspflicht
- Teilnahmebestätigung (Zertifikat)
- "Regeln"
- Codio
- Skript

14.00 - 14.15	Erwartungen und Vorkenntnisse
14.15 - 14.45	Einführung in Python und Umgebung
14.45 - 15.30	Datentypen, Operatoren, Variablen und Zuweisungen
15.30 - 15.45	Pause
15.45 - 16.45	Bedingte Ausführung
17.00 - 18.00	Schleifen

10.15 - 10.30	Besprechung Tagesplan
10.30 - 11.30	Wiederholung
11.00 - 11.30	Funktionen
11.30 - 11.45	Pause
11.45 - 12.00	Listen
13.00 - 13.45	Mittagspause
13.45 - 15.30	Datein einlesen/ausgeben
15.30 - 16.00	allgemeine Theorie

Die Programmiersprache Python

- Warum Python?
 - flache Lernkurve, sehenswerte Ergebnisse bereits nach dem ersten Tag
 - verankert in Forschung und Wirtschaft
 - der englischen Sprache sehr ähnlich

What's The Best Programming Language for First-Time Learners? (Poll Closed)

Java 17.63% (3,291 votes)



Ruby 8.39% (1,566 votes)



Python 34.16% (6,376 votes)



C/C++ 23.29% (4,347 votes)



JavaScript 16.53% (3,085 votes)



Total Votes: 18,665

Quelle: lifehacker.com

Codio

Bild einfügen zur Oberfläche. benutzernamen hinweisen.

Kommentare

- Wir kommentieren mit #

```
1  # Einfach so
```

- Einzeiler
- Sinnvolle Kommentare
- Am Anfang jeder Python-Datei ein Kommentar, der den Inhalt beschreibt

- Fast alles wird klein geschrieben
- Es gibt Ebenen (durch Einrückungen = 4 Leerzeichen)
- Leerzeilen und Umbrüche sind nicht nötig, aber manchmal sinnvoll

- wird "von oben nach unten" ausgeführt
- kein automatisches springen (nach oben) oder neu starten
- ein komplettes Python Dokument mit allen Befehlen

- wird ähnlich wie in der Mathematik verwendet (nur nicht mit Zahlen)
- eine Vielzahl von Befehlen (vorgefertigt oder selbstgeschrieben) zusammengefasst in einer bestimmten Schreibweise

```
1 #Ein Beispielcode
2
3 e2g_dict = {'a':'ein', 'is':'ist', 'test':'Test', '
    this':'dies'}
4
5 # Englisch nach Deutsch uebersetzen
6 def translate(english):
7     return e2g_dict[english]
8
9 esentence = 'this is a test'
10 elist = esentence.split()
11 glist = []
12
13 for eword in elist:
14     glist = glist + [translate(eword)]
15
16 gsentence = " ".join(glist)
17 print gsentence
```

Datentypen

Lernziele

- Die wichtigsten Datentypen kennenlernen
- Diese ausgeben können
- Datentypen in andere Datentypen umwandeln

Erste wichtige Funktion: print()

print():

Gibt alles innerhalb der Klammern aus.

```
1 print("Hallo")
2 print(1)
3 print(1+2)
```

String, str:

- ist eine Zeichenkette
- wird in " " geschrieben

```
1 "Ich bin vom Typ String, eine Reihe von Zeichen"  
2 "1"  
3 " "
```

Hello World

```
1 print("Hello World!")
```

- Dieser Befehl gibt den Text (String) "Hello World!" aus.

Glückwunsch

Ihr habt gerade euer erstes Codeprogramm geschrieben!

Zahlen

Integer, int:

- ist eine ganze Zahl

```
1 -1
2 2
3 3
4 ...
```

Float, float:

- ist eine Gleitkommazahl

```
1 3.1415
2 3.0
3 -2.3
```

Boolean, bool:

- Wahrheitswert

```
1 True
2 False
```

- `int(...)`: Castet zu int.
- `float(...)`: Castet zu float.
- `str(...)`: Castet zu String.

Wandle um und gebe mit `print()` aus

- 5 zu "5"
- "5.0" zu 5.0
- "Hallo" + 5 zu "Hallo 5"

Typumwandlung

- `int(...)`: Castet zu int.
- `float(...)`: Castet zu float.
- `str(...)`: Castet zu String.

Wandle um und gebe mit `print()` aus

- 5 zu "5"
- "5.0" zu 5.0
- "Hallo" + 5 zu "Hallo 5"

```
1 print(str(5))  
2 print(float("5.0"))  
3 print("Hallo" + " " + str(5))
```

Operatoren

- Rechenoperatoren
- Vergleichende Operatoren
- Logische Operatoren

- $+$ und $-$
- $*$ und $/$
- Modulo $\%$ (entspricht dem Rest, der durch eine Teilung entsteht)

Zahlen und Rechenoperatoren - Übung

Was ergeben folgende Ausdrücke? Überprüfe mit Python.

```
1 print("Ich" + " bin " + str(10) + " Jahre alt")
2 print("Hallo"*2)
3 print(2.45 + 3)
4 print("Hallo " + "3")
5 print(1/2.5 +2)
6 print(3%2)
7 print(6%3)
```

Zahlen und Rechenoperatoren - Übung

Was ergeben folgende Ausdrücke? Überprüfe mit Python.

```
1 print("Ich" + " bin " + str(10) + " Jahre alt")
2 print("Hallo"*2)
3 print(2.45 + 3)
4 print("Hallo " + "3")
5 print(1/2.5 +2)
6 print(3%2)
7 print(6%3)
```

```
1 Ich bin 10 Jahre alt
2 HalloHallo
3 5.45
4 Hallo 3
5 2.4
6 1
7 0
```

Vergleichende Operatoren

Wollen wir aber Datentypen vergleichen, benötigen wir weitere Operatoren.

Diese ergeben immer einen Booleanwert (True/False).

- `==` prüft zwei Werte auf Gleichheit
- `!=` prüft zwei Werte auf Ungleichheit
- `>` größer (bei Strings wird automatisch die Länge verglichen)
- `<` kleiner (bei Strings wird automatisch die Länge verglichen)
- `<=`, `>=` kleiner-gleich, größer-gleich (bei Strings wird automatisch die Länge verglichen)

Vergleichende Operatoren - Übung

Was ergeben folgende Ausdrücke? Überprüfe mit Python.

```
1 print(3 > 4)
2
3 print(6 != 7)
4
5 print("Hallo" < "Hallo Welt!")
6
7 print("Hallo" == "Hallo Welt")
```

Vergleichen von zwei Wahrheitswerten (meist auf Grundlage von vergleichenden Operatoren)

Diese ergeben immer einen Booleanwert (True/False).

- **and** logisches 'Und' (True, wenn beide Seiten wahr sind)
- **or** logisches 'Oder' (True, wenn eine Seite, die andere oder beide wahr sind)
- **not** verneint einen Ausdruck (Verneinung: aus True wird False, aus False wird True)

Was ergeben folgende Ausdrücke? Überprüfe mit dem Python Interpreter.

```
1 print(3 > 4 or 6 != 7)
2
3 print("Hallo" < "Hallo Welt!" and 3 > 4)
4
5 print(not( "Hallo" == "Hallo Welt"))
```


Variablen, Zuweisungen und Typumwandlung

Lernziele

- Kennenlernen von Variablen und Zuweisungen
- Variablen und Zuweisungen anwenden

- Eine Art Platzhalter/Speicherplatz
- Man kann in ihnen Werte speichern
- Sie werden kleingeschrieben
- Wenn möglich sinnvoll benennen
- Bsp. name, alter, prozent, age, pi, todelete

- Zuweisung von Werten zu einer Variablen mit dem Zuweisungsoperator =

```
1 a = 5
2 b = 3.14
3 c = "Hallo Grundkurs:Programmieren"
```

- der Variable kann auch das Ergebnis einer Operation zugewiesen werden

```
1 divisor = 1000
2 dividend = 200
3 percent = dividend / divisor * 100
```

Fülle die ... aus

```
1  topprint = "Hallo"
2  print(topprint)
3  Ausgabe: ...
4
5  name = ...
6  alter = ...
7  print(...)
8  Ausgabe soll sein: Max Mustermann ist 20 Jahre alt
```

Fülle die ... aus

```
1  toprint = "Hallo"
2  print(toprint)
3  Ausgabe: ...
4
5  name = ...
6  alter = ...
7  print(...)
8  Ausgabe soll sein: Max Mustermann ist 20 Jahre alt
```

```
1  toprint = "Hallo"
2  print(toprint)
3  Ausgabe: Hallo
4
5  name = "Max Mustermann"
6  alter = 20
7  print(name + " ist " + str(alter) + " Jahre alt")
```

Erneute Zuweisung

- Soll einer Variable ein neuer Wert zugewiesen werden, so ist eine neue Zuweisung mit = notwendig.
- Beispiel: a um 5 erhöhen. Korrigiere den Code

```
1   a = 5  
2   a + 5  
3   print(a)
```

Erneute Zuweisung

- Soll einer Variable ein neuer Wert zugewiesen werden, so ist eine neue Zuweisung mit = notwendig.
- Beispiel: a um 5 erhöhen. Korrigiere den Code

```
1   a = 5  
2   a + 5  
3   print(a)
```

```
1   a = 5  
2   a = a + 5  
3   print(a)
```


Der Unterschied zwischen = und == ist sehr wichtig.

- == Vergleich beider Seiten; gibt False/True zurück
- = ist eine Zuweisung (Lernen wir im nächsten Kapitel kennen)

Welche Ausgabe wird folgendes Programm haben?

```
1  a = 21
2  b = 21
3  a == b + 1
4  c = a==b
5  print(c)
```

Eine weitere wichtige Funktion: input()

input()

- Liest die letzte Konsolenzeile ein
- gibt den Konsoleneintrag als String zurück
- input("") Gibt in der Konsole den Inhalt innerhalb der "" aus bevor eingelesen wird.
- z.B name = input()

Aufgabe

- Lasse dich von deinem Programm begrüßen, indem du `input("Hallo, wie heißt du? ")` verwendest
- deinen Namen als Eingabe in einer Variable speicherst.
- Lasse ausgeben: `Hallo 'name'`

Aufgabe

- Lasse dich von deinem Programm begrüßen, indem du `input("Hallo, wie heißt du? ")` verwendest
- deinen Namen als Eingabe in einer Variable speicherst.
- Lasse ausgeben: `Hallo 'name'`

Lösung

```
1  name = input("Hallo, wie heisst du?")
2  'Maren'
3  print("Hallo " + name)
```

Aufgabe

- Lasse dich von deinem Programm begrüßen, indem du mit `input "Hallo, wie heißt du? "` ausgeben lässt. (hast du bereits)
- Deinen Namen als Eingabe in einer Variable speicherst. (hast du bereits)
- Lasse das Programm nach deinem Alter mit (`"Wie alt bist du? "`) fragen
- Speichere die Eingabe als Integer in einer Variable
- Erhöhe das Alter danach um 1
- Lasse ausgeben: `"Du heißt 'name' und wirst 'alter' Jahre alt."`

Lösung

```
1 name = input("Hallo, wie heisst du? ")
2 alter = int(input("Wie alt bist du? "))
3 alter = alter + 1
4 print("Du heisst " + name + " und wirst " + str(alter)
      + " Jahre alt.")
```

Bedingte Ausführung

Bedingte Ausführung

Umgangssprachlich:

Wenn (if) eine **Bedingung** True ist, dann führe Programmcode 1 aus, andernfalls (else) Programmcode 2

```
1 if Bedingung == True:
2     # Programmcode 1
3 else:
4     # Programmcode 2
```

```
1 zahl = int(input())
2
3 if zahl > 10:
4     print("Die Zahl ist > 10.")
5 else:
6     print("Die Zahl ist <= 10.")
```

Schreibe ein Programm, dass eine Eingabe einliest und speichert.
mit `input("Wetter: ")` Falls die Eingabe "Schnee" lautet, gebe "Es ist Winter" aus. Andernfalls gebe "Noch einmal Glück gehabt" aus.

Bedingte Ausführung

Schreibe ein Programm, dass eine Eingabe einliest und speichert. mit `input("Wetter: ")` Falls die Eingabe "Schnee" lautet, gebe "Es ist Winter" aus. Andernfalls gebe "Noch einmal Glück gehabt" aus.

```
1 inp = input("Wetter: ")
2 if inp == "Schnee":
3     print("Es ist Winter")
4 else:
5     print("Noch einmal Glück gehabt")
```

Mehrfach bedingte Ausführung

Umgangssprachlich:

Wenn (if) eine **Bedingung1** True ist, dann führe Programmcode 1 aus,

falls nicht dann prüfe (elif) ob **Bedingung2** True ist, dann führe Programmcode 2 aus,

andernfalls (else) Programmcode 3

```
1  if Bedingung == True:
2      # Programmcode 1
3  elif Bedingung2 == True:
4      # Programmcode 2
5  else:
6      # Programmcode 3
```

Mehrfach bedingte Ausführung - Beispiel

```
1 zahl = int(input())
2
3 if zahl > 10:
4     print("Die Zahl ist > 10.")
5 elif zahl > 5:
6     print("Die Zahl ist > 5 und <= 10.")
7 else:
8     print("Die Zahl ist <= 5.")
```

Ergänze das zuvor erstellte Programm so, dass nicht nur der Fall "Schnee" abgefragt wird, sondern auch "Sonne". Bei Eingabe Sonne verwende `print("Ein paar letzte Sonnenstrahlen")` und `print("Genieße solange wie möglich")`

Bedingte Ausführung

Ergänze das zuvor erstellte Programm so, dass nicht nur der Fall "Schnee" abgefragt wird, sondern auch "Sonne". Bei Eingabe Sonne verwende `print("Ein paar letzte Sonnenstrahlen")` und `print("Genieße solange wie möglich")`

```
1 inp = input("Wetter: ")
2 if inp == "Schnee":
3     print("Es ist Winter")
4 elif inp == "Sonne":
5     print("Ein paar letzte Sonnenstrahlen")
6     print("Genieße solange wie möglich")
7 print("Noch einmal Glück gehabt")
```

- Bedingungen und Schleifen (dazu später) können beliebig oft ineinander geschachtelt werden
- Erkennbar durch Einrückungen
- Beachte Logik
- Zu viele Schachtelungen führen zu Unübersichtlichkeit => schlechter Code

Schachtelung bedingter Ausführungen

```
1 zahl = int(input())
2
3 if zahl < 10:
4     if zahl < 5:
5         print("Die Zahl ist < 5")
6     else:
7         print("Die Zahl ist >= 5 und < 10")
8 else:
9     print("Die Zahl ist > 10")
```

Aufgabe: Hundesalter in Menschenalter

Bei kleinen Hunden entspricht das erste Lebensjahr etwa 20 Menschenjahren. Das zweite entspricht 8 Jahren und alle weiteren Hundejahre entsprechen jeweils 4 Menschenjahren. Bei einem 5-jährigen Hund rechnen Sie also: $20 + 8 + 4 + 4 + 4 = 40$. Fünf Hundejahre wären demnach etwa 40 Menschenjahre.

Bedingte Ausführung - Übung 1

Kurz:

- 1 Hundejahr = 20 Jahre
- 2 Hundejahre = 28 Jahre
- Über 2 Jahren = $20 + 8 + (\text{alter} - 2) * 4$ Jahre

Aufgabe: Es soll ein Programm geschrieben werden, das mit `input()` nach dem Alter fragt (nur positives Hundesalter). Mit bedingter Ausführung das Menschenalter ermittelt und ausgibt.

- `input("Alter des Hundes: ")`
- bedingte Ausführung
- `print("Das entspricht ca. ??? Jahren.")`

Lösung

```
1 alter = int(input("Alter des Hundes: "))
2 if age == 1:
3     print("Das entspricht ca. 28 Jahren.")
4 elif age == 2:
5     print("Das entspricht ca. 28 Jahren.")
6 else:
7     human = 28 + (age - 2)*4
8     print("Das entspricht ca. " + str(human) + " Jahren.")
```

Schleifen

- for-Schleife
- while-Schleife

Aufgabe 1

Erweitere das Programm so, dass der String "Hello World" 6-mal ausgegeben wird.

Aufgabe 1

Erweitere das Programm so, dass der String "Hello World" 6-mal ausgegeben wird.

Lösung

```
1  print("Hello World!")
2  print("Hello World!")
3  print("Hello World!")
4  print("Hello World!")
5  print("Hello World!")
6  print("Hello World!")
```


for-Schleife

Umgangssprachlich:

Eine Variable (kann ein Buchstabe oder Wort sein) nimmt den Anfangswert an und erhöht sich pro Schleifendurchlauf um 1, solange die Variable $<$ Ende. (Sie durchläuft alle Elemente in der range)

```
1  for Variable in range(Anfang, Ende):  
2      # Programmcode
```

```
1  for i in range(1,7):  
2      print("Hello World!")
```

Verändere den Code so, dass ...

- Aufgabe 1: ...nur dreimal "Hello World!" ausgegeben wird
- Aufgabe 2: ...nach jedem der 3 "Hello World!" ein "Hello" ausgegeben

Verwendung des Parameters im Code

Aufgabe

Schreibe ein Programm, das von 1 bis 100 zählt.

1, 2, 3,

Lösungsvariante 1

```
1  print(1)
2  print(2)
3  print(3)
4  print(4)
5  print(5)
6  print(6)
7  print(7)
8  ...
```

Verwendung des Parameters im Code

Aufgabe

Schreibe ein Programm, das von 1 bis 100 zählt.

1, 2, 3,

Lösungsvariante 1

```
1  print(1)
2  print(2)
3  print(3)
4  print(4)
5  print(5)
6  print(6)
7  print(7)
8  ...
```

```
1  for i in range(1, 101):
2      print(i)
```

Verändere den Code so, dass...

- Aufgabe 1: nur Zahlen zwischen 35 und 40 ausgegeben werden
- Aufgabe 2: die Quadratzahlen für 1 bis 4 ausgegeben werden

Die while-Schleife

Umgangssprachlich:

Wiederhole den Programmcode solange die Bedingung True ist.

Gefahr: Falls immer das gleiche (Bedingung wird nicht verändert) geprüft wird => Schleife ohne Ende

```
1 while Bedingung == True:  
2     # Programmcode
```

Beispiel zähle bis 3

```
1 i = 1  
2 while i <= 3:  
3     print(i)  
4     i = i+1
```

Aufgabe 1

Verändere den Code so, dass er mit einer while-Schleife 6-mal den String "Hello World" ausgibt.

Aufgabe 1

Verändere den Code so, dass er mit einer while-Schleife 6-mal den String "Hello World" ausgibt.

Lösung

```
1 i = 1
2 while i <= 6:
3     print("Hello World!")
4     i = i+1
```


Aufgabe 2

Schreibe einen Code, der solange fragt "Nerv ich dich? " bis er die Eingabe "JA!" erhält.

Danach lass das Programm "Schade" ausgeben.

Aufgabe 2

Schreibe einen Code, der solange fragt "Nerv ich dich? " bis er die Eingabe "JA!" erhält.

Danach lass das Programm "Schade" ausgeben.

Lösung

```
1 inp = input("Nerv ich dich? ")
2 while inp != "JA!":
3     inp = input("Nerv ich dich? ")
4 print("Schade")
```

Magische Miesmuschel

Programmiere einen einfachen Bot.

Er wartet mit ("Du: ") auf eine Nein/Ja-Frage und beantwortet sie mit einer zufälligen (random) Antwort ("Bot: " + Antwort).

Er fragt so lange nach bis er "Ich will nicht mehr" als Eingabe erhält. Er verabschiedet sich mit "Bot: Bye"
nötig:

- input
- while-Schleife
- Bedingung
- random Zahl (soll einer bestimmten Antwort zugeordnet werden)

Die while-Schleife

Lösung

```
1 from random import *
2
3 inp = input("Du: ")
4 while inp != "Ich will nicht mehr" :
5     zahl = randint(0, 1)
6     if zahl == 0:
7         print("Bot: Nein")
8     else:
9         print("Bot: Ja")
10    inp = input("Du: ")
11 print("Bot: Bye")
```

Wiederholung

- String (str)
- Boolean (bool)
- Integer (int)
- Float (float)

- ==: prüft zwei Werte auf Gleichheit
- !=: prüft zwei Werte auf Ungleichheit
- >: größer
- <: kleiner
- <=, >=, kleiner-gleich, größer-gleich
- and: logisches 'Und'
- or: logisches 'Oder'
- not: verneint einen Ausdruck

- Werte werden Variablen mit `=` zugewiesen
- sinnvolle, kleingeschriebene Variablennamen
- auf Variablentyp achten (welchen Datentyp hat der zugewiesene Wert)

Bedingte Ausführung

```
1  if Bedingung == True:
2      # Programmcode 1
3  elif Bedingung2 == True:
4      # Programmcode 2
5      ...
6  else:
7      # Programmcode 3
```

for-Schleife

```
1 for Variable in range(Anfang, Ende):  
2     # Programmcode
```

while-Schleife

```
1 while Bedingung == True:  
2     # Programmcode
```

print()

- Gibt einen String aus
- Die einzelnen Teile in den Klammern müssen vom gleichen Datentyp sein (meist String)

input()

- Liest eine Zeile ein
- Gibt einen String zurück (Typumwandlung hinterher)
- `input("Hallo")` gibt Hallo aus, bevor die Eingabe erwartet wird

Aufgabe: Ein komplexerer Chatbot

- Wiederholung bis Eingabe "Genug"
- Eingabe einlesen mit "Du: "
- Wenn die Eingabe "Alles okay?" lautet, soll dreimal "Bot: SOS" ausgegeben werden
- Wenn die Eingabe "Wie geht es dir?" lautet, soll zufällig "Bot: Gut", "Bot: Schlecht", oder "Bot: Passt schon" ausgegeben werden (`from random import *` und `randint`)
- Wenn die Eingabe "Was ist der Sinn des Lebens" lautet, soll "Bot: 42" ausgegeben werden

Wiederholungsaufgabe

- Wenn die Eingabe "Rechne aus wie alt ich bin" lautet, soll der Bot nach dem Geburtsjahr fragen und dann nach dem Monat (als Zahl) und dann nach dem Tag.
Fall 1: Falls $\text{Monat} < 11$ or ($\text{Monat} == 11$ and $\text{Tag} < \text{heutiger Tag}$)
Fall 2: heute Geburtstag
Fall 3: noch nicht Geburtstag gehabt
- sonst "Bot: Stelle mir eine andere Frage"
- Sobald die Eingabe "Genug" lautet und die Schleife verlassen worden ist, gebe "Bot: Bis bald!" aus

Funktionen

- Funktionen ohne Parameter
- Funktionen mit Parameter
- Rückgabewert

- Fasst mehrere sinnvolle Befehle zusammen
- kann später im Code aufgerufen werden
- "Auslagerung von Code" (Beispielsweise, weil er häufiger verwendet werden soll)

Funktionen ohne Parameter

Jedes mal, wenn die Funktion `name()` im späteren Verlauf aufgerufen wird, wird der Programmcode ausgeführt.

Wichtig: Die Definition der Funktion ist keine Ausführung der Funktion.

```
1 def name():  
2     #Programmcode
```

Beispiel

```
1 def greet():  
2     print("Hey!")  
3     print("How are you?")
```

Funktionen mit Rückgabewert

Die Funktion gibt einen Wert zurück, der einer Variablen zugewiesen werden kann.

```
1 def name():  
2     #Programmcode  
3     return rückgabewert
```

```
1 def greet():  
2     print("Hey!")  
3     print("How are you?")  
4     return 2  
5 fragen = greet()  
6 print(fragen)
```

Funktionen mit Parameter

Es werden ein oder mehrere Parameter übergeben.

Bei jedem Aufruf muss die Anzahl an Parametern übergeben werden.

```
1 def name(a, b, c, ....):  
2     #Programmcode
```

```
1 def greet(name):  
2     print("Hey!" + name)  
3     print("How are you?")
```

Funktionen mit Parameter und Rückgabewert

```
1 def name(a, b, c, ....):  
2     #Programmcode  
3     return wert
```

```
1 #gibt die Summe zweier Werte zurück  
2 def sum(wert1, wert2):  
3     return wert1 + wert2
```

Schreibe eine Funktion, die die Summe aus 3 Zahlen bildet.

Lese 3 Zahlen jeweils ein und speichere sie ab.

Rufe die Funktion mit den 3 Zahlen als Parameter auf, speichere das Ergebnis und gebe es aus.

Listen

- Einfache Listen
- Wichtige Listenfunktionen
- Mehrdimensionale Listen

Einfache Listen

- Datenstruktur
- Speichert eine beliebige Anzahl an Elementen
- wird mit eckigen Klammern [] dargestellt
- Trennung der Elemente durch Kommas
- Das erste Element ist der 0. Eintrag

```
1 zahlen = [1, 2, 3, 4, 5, 6]
2 texte = ["Hallo", "Welt", ".", "Grundkurs", "
    Programmieren"]
3 gemischt =[True, "String", 3]
```

Einfacher Zugriff auf Elemente

- Stelle(index) wird benötigt

```
1 zahlen = [1, 2, 3, 4, 5, 6]
2
3 #Speichert den Inhalt der 2.Position in pos2
4 pos2 = zahlen[1]
5
6 #Speichert in die Liste an Position 2 die Zahl 99
7 zahlen[1] = 99
8 #aktualisierte Liste: [1, 99, 3, 4, 5, 6]
```

Listenfunktionen

Seien $s = [1, 2]$ und $t = [3, 4]$ zwei Listen

- $s + t$

```
1 s + t == [1, 2, 3, 4]
```

- $\text{len}(s)$

```
1 len(s) == 2
```

- $\text{min}(s)$

```
1 min(s) == 1
```

- $\text{max}(s)$

```
1 max(s) == 2
```

- $\text{sum}(s)$

```
1 sum(s) == 3
```

Listenfunktionen

Sei $s = [1, 2, 3, 4, 5]$

- `s.append(x)`

```
1 s.append(6) => s == [1, 2, 3, 4, 5, 6]
```

- `s.pop(i)`

```
1 s.pop(1) => s == [1, 3, 4, 5, 6]
2 #Besitzt einen Rückgabewert
```

- `s.remove(x)`

```
1 s.remove(3) => s == [1, 4, 5, 6]
```

- `s.insert(position, object)`

```
1 s.insert(2, 8) => s == [1, 4, 8, 5, 6]
```

Mehrdimensionale Listen/ Listen in Listen

- Listen innerhalb listen

```
1 matrix = [[1, 2], [3, 4]]
```

Speichert den Inhalt der Position 1 der inneren Liste, die selbst auf Position 0 der äußeren Liste ist in pos2. `pos2 == 2`

```
1 pos2 = zahlen[0][1]
```

Speichert 99 an die Stelle `[1][0]`

```
1 zahlen[1][0] = 99  
2 #aktualisierte Liste: [[1, 2],[99, 3]]
```

Wichtige Funktionen

Die Liste bietet eine große Anzahl an Funktionen, die auf ihnen ausgeführt werden können.

```
1 liste = ["Grundkurs", "Programmieren", 42, "Pie",  
2         3.14]  
3 liste[2] = 99  
4 len(liste)  
5 liste.append("Passau")  
6 liste.insert(2, "Falke")  
7 liste.index(3.14)  
8 liste.remove(3.14)  
9 sum([1,3,5])  
10 max([1,3,5])
```

Aufgabe

Versuche zu erraten, was die Ausgabe dieses Programms ist.

```
1 liste_a = ["Hallo", "schoenes", "Wetter"]
2 liste_b = liste_a
3
4 liste_b[1] = "schlechtes"
5
6 print(liste_a[0], liste_a[1], liste_a[2])
```

Aufgabe

Versuche zu erraten, was die Ausgabe dieses Programms ist.

```
1 liste_a = ["Hallo", "schoenes", "Wetter"]
2 liste_b = liste_a
3
4 liste_b[1] = "schlechtes"
5
6 print(liste_a[0], liste_a[1], liste_a[2])
```

Lösung

Hallo schlechtes Wetter

Schreibe ein Programm, dass drei Prüfungs-Noten einliest, in einer Liste speichert und dir nach jeder Eingabe den Durchschnitt errechnet.

Schreibe ein Programm, dass drei Prüfungs-Noten einliest, in einer Liste speichert und dir nach jeder Eingabe den Durchschnitt errechnet.

Lösung

siehe Beamer

- Dateien einlesen
- Inhalte bearbeiten
- Umgang mit verschiedenen Dateitypen

Dateien lesen und schreiben

Um Daten, die unsere Programme ausgeben bzw. benötigen, brauchen wir eine Möglichkeit, diese zu speichern.

Dateien lesen und schreiben

Um Daten, die unsere Programme ausgeben bzw. benötigen, brauchen wir eine Möglichkeit, diese zu speichern.

```
1 daten = open("daten.txt", "r")
2 for line in daten:
3     print(line.rstrip())
4
5 daten.close()
```

Dateien lesen und schreiben

Um Daten, die unsere Programme ausgeben bzw. benötigen, brauchen wir eine Möglichkeit, diese zu speichern.

```
1 daten = open("daten.txt", "r")
2 for line in daten:
3     print(line.rstrip())
4
5 daten.close()
```

```
1 zahlen = [1, 2, 3]
2 daten = open("daten.txt", "w")
3 for zahl in zahlen:
4     daten.write(str(zahl))
5
6 daten.close()
```

Aufgabe

Baue das Notenprogramm so um, dass die Noten beim Start des Programms aus einer Datei gelesen werden und nach Abschluss der Eingabe wieder dort hinein geschrieben werden.

Je größer ein Projekt wird, desto wichtiger ist es, den Überblick zu behalten. Funktionen sind eine Art, das Programm übersichtlich zu halten. Objektorientierung eine weitere.

Je größer ein Projekt wird, desto wichtiger ist es, den Überblick zu behalten. Funktionen sind eine Art, das Programm übersichtlich zu halten. Objektorientierung eine weitere.

```
1 # einfachste Art einer Klasse
2 class Person:
3     pass
4
5 james = Person()
6
7 james.name = "James"
8 james.alter = 42
```

Objektorientierung, die `__init__()` methode

```
1 class Person():
2
3     def __init__(self):
4         self.name = "James"
5         self.alter = 42
6
7     def alter_plus_10(self):
8         return self.alter + 10
```

Aufgabe

- Verpacke das Notenprogramm in eine eigene Klasse
Notenprogramm mit dem Attribut `noten`, in dem die Noten gespeichert sind.
- Füge die Methode `errechne_durschnitt()` hinzu, die den Durchschnitt errechnet

- Unterscheidungsmerkmale
 - Programmierparadigma: imperativ, funktional oder objektorientiert
 - Typsicherheit
 - kompiliert vs. interpretiert
 - allgemein vs. domänenspezifisch
 - hardwarenah vs. höhere Programmiersprachen

- Imperative Programmiersprachen: C/C++, C#, Java ...
- Funktionale Programmiersprachen: SQL, Haskell, Erlang, (Scala) ...
- Objektorientierte Programmiersprachen: C++, C#, Java, Javascript, PHP, Python ...

Imperative Sprachen (C/C++, C#, Python, Java, ...)

- ältestes Programmierparadigma
- große Verbreitung in der Industrie
- besteht aus Befehlen (lat. imperare = befehlen)
- Abarbeiten der Befehle 'Schritt für Schritt'
- sagt einem Computer, 'wie' er etwas tun soll

```
1 print("Hey, whats' up?")
2 sleep(3)
3 print("Learning Python right now")
4 sleep(2)
```

- Verwendung
 - 'Standard-Software', hardwarenahe Entwicklung

Funktionale Sprachen (Haskell, Erlang, SQL, Lisp, ...)

- vergleichsweise modern
- sagt einem Computer, 'was' das Ergebnis sein soll
- `SELECT name FROM students WHERE major='law' AND semester='1';`
- Verwendung
 - akademische Zwecke
 - sicherheitskritische und ...
 - hoch performante Anwendungen

```
1 square :: [Int] -> [Int]
2 square a = [2*x | x <- a]
```

$$x = x + 1$$

Objektorientierte Sprachen (Java, Python, C++, C#, ...)

- starke Verbreitung
- Abbilden der realen Welt der Dinge auf Objekte
- Klasse: Bauplan eines Objekts bestehend aus Eigenschaften (Attributen) und Methoden
- Vererbung möglich
- Verwendung
 - Standard-Software
 - Modellierung realer Projekte (Unternehmen, Mitarbeiter, Kunden, Waren, ...)
 - große Projekte (→ Planung durch Klassendiagramme)

Objektorientierung: Beispiel

```
1  class Konto:
2      def __init__(self, name, nr):
3          self.inhaber = name
4          self.kontonummer = nr
5          self.kontostand = 0
6      def einzahlen(self, betrag):
7          self.kontostand = kontostand + betrag
8      def auszahlen(self, betrag):
9          self.kontostand = kontostand - betrag
10     def ueberweisen(self, ziel, betrag):
11         ziel.einzahlen(self.betrag)
12         self.auszahlen(betrag)
13     def kontostand(self):
14         return self.kontostand
15
16 class Unternehmenskonto(Konto):
17     def erhalteBonus(self, bonus):
18         self.kontostand = kontostand + bonus
```

- kompilierte Sprachen (Java, C/C++, C#, ...):
 - Übersetzung des (kompletten) Programmcodes in Maschinencode
 - dann Ausführung des Maschinencodes
- interpretierte Sprachen (Python, Lisp, PHP, JavaScript, ...):
 - Übersetzung einer einzelnen Programmanweisung
 - Ausführung dieser Anweisung
 - Übersetzung der nächsten Anweisung

Hardwarenahe und höhere Sprachen

- hardwarenah: abhängig von der Bauweise des Prozessors
- höhere Sprachen: von der Bauweise abstrahiert (print(), sleep())

```
1 .START ST
2   ST: MOV R1,#2
3       MOV R2,#1
4   M1: CMP R2,#20
5       BGT M2
6       MUL R1,R2
7       INI R2
8       JMP M1
9   M2: JSR PRINT
10      .END
```

```
1
2 a = 2;
3 i = 1;
4 # compare i ==
   20
5 # if True, jump
   to M2
6 a = a*i;
7 i++;
8 # jump to M1
9 print(a)
```

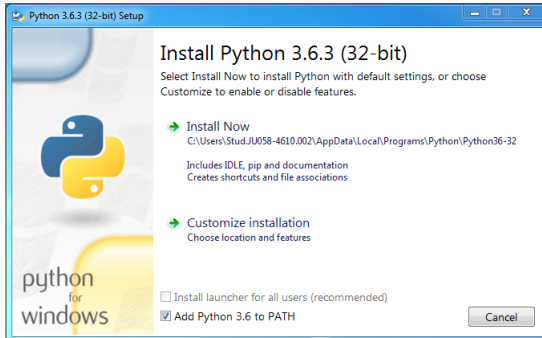
```
1 a = 2;
2 for i in range
   (1, 20) {
3
4     a = a*i;
5
6 }
7
8 print(a);
9
```

Populäre Programmiersprachen

- C++
 - imperativ, objektorientiert, typsicher, kompiliert, allgemein, höhere Sprache (dennoch hardwarenah)
 - große Verwendung in hocheffizienten Systemen (Betriebssysteme, Grafikberechnungen, Computerspiele, . . .)
 - Erweiterung von C mit Objektorientierung
- Java
 - imperativ, objektorientiert, typsicher, kompiliert, allgemein
 - im bayrischen Lehrplan und an vielen Universitäten 'erste' Sprache
 - ebenfalls große Verbreitung
- Python
 - (imperativ), (funktional), objektorientiert, dynamisch getypt, interpretiert, allgemein
 - große Verbreitung auch gerade im akademischen Umfeld, Web, Machine Learning und Data Science

Installieren von Python

- Python 3.6.3 unter <https://www.python.org/downloads/> herunterladen und ausführen
- Zum 'PATH' hinzufügen und '...for all users' deaktivieren



Achtung

Word, TextEdit, Notepad, oder Wordpad sind Textverarbeitungsprogramme, keine Quelltext-Editoren und schon gar keine Entwicklungsumgebungen

Achtung

Word, TextEdit, Notepad, oder Wordpad sind Textverarbeitungsprogramme, keine Quelltext-Editoren und schon gar keine Entwicklungsumgebungen

- Editoren wie Sublime Text, Atom oder IDLE sind für uns ausreichend
- große IDE's wie Eclipse, IntelliJ oder PyCharm bieten weitere Funktionen

- Pythonprogramme in IDLE schreiben und ausführen
 1. Datei > Neue Datei
 2. geeigneten Speicherort aussuchen, bspws.
Dokumente/GrundkursProgrammieren/helloworld.py
 3. Programm schreiben...
 4. Programm unter Run > Run Module ausführen oder F5 drücken

- Universität Passau: 'Programmierung I' (5102) an der FIM
- Automate the Boring Stuff with Python: Practical Programming for Total Beginner (Sweigart, 2015)
- 'How to think like a Computer Scientist' (Wentworth, Peter and Elkner, Jeffrey and Downey, Allen B and Meyers, Chris, 2011)

- Danke für die Teilnahme! Informationen zu weiteren Kursen im jeweiligen Semester beim ZKK
- www.evaluation.uni-passau.de (Unter Umständen muss noch das Zertifikat heruntergeladen werden)
- Sommersemester 2018 > ZKK IT-Kurse > Token eingeben