

Programmierung mit



Vorstellungsrunde



Kelvin Homann

- 1998 geboren in Neustadt a. Rbge.
- 2017 - 2021 Bachelor-Studium Mediendesigninformatik
- 2021 Start ins Berufsleben (IT-Consultant Software Engineering)

Wie könnt ihr mich erreichen?

kevin.homann@hs-hannover.de



Wer seid ihr?

- Name
- Alter
- Herkunft
- Werdegang
- Vorkenntnisse in der Softwareentwicklung?
- Wieso Softwareentwicklung?
- Erwartungen und Wünsche an den Kurs



Was ist Softwareentwicklung?

Softwareentwicklung = Programmierung?



Programmierung = Problemlösung

- Vorliegendes **Problem**
- Wahl einer **Programmiersprache**
- Entwicklung eines **Algorithmus** zur Lösung des Problems
- Überführung in ein ausführbares **Programm**

- **Programmierung im Kleinen:** Lösen von Teilproblemen, Aufgaben, ...
- **Softwareentwicklung (Programmierung im Grossen):** Verbinden von Teilproblemlösungen zu komplexeren Strukturen, Sowie Abbildung von Arbeitsabläufen, ...



Probleme - Die Probleme mit den Problemen

- Probleme können beliebig komplex sein
- **Niedrige Komplexität:** Wenn ich auf den Button klicke möchte ich, dass der Button grün wird
- **Mittlere Komplexität:** Ich möchte, dass sich Nutzer bei meinem Online-Dienst registrieren können
- **Hohe Komplexität:** Ich möchte, dass Nutzer in unserem Online-Dienst ihre komplette Steuererklärung machen können und diese anschließend direkt an das entsprechende Finanzamt vermittelt wird
- Vor dem Beginn der Programmierung müssen Probleme in beherrschbare Teilprobleme zerlegt werden (**Divide & Conquer**)



Algorithmen

- Werden in **Programmiersprachen** formuliert
- Beschreiben eine **Abfolge** von Verarbeitungsschritten
- Die ein vorliegendes **Problem** lösen
- Und sind so formuliert, dass sie **automatisiert** ausgeführt werden können
- Bsp.: Kochrezepte, Anleitungen zum Möbelaufbau, mathematische Formeln, ...



Programmiersprachen

- **Präzise formale Sprachen** zur Formulierung von Algorithmen
- Welche ist die beste Programmiersprache?
 - Für jeden Einsatzbereich gibt es verschiedene spezialisierte Sprachen, Somit gibt es die eine beste Sprache nicht
- Welche Programmiersprache sollte ich lernen?
 - Hängt stark vom Interessengebiet bzw. angestrebtem (Job) Ziel ab
- **“Kann man eine, kann man alle.“**
 - Paradigmen können sehr unterschiedlich sein
 - Grundstrukturen sind wiederkehrend



Programmiersprachen

- **Praxis:** Man lernt meist erst viele verschiedene Programmiersprachen und spezialisiert sich später
- Man kommt nicht mit Kenntnissen über nur eine einzige Sprache aus, auch um Stärken und Schwächen bewerten zu können



Python

- Multiparadigma-Sprache (objektorientiert, funktional, ...)
- 1994 von Guido van Rossum in Amsterdam entwickelt
- Einfach durch sehr einfache Syntax und somit Übersichtlichkeit
- Plattformunabhängig
- Open source
- Weit verbreitet
- Umfangreiche Bibliotheken, die Problemlösungen vereinfachen (UI, Datenbanken, Webentwicklung, Asynchronität, ...)
- 2021: Platz 3 Stackoverflow Developer Survey hinter Javascript und HTML/CSS



Programmierung mit Python – Es geht los 🤖

- Test der Zugänge mit den Hochschul-Accounts
- Installation von Visual Studio Code <https://code.visualstudio.com/>
- Installation von Python <https://www.python.org/downloads/windows/>
- Installation von git <https://git-scm.com/>
- Test in der Console/im Terminal
- DEMO :D
- `print('Hello World')`



Git-Cheatsheet

Repository Klonen:

- `git clone https://github.com/kelvin-homann/refugeeks`

Committen:

- `git commit -m "Eine sinnvolle Commit-Message"`

Branch erstellen:

- `git checkout -b "name_des_branch"`

Pushen:

- `git push`

Pullen:

- `git pull`

Wichtig! Status abfragen:

- `git status`



Programmierung mit Python – Grundstrukturen



Programmierung - Grundstrukturen

- Variablen
- Schleifen
- Bedingungen
- Funktionen
- Klassen
- Listen
- Kommentare



Variablen – Basis Datentypen

Integer

Repräsentiert Natürliche Zahlen

```
int  
integer_var = 0
```

Float

Repräsentiert Reelle Zahlen

```
float  
float_var = 2.051
```

String

Repräsentiert Text als Liste von Charakteren

```
str  
string_var = 'This is a String'
```



Variablen – Basis Datentypen

Boolean

Repräsentiert einen Wahrheitswert

```
bool  
boolean_var = True
```

None

Repräsentiert eine Variable mit "Nichts" als Wert

```
None  
null_var = None
```



Variablen – Überschreiben

In Python können Variablen beliebig überschrieben werden
- gilt nicht für alle Programmiersprachen!

```
boolean_var = True  
print(type(boolean_var)) # output: <class 'bool'>
```

```
boolean_var = 'Boolean var is a string now'  
print(type(boolean_var)) # output: <class 'str'>
```



Variablen – Casting

Typ von Variablen kann durch Casting in einen anderen Typen
Verwandelt werden

Explizites Casting

```
float_var = 20.051 # float  
print(type(float_var)) # output: <class 'float'>
```

```
float_var = str(float_var)  
print(type(float_var)) # output: <class 'str'>
```

Implizites Casting

```
int_var = 123 # int  
float_var = 1.23 # float
```

```
new_var = int_var + float_var # float
```



Aufgaben 1

1.

Schreibt ein Programm, das auf der Konsole einen Celsius-Wert einliest und den Fahrenheit-Wert zurückliefert.

2.

a) Recherchiert, was der BMI (Body Mass Index) ist. Schreibt ein Programm, das

- eine Floatzahl und eine Integerzahl einliest,
- den BMI berechnet und ausgibt.

b) Für eine schöne Ausgabe soll BMI-Wert auf eine Nachkommastelle gerundet werden. (Sucht auch hier den passenden Befehl!).

c) Gebt in Abhängigkeit vom errechnete BMI-Wert aus, ob es sich um Unter-, Normal- oder Übergewicht handelt.



Bedingungen – if-else

Ist gleich: $a == b$

Nicht gleich: $a != b$

Kleiner als: $a < b$

Kleiner als oder gleich: $a \leq b$

Größer als: $a > b$

Größer als oder gleich: $a \geq b$

```
a = 500  
b = 21
```

```
if b > a:  
    print("b ist größer als a")  
elif a == b:  
    print("a und b sind gleich")  
else:  
    print("a ist größer als b")
```



Bedingungen – switch-case

Bietet sich für viele Fallunterscheidungen an

Switch-Case: erst ab v3.10 möglich

number = 200

match number:

case 100:

print("Die Nummer 100")

case 200:

print("Gewonnen mit Nummer 200!")

case 300:

print("Die Nummer 300")

case _:

print("Diese Nummer kenne ich nicht")



Schleifen – for-Schleife

Kontrollstruktur um Anweisung wiederholt auszuführen

```
names = ["Joe", "Max", "Kelvin"]
```

```
# Anweisung für alle Strings anwenden
```

```
for name in names:
```

```
    print(name)
```

```
# Schleife bei Bedingung abbrechen
```

```
for name in names:
```

```
    print(name)
```

```
    if name == "Max":
```

```
        break
```

```
# Anweisung 10-mal ausführen
```

```
for i in range(10):
```

```
    print(i)
```

"name" oder "i" sind die sogenannten Laufvariablen



Schleifen – while-Schleife

"Während Bedingung wahr ist, führe Anweisung aus"

Eine sich wiederholende "if"-Bedingung bis die Bedingung "False" ist

```
i = 1  
  
while i < 10:  
    print(i)  
    i += 1 # i = i + 1
```

```
i = 1  
  
while i < 10:  
    print(i)  
    if i == 3:  
        break  
    i += 1
```



Aufgaben 2

1. Entwerft ein Programm, welches die Länge eines Strings ausgibt. Dabei soll natürlich nicht `len()` verwendet werden.
2. Entwerft ein Programm, dass die Fläche von geometrischen Figuren (Kreis, Quadrat, Rechteck) berechnet. Dazu soll die BenutzerIn
 - die gewünschte Form (Kreis, Quadrat, Rechteck) eingeben
 - die für diese erforderlichen Eingaben machen
 - den Flächenwert ausgeben.



Aufgaben 2

3. Schreibt ein Programm, das für eine beliebige Anzahl von Zahlen deren Durchschnitt berechnet.

Dazu soll erst die Anzahl der Zahlen eingegeben werden, im Anschluss werden entsprechend viel Zahlen eingegeben und der Durchschnitt angezeigt. Ein möglicher Ablauf ist wie folgt:

```
Für wieviele Zahlen soll der Durchschnitt berechnet werden: 3
zahl eingeben: 5
zahl eingeben: 7
zahl eingeben: 4
Summe: 16.0
Durchschnitt: 5.333333333333333
```



Listen - Motivation

Motivation

Werden zur Verarbeitung vieler Objekte benötigt.

Listen sind ein **strukturierter Datentyp** mit einer **internen Struktur**.

Python bietet 3 Datentypen dafür an:

- list
- tuple
- dict

Beispiel:

```
list = ["python", "refugeeks", "I am a string!"]
```



Listen – Zugriff auf Objekte


Listen sind eine **sequentielle Folge** von Objekten.

Zugriff auf Index

Indizes starten bei 0 (Wie auch bei strings)

```
list = ["python", "refugeeks", "I am a string!"]
```

`list[0]`



`list[2]`

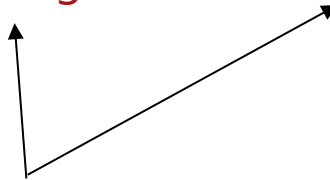


Slicing

Es kann auch [von:bis] eine Liste abgefragt werden

```
list = ["python", "refugeeks", "I am a string!"]
```

`list[1:2]`



Listen in Listen

Listen können auch selbst Listen enthalten

```
list = [[True, False, True], [25, 63, 23]]
```

`list[0][2]`



Aufgabe: Was steht in `list[1][1]`?



Listen verändern

Listen sind veränderbar (**mutable**)

```
list = ["python", "refugees", "I am a string!"]  
  
list[1] = 'Apple'  
print(list) # ["python", "Apple", "I am a string!"]
```

Listen sind kombinierbar oder können vervielfacht werden

```
a = [1, 6, 2]  
b = [5, 2, 6]  
c = a + b  
print(c) # [1, 6, 2, 5, 2, 6]  
d = c * 2  
print(d) # [1, 6, 2, 5, 2, 6, 1, 6, 2, 5, 2, 6]
```



Listen - Methoden

Auf einem Listenobjekt können verschiedene Hilfsmethoden ausgeführt werden

append() - Fügt der Liste ein neues Objekt an

```
list = ["python", "refugeeks", "I am a string!"]  
list.append('Apple')  
print(list) # ["python", "refugeeks", "I am a string!", "Apple"]
```

extend() - Anhängen einer Liste

```
a = [1, 6, 2]  
b = [5, 2, 6]  
a.extend(b)  
print(a) # [1, 6, 2, 5, 2, 6]
```



Listen - Methoden

Auf einem Listenobjekt können verschiedene Hilfsmethoden ausgeführt werden

pop() - Löschen von Elementen an bestimmtem Index

- Gibt zusätzlich das gelöschte Objekt zurück
- Ohne Index wird das letzte Element gelöscht

```
list = ["a", "b", "c"]  
list.pop(1)  
print(list) # ["a", "c"]
```

remove() - Löscht **ein** bestimmtes Element in der Liste

```
list = ["a", "b", "c"]  
list.remove("c")  
print(list) # ["a", "b"]
```



Listen - Methoden

Auf einem Listenobjekt können verschiedene Hilfsmethoden ausgeführt werden

del() - Löschen mehrerer Elemente der Liste

- Keine Listenmethode!

```
list = ["a", "b", "c", "d"]  
del(list[1:3])  
print(list) # ["a", "d"]
```

sort() - Sortiert die Liste

- Funktioniert nur mit Listen mit einem Datentypen

```
list = [5, 3, 1]  
list.sort()  
print(list) # [1, 3, 5]
```



Listen - Methoden

Auf einem Listenobjekt können verschiedene Hilfsmethoden ausgeführt werden

len() - Gibt die Länge der Liste zurück

- Geht auch für Strings

```
list = [5, 3, 1]  
print(len(list)) #3
```



Element in Liste prüfen

Um herauszufinden ob sich ein bestimmtes Element in einer Liste befindet kann der Operator „**in**“ verwendet werden. Die Logik kann mit „**not**“ umgedreht werden, um zu prüfen ob sich ein Element **nicht** in der Liste befindet.

```
>>> 3 in my_list
True
>>> 5 in my_list
False
>>> 5 not in my_list
True
>>>
```



Logische Operatoren

Mit logischen Operatoren können Wahrheitsvergleiche kombiniert werden.

Der **and** Operator wird als „*und*“ verstanden.
Es müssen beide Seiten des Vergleichs Wahr sein.

Der **or** Operator wird als „*oder*“ verstanden.
Es muss eine der beiden Seiten des Vergleichs Wahr sein.

```
>>> a = 50
>>> b = 25
>>> a > 40 and b > 40
False
>>> a > 100 and b < 50
False
>>> a == 0 and b == 0
False
>>> a > 0 and b > 0
True
>>> a > 0 or b < 0
True
>>> a > 0 or b > 0
True
>>> b < 0 or b < 0
False
```



Aufgaben 3

1. Schreibt ein Programm das drei Eingaben akzeptiert und bestimmte Buchstaben in einem Wort mit einem Zeichen übersetzt.

```
Bitte gib ein Wort ein: Test
Welcher Buchstabe soll ersetzt werden?: e
Womit soll der Buchstabe ersetzt werden?: *
Das neue Wort ist: T*st
```

Tipp: Die Lösung für diese Aufgabe ist noch ohne Listen.



Aufgaben 3

2. Schreibt ein Programm das das größte Element in der folgenden Liste zurückgibt.

```
my_list = [23, 52, 5334, 45, 11, 3, 76]
```



Aufgaben 3

3. Schreibt ein Programm welches alle doppelten Werte in einer Liste entfernt.



Funktionen - Motivation

Größere Programme müssen strukturiert geschrieben werden um wartbar und verständlich zu bleiben

- Funktionen fassen mehrere Anweisungen zusammen
- Funktionen erlauben wiederverwendung von Code
- Setzen das mathematische Konzept $f(x) = x^2$ um
 - Aufruf von Funktion gibt einen bestimmten Wert (Objekt) zurück



Funktionen – built-in Funktionen

Python besitzt viele eingebaute Funktionen

- Bereits einige bei Listen kennengelernt
- Weitere built-in Funktionen:

Länge eines Strings ausgeben

```
print(len("string")) # 6
```

Zahlen runden

```
print(round(0.12345, 2)) # 0.12
```

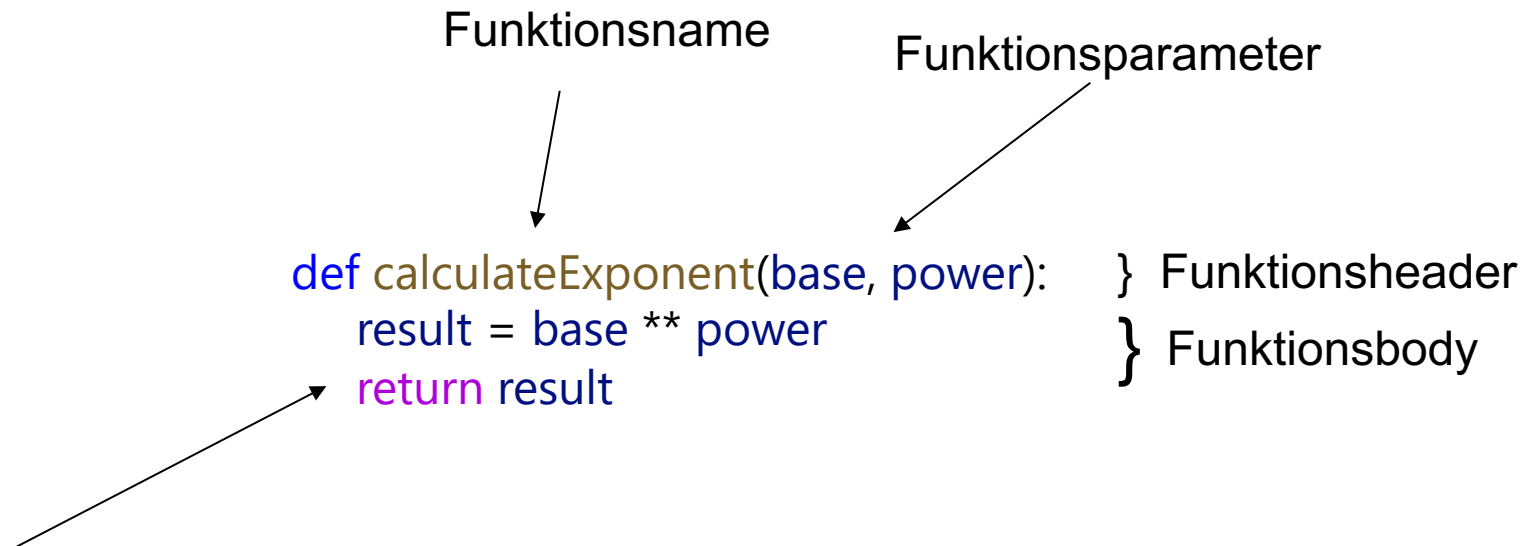
Datentypen konvertieren

```
print(float('3.14')) # 3.14 als Zahl
```



Funktionen selbst definieren

Funktionen bestehen aus einem Funktionsheader und einem Funktionsbody



Return beendet die Funktion

- Optional
- Ohne return wird None zurückgegeben



Funktionen - Seiteneffekte

Ein Seiteneffekt tritt auf wenn eine Funktion außerhalb des eigenen Scopes Variablen verändert

- Variablen innerhalb einer Funktion (also dem Scope) existieren nur dort

Verwendung von unveränderlichen Datentypen ist frei von Seiteneffekten

```
def double(x):  
    x = x * 2
```

```
y = 3  
double(y)
```

```
print(y) # bleibt 3
```

Dies gilt aber nicht für Listen oder Klassenobjekte

```
list = [2, 4]
```

```
def doubleList():  
    for i in range(len(list)):  
        list[i] = list[i] * 2
```

```
doubleList()
```

```
print(list) # [4, 8]
```



Aufgaben 4

1. Refactoring: Wandelt die Aufgabe zur Berechnung des BMI in folgendes Programmdesign mit drei Funktionen um:

- Eine Funktion, die aus Gewicht und Größe den BMI zurückliefert.
- Eine zweite Funktion die die Benutzerschnittstelle realisiert, also den Benutzer zur Eingabe auffordert.
- Eine dritte Funktion, die die Interpretation des BMI (zu schwer, zu leicht,..) ausgibt.



Aufgaben 4

2. Schreibt eine Funktion `middle()`, die für eine Liste ihr erstes und ihr letztes Element entfernt und die neue Liste zurückgibt.

`middle([1,2,3,4])` liefert also `[2,3]`.



Aufgaben 4

3. Schreibt eine Funktion *greatestChar(input)*, die den größten Buchstaben zurückliefert.

Also: *greatestChar ('aehiozehi')* liefert 'z' zurück.



Aufgaben 4

4. Schreibt eine Funktion *flatten(list)*, welche eine verschachtelte 2-Dimensionale Liste in eine eindimensionale Liste umwandelt.

```
def flatten(list):  
    ...
```

```
two_d_list = [[24, 23, 1, 32, 2, 3], [29, 64, 4, 1, 9]]  
...
```



```
[24, 23, 1, 32, 2, 3, 29, 64, 4, 1, 9]
```



Aufgaben 4

4. Refactoring: Wandelt die Aufgabe zur Berechnung der geometrischen Flächen selbstständig in eine Programmdesign mit Funktionen um.



Aufgaben 4

5. Schreibt eine Funktion, welche eine Liste von Wörtern annimmt und das längste Wort und die Länge des längsten Wortes zurückgibt.



Aufgaben 4

6. Schreibt eine Funktion, welche alle Werte außer Integer Werten aus einem gegebenen Array mit gemischten Werten zu entfernen.

```
random_list = [23.83, 62, -3.8, 'Python', 0, 'Refugeeks', 26]
```



Aufgaben 4

7. Schreibt eine Funktion, welche die Indizes der Elemente einer gegebenen Liste findet, die größer als ein bestimmter Wert sind.

