

Objektorientierung und Funktionale Programmierung

Einführung

Kristof Van Laerhoven
kvl@eti.uni-siegen.de

Michael Moeller
Michael.moeller@uni-siegen.de

Andreas Hoffmann
Andreas.Hoffmann@uni-siegen.de
uni-siegen.de



Wichtiger Hinweis:
Sollte etwas aus der Vorlesung in dem
Überblick fehlen, bedeutet dies NICHT,
dass dies nicht klausurrelevant ist!

Das Lernen anhand dieses Überblicks
alleine reicht nicht!

Vorlesung 1a: Erste Schritte + einfache Variablen

Erste Einführung: `print`, `input`, einfache strings hierfür, einfache Zuweisungen zu Variablen, Typen `int`, `float`, `bool`, `string`.

```
name = input("Wie lautet Dein Name?")  
print( --- LÜCKE --- )
```

- a) Schreiben sie Code für die --- LÜCKE ---, welcher es erlaubt die Person namentlich mit „Hallo“ zu begrüßen und ein Ausrufungszeichen anzuhängen.
- b) Was können sie tun wenn der Name in der Ausgabe in Anführungszeichen stehen soll? Schreiben sie Code hierfür.

```
a = 1  
A = 1.0  
print(type(a)==type(A))
```

Was ist der Unterschied zwischen dem Zuweisungsoperator `=` in Python und dem mathematischen Gleichheitszeichen? Erklären sie anhand eines Beispiels.

Was passiert beim Ausführen dieses Codes?

- a) Es gibt einen Fehler
- b) Es wird True ausgegeben
- c) Es wird False ausgegeben

Vorlesung 1b: Typen von Variable und Befehle um mit ihnen umzugehen

`+`, `*`, `len`, `[a:b:c]`, `split`, `str(42)`, `.format` und Platzhalter mit und ohne Namen, `.count`, `.replace`, `.find`, für strings, die – wie besprochen und später detailliert – immutable sind.

```
satz = "Wenn Robben hinter Robben robben, robben Robben Robben nach."  
def zähle_anzahl_des_wortes_im_string(text, wort):  
    --- LÜCKE ---  
    return anzahl  
  
print(zähle_anzahl_des_wortes_im_string(satz, "robben"))
```

- Füllen sie die Lücke in obigem Programm so, dass die Häufigkeit vom im Eingabeargument „wort“ gespeicherten String im String „text“ gezählt und die Anzahl zurückgegeben wird.
- Welche Zahl gibt der print-Befehl aus?

Vorlesung 1b: Typen von Variable und Befehle um mit ihnen umzugehen

`int` und `float` als Typen für Zahlen auf denen wir Rechenoperationen ausführen können.

Operator	Beschreibung	Beispiel	Resultat
+	Addition zweier Zahlen	3+2	5
-	Subtraktion zweier Zahlen	3-2	1
*	Multiplikation zweier Zahlen	3*2	6
/	Division zweier Zahlen	3/2	1.5
//	Ganzzahlige Division	3//2	1
%	Modulus (Rest der Division)	5%3	2
**	Exponent (linker Wert hoch rechter Wert)	3**2	9

```
a = 1
b = a / 3
print(type(b))
```

Was passiert hier?

- a) Es gibt einen Fehler
- b) Es wird <class 'int'> ausgegeben
- c) Es wird <class 'float'> ausgegeben

Schreiben sie ein Programm, das den Benutzer nach einem Radius fragt und den Flächeninhalt des zugehörigen Kreises ausgibt.

Vorlesung 1c: Erste Kontrollstrukturen

if, else, elif, while um Programme kontrollieren zu können und Anweisungen wiederholen zu können

Operator	Beschreibung	Beispiel	Resultat
==	Sind zwei Werte gleich?	3==3	True
!=	Sind zwei Werte ungleich?	3!=3	False
>	Ist der linke Wert echt größer als der rechte?	3>3	False
<	Ist der linke Wert echt kleiner als der rechte?	3<3	False
>=	Ist der linke Wert größer oder gleich dem rechten?	3>=3	True
<=	Ist der linke Wert kleiner oder gleich dem rechten?	3<=3	True

Operator	Beschreibung	Beispiel	Resultat
and	True nur wenn beide Werte True sind.	True and False	False
or	True wenn mindestens einer der Werte True ist.	True or True	True
not	True wenn der Wert False ist.	not False	True

Vorlesung 1c: Erste Kontrollstrukturen

`if`, `else`, `elif`, `while`, `for`, `range`, `enumerate`, um Programme kontrollieren zu können und Anweisungen wiederholen zu können; for-Schleifen auf iterierbaren Objekten

```
a=0
while (--- LÜCKE 1 ---):
    if (--- LÜCKE 2 ---):
        print(a)
```

Vervollständigen sie die beiden Lücken so, dass alle ohne Rest durch 3 teilbaren Zahlen kleiner als 100 ausgegeben werden.

```
a=5

if a>0:
    print('Größer 0!')
elif(a>2):
    print('Größer 2!')
else:
    print('Kleiner 0!')
```

Ein Kollege von Ihnen hat den folgenden Code geschrieben und sie um Hilfe bei der Fehlersuche gebeten. Was ist hier falsch implementiert? Was gibt der Code im Beispiel aus?

Vervollständigen sie den folgenden Code

```
symbols = ["scissors", "paper", "rock", "lizzard", "spock"]  
verbs = ["cuts", "gets crushed by", "decapitates", "gets smashed by",  
         "covers", "gets eaten by", "disproves",  
         "crushes", "gets vaporized by",  
         "poisons"]  
### Hier kommt ihr Code hin ###
```

um mittels zweier Schleifen die folgende Ausgabe zu erzeugen

```
scissors cuts paper  
scissors gets crushed by rock  
scissors decapitates lizzard  
scissors gets smashed by spock  
paper covers rock  
paper gets eaten by lizzard  
paper disproves spock  
rock crushes lizzard  
rock gets vaporized by spock  
lizzard poisons spock
```

Vorlesung 1d: Listen (mutable)

append, remove, insert, pop, und Indizierung

```
def runde_ungerade_zahlen_auf(liste):  
    for i in range(len(liste)):  
        if (--- LÜCKE 1 ---):  
            --- LÜCKE 2 ---
```

- Vervollständigen sie die beiden Lücken so, dass alle ungeraden Zahlen in der Eingabeliste ,liste' durch die nächst größere gerade Zahl ersetzt werden. Sie dürfen davon ausgehen, dass der Benutzer nur natürliche Zahlen als Elemente der Liste wählt.
- Die obige Funktion hat keine Rückgabe definiert. Macht diese Funktion so überhaupt Sinn? Begründen sie ihre Antwort!

Ähnlich (mit einer vorgegebenen Struktur): Entfernen sie die erste ungerade Zahl in einer Liste.

Mit welchem Befehl kann man das letzte Element einer Liste zurückgeben und dieses gleichzeitig aus der Liste löschen? Beschreiben sie in zwei Sätzen ein Anwendungsbeispiel.

Vorlesung 2: Funktionen

`def`, `return`, `#`, `"""`, docstring, Eingabeargumente (*positional* und *keyword*), `*` für variable Anzahl an positional Argumenten, `**` für variable Anzahl an keyword Argumenten, `import` für Funktionen in ausgelagerten Dateien, namespaces

Schreiben sie eine Beispielfunktion mit mindestens einem positional und einem keyword Argument. Ihre Funktion soll mit einem docstring beginnen und als Rückgabe nur das positional Argument liefern.

Vorlesung 2: Funktionen

Funktionen in Variablen speichern, Funktionen höherer Ordnung (=Funktionen die als Eingabe oder Ausgabe Funktionen haben). Dies ist ein erstes wichtiges Konzept welches funktionale Programmierung vereinfacht

```
def meine_rechenoperation(x,y):  
    return x+y  
  
def rechenoperation_anwenden(x,y,rechenoperation):  
    return --- LÜCKE 1 ---  
  
a = --- LÜCKE 2 ---  
  
rechenoperation_anwenden(1,5,a)
```

Füllen sie Lücke 1 so aus, dass die Rechenoperation auf x und y angewendet wird. Füllen sie Lücke 2 so aus, dass die letzte Zeile zu einem Anwenden von meine_rechnoperation auf 1 und 5 führt.

Vermeiden sie die Definition von meine_rechenoperation indem sie in Lücke 2 die gleiche Funktion mittels der lambda Notation definieren

Schreiben sie eine Funktion, die eine Funktion als Rückgabewert liefert.

Vorlesung 3: Gültigkeitsbereiche, Mutability, erste funktionale Schritte

Veränderbare (z.B. list, dict, set, selbstgeschriebene Klassen) vs. unveränderbare (z.B. int, float, bool, string, tuple)

Datentypen in Python, Gültigkeitsbereiche und Lebensdauer, **global**,

```
t1 = "1,2,3,4,5"  
t2 = [1,2,3,4,5]  
t3 = (1,2,3,4,5)
```

Für welches der t1, t2, t3 wäre ein Aufruf der Form t1[2] = 0 ohne Fehler möglich? Woran liegt dies?

```
a = 1  
def geht_das():  
    print(a)  
geht_das()
```

```
a=1  
def was_passiert_hier1():  
    a = 5  
was_passiert_hier1()
```

```
a = [1]  
def was_passiert_hier2():  
    a[0] = 5  
was_passiert_hier2()
```

Vorlesung 3: Gültigkeitsbereiche, Mutability, erste funktionale Schritte

Pure functions und funktionale Programmierung

Wir sollten unsere Programme (soweit es geht) nur durch *pure functions* aufbauen, d.h. durch Funktionen die

- 1) *nur von ihren Eingabeargumenten abhängen* und
- 2) *keine Nebeneffekte haben! Ihr einziger Effekt ist die Rückgabe von Werten.*

- (Code) Ist diese Funktion pure?
- (Funktionaler Code) Schreiben Sie diesen Code imperativ (mit for- und while Schleifen ohne Funktionen).
- (Imperativer Code) Schreiben Sie diesen Code funktional. (Schrittweise, z.B. Definition von Funktionen, Rekursion statt Schleifen, states zum Speichern des Spielstandes)

Vorlesungen 4+5: Einige Techniken für funktionales Programmieren

Programme umschreiben um sie aus pure functions aufzubauen; **map**, **filter**, **reduce**, Nebeneffekte an einer Stelle sammeln; state eines Programms in einem dictionary speichern, alle states als Tupel merken, Rekursion

```
summe = 0
liste = [1,2,3,4,5,6,7]

def addiere_zahl_hinzu(summand):
    global summe
    summe += summand

for el in liste:
    if (el%2==0):
        addiere_zahl_hinzu(el)

print(summe)
```

Schreiben sie eine rekursive pure Funktion **summiere_gerade_zahlen** sodass das Programm auf der linken Seite durch den Aufruf

```
print(summiere_gerade_zahlen((1,2,3,4,5,6,7)))
```

Nutzen sie die Befehle **filter** und **reduce** um das Programm auf der linken Seite in einer einzigen (langen) Zeile schreiben zu können.

```
def summiere_gerade_zahlen(tupel):  
    if tupel:  
        if (tupel[-1]%2==0):  
            return tupel[-1] + summiere_gerade_zahlen(tupel[0:-1])  
        else:  
            return summiere_gerade_zahlen(tupel[0:-1])  
    else:  
        return 0  
  
print(summiere_gerade_zahlen((1,2,3,4,5,6,7)))  
  
from functools import reduce  
print(reduce(lambda x,y: x+y, filter(lambda x: x%2==0, (1,2,3,4,5,6,7))))
```

Wichtige Klausur-Übungstipps

Üben Sie kleine Spiele zu programmieren:

- Kniffel-Light (Yahtzee Mini), vereinfachte Regeln:
 - Es wird mit 5 Würfeln geworfen.
 - Der Spieler hat 3 Versuche, um einen Dreierpasch zu würfeln.
 - Nach jedem Wurf kann er einzelne Würfel festhalten.
 - Ziel: Möglichst hohe Punktzahl (nur *Dreierpasch*-Punkte zählen)
- Mäxchen/Meiern mit einem Würfel und zwei Spielern
 - Ein Spieler Würfelt verdeckt und nennt dem nächsten Spieler eine Zahl
 - Der nächste Spieler kann die Zahl glauben und selbst weiter würfeln. In diesem Fall muss er als nächstes eine Zahl sagen, die größer ist, als die, die der Spieler vor ihm hatte.
 - Alternativ kann er die Zahl anzweifeln. In diesem Fall wird die wahre Zahl genannt. War die Zahl eine Lüge (sodass zurecht angezweifelt wurde), bekommt er einen Punkt. War die Zahl keine Lüge, bekommt der Gegner einen Punkt.
- Hohe Hausnummer
 - Es wird drei Mal mit einem Würfel geworfen.
 - Ziel ist es aus den drei Würfeln eine möglichst große dreistellige Zahl zusammensetzen. Varianten: 1) Die beste Zahl wird automatisch zusammengesetzt. 2) Nach jedem Wurf muss der Spieler sagen ob die gewürfelte Zahl an die Einer-, Zehner- oder Hunderterstelle geschrieben werden soll.
 - Wer am Ende die höchste Zahl hat, gewinnt.

Üben Sie kleine Spiele zu programmieren:

- Galgenmännchen:
 - Man versucht Buchstaben in einem unbekannten Wort (bekannter Länge) zu raten.
 - Ist der Buchstabe tatsächlich im Wort, wird es an den Stellen, an denen der Buchstabe im Wort vorkommt, eingetragen. Dies gibt einen Hinweis dazu, was das Wort sein könnte.
 - Ist der Buchstabe nicht im Wort, zählt man die falschen Versuche hoch. Beim 6. Fehler endet das Spiel und man hat verloren.
- Runterzählen von 101.
 - Man hat 101 Punkte und versucht auf (unter) 0 zu kommen.
 - Das Spiel verläuft in Runden. In jeder Runde beginnt man mit 0 Rundenpunkten und würfelt.
 - Wirft man eine 1, endet die Runde und die Rundenpunkte werden auf 0 zurückgesetzt.
 - Wirft man etwas anderes als eine 1, wird der Würfelwurf zu den Rundenpunkten hinzuaddiert.
 - Nach jedem Wurf kann man sich entscheiden die Runde freiwillig zu beenden. In diesem Fall werden die Rundenpunkte von den Gesamtpunkten abgezogen.
- Kooperationsspiel
 - Jeder Spieler kann jede Runde entscheiden zu kooperieren (k) oder den anderen Spieler auszunutzen (a). Die Entscheidungen werden verdeckt getroffen und anschließend gemeinsam offen gelegt.
 - Wenn beide kooperieren, bekommen beide 3 Punkte. Kooperiert einer und der anderen wählt “ausnutzen”, so bekommt (k) 0 Punkte und (a) 5 Punkte. Wenn beide “ausnutzen” bekommen beide 0 Punkte.

Danke für Ihre Aufmerksamkeit!

Welche Fragen sind offen geblieben?