

Einführung in die Programmierung

Ronja Düffel
WS2019/20

26. September 2019

Rückblick

- Datentypen:
 - `bool`
 - Zahlen (`int` und `float`)
 - `string`
- Variablen
- Kontrollstrukturen
 - Verzweigungen (`if...:` und `if...else:`)
 - Schleifen (`while...:` und `for...:`)

Aggregierte Datentypen

Datentyp: Zusammenfassung von Objektmengen und der darauf definierten Operationen

z.B. die elementaren Datentypen `bool`, `int`, `float`

aggregierter Datentyp: Aus anderen (elementaren oder aggregierten) Datentypen zusammengesetzter Datentyp.

- um mehrere Daten die zusammengehören zu verwalten
- in Python gibt es vordefinierte aggregierte Datentypen
z.B. `set`, `tupel`, `dict`, `list`

Listen

- mehrere Werte unter einem Variablennamen zusammengefasst
- werden durch eckige Klammern [] angezeigt, Listenelemente werden durch Komma getrennt.

```
>>> liste_1 = [2,3,4,6,10]
>>>
>>> liste_2 = ['Joe', 'Jack', 'Alice']
>>>
>>> liste_3 = [0.5, 3, 'Blumen', 7.34]
```

- auf einzelne Werte kann über den Index zugegriffen werden

```
>>> liste_1[0]
2
>>> liste_3[-2]
'Blumen'
>>> liste_1[1]
3
>>>
```

Listen

Operator/ Funktion	Beschreibung
<list>[x]	Zugriff auf Element mit Index x
<list>[x:y]	Zugriff auf Teilliste von Index x bis y-1 (!!!)
<list> + <list>	zusammenfügen von Listen
<list>.append(x)	hinzufügen von x

```

>>> liste_1[1:4]
[3, 4, 6]
>>> liste_2 + liste_3
['Joe', 'Jack', 'Alice', 0.5, 3, 'Blumen', 7.34]
>>> liste_1.append('Bob')
>>> liste_1
[2, 3, 4, 6, 10, 'Bob']
>>>

```

Listen

<code>del <list>[x]</code>	löschen von Element mit Index x
<code><list>.remove(x)</code>	löschen von Element x
<code>len(<list>)</code>	Länge der Liste

```
>>> liste_4=[1,5,3,2,5]
>>> del liste_4[3]
>>> liste_4
[1, 5, 3, 5]
>>> liste_4.remove(5)
>>> liste_4
[1, 3, 5]
>>> len(liste_4)
3
>>> |
```

Funktionen

für Operationen die immer wieder gebraucht werden

- + Wiederverwertbarkeit
- + leichte Wartbarkeit
- + nur einmal schreiben
- + leicht auszutauschen
- + Übersichtlichkeit

Funktionen

- werden mit `def`-Anweisung definiert, Übergabeparameter in runden Klammern () dahinter
- Funktionsrumpf muss eingerückt sein
- Ende der Funktion durch beenden der Einrückung
- Schlüsselwort `return` beendet die Funktion und veranlasst Zuweisung des Rückgabewerts

```
1 def add(a,b):  
2     '''Addiere die Zahlen a und b'''  
3     return a+b
```

- `return`-Anweisung ist optional. Hat die Funktion keinen Rückgabewert, so wird das Objekt `None` zurückgegeben.

Datentyp None

- hat nur einen einzigen Wert: `None`
- Schlüsselwort, dient als Platzhalter für Variablen, die keinen Wert haben oder deren Wert noch nicht bekannt ist
- Bei Auswertung von Ausdrücken in der Python-Shell wird nur etwas ausgegeben, wenn der Rückgabewert nicht `None` ist

Beispiel None

```
>>> empty = None
>>> empty # keine Ausgabe!
>>> print(empty)
None
```

Funktionen

- Funktionsdefinition muss im Code (lexikalisch) **vor** dem Aufruf erfolgen
- Übergabeparameter müssen beim Aufruf in der richtigen Reihenfolge angegeben werden

Beispiel: Parameterübergabe

```
1 def getNewBalance (now, spent, name):
2     amount = now - spent
3     owner = name
4     return ((owner, amount))
5
6 balance = getNewBalance('Bob', 500, 150)
7 print(balance)
```

```
Traceback (most recent call last):
  File "/media/ronja/Elements/WS1718/Material/Folien/prameter.py", line 6, in <module>
    balance = getNewBalance('Bob', 500, 150)
  File "/media/ronja/Elements/WS1718/Material/Folien/prameter.py", line 2, in getNewBalance
    amount = now - spent
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>>
```

Parameter benennen

```
1 def getNewBalance (now, spent, name):
2     amount = now - spent
3     owner = name
4     return ((owner, amount))
5
6 balance = getNewBalance(name='Bob', now=500, spent
7     =150)
8 print(balance)
```

```
('Bob', 350)
>>> |
```

Fehlersuche

- mit `type()` kann man sich den Datentyp einer Variablen ausgeben lassen

```
>>> zahl = '5'  
>>> type(zahl)  
<class 'str'  
>>>
```

Beispiel type()

```
1 side = input('Seitenlänge in cm: ')
2 area = side**2
3 print('Flächeninhalt:',area)
```

```
1 side = input('Seitenlänge in cm: ')
2 print('debug:',side, type(side))
3 area = side**2
4 print('Flächeninhalt:',area)
```

```
Seitenlänge in cm: 6
debug: 6 <class 'str'>
Traceback (most recent call last):
  File "/home/ronja/Uni/Lernzentrum/Vorkurs/WS1718/Material/Folie
n/typeErrorFind1.py", line 3, in <module>
    area = side**2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and
'int'
>>> |
```

Gültigkeitsbereiche

```
1 # Gueltigkeitsbereiche
2
3 x = 5
4
5 def sowas():
6     x = 0
7     print ('Wert von x in der Funktion:',x)
8     return
9
10 print('Wert von x vor Funktionsaufruf:',x)
11 sowas()
12 print('Wert von x nach Funktionsaufruf:', x)
```

```
Wert von x vor Funktionsaufruf: 5
Wert von x in der Funktion: 0
Wert von x nach Funktionsaufruf: 5
```

```
>>>
```


Gültigkeitsbereiche

```
1 # Gueltigkeitsbereiche
2
3 x = 5
4
5 def sowas():
6     global x
7     x = 0
8     print ('Wert von x in der Funktion:',x)
9     return
10
11 print('Wert von x vor Funktionsaufruf:',x)
12 sowas()
13 print('Wert von x nach Funktionsaufruf:', x)
```

Wert von x vor Funktionsaufruf: 5

Wert von x in der Funktion: 0

Wert von x nach Funktionsaufruf: 0

Gültigkeitsbereiche

```
1 # Gueltigkeitsbereiche
2 x = 5
3 y = 2
4 def sowas():
5     global x
6     x = 0
7     z = x - y
8     print ('Wert von x in der Funktion:',x)
9     print('Wert von z in der Funktion:', z)
10    return
11 print('Wert von x vor Funktionsaufruf:',x)
12 sowas()
13 print('Wert von x nach Funktionsaufruf:', x)
```

Wert von x vor Funktionsaufruf: 5
Wert von x in der Funktion: 0
Wert von z in der Funktion: -2
Wert von x nach Funktionsaufruf: 0

Gültigkeitsbereiche

```
1 # Gueltigkeitsbereiche
2 x = 5
3 y = 2
4 def sowas():
5     global x
6     x = 0
7     z = x - y
8     print ('Wert von x in der Funktion:',x)
9     print('Wert von z in der Funktion:', z)
10    return
11 print('Wert von x vor Funktionsaufruf:',x)
12 sowas()
13 print('Wert von x nach Funktionsaufruf:', x)
14 print('Wert von z nach Funktionsaufruf:',z)
```

Ausgabe

```
Wert von x vor Funktionsaufruf: 5
Wert von x in der Funktion: 0
Wert von z in der Funktion: -2
Wert von x nach Funktionsaufruf: 0
Traceback (most recent call last):
  File "/home/ronja/Uni/Lernzentrum/Vorkurs/WS1920/Material/Folien/g
  line 17, in <module>
    print('Wert von z nach Funktionsaufruf:',z)
NameError: name 'z' is not defined
>>>
```

Gültigkeitsbereiche

- Variablenname ist in dem Anweisungsblock gültig, in dem er definiert wird.
- unterscheide zwischen *lokalen* (innerhalb Block/Funktion) und *globalen* (auch außerhalb) Variablen
- Verwendung globaler Variablen innerhalb von Funktionen mit `global`

Wiederverwendung von Funktionen in anderen Programmen :

- `import`
 - `import <Modulname>` (Dateiname ohne `.py`)
Verwendung durch `<Modulname>.<Funktionsname>`
(kein Namenskonflikt)
 - `from <Modulname> import <Funktionsname/n>`
Verwendung durch `<Funktionsname>`
(!gleichnamige Funktionen werden überschrieben!)
 - `from <Modulname> import *`
Alles wird importiert, gefährlich aber "bequem"

Beispiel import

Datei: rechnen.py

```
1 def add(a,b):
2     '''Addiere die Zahlen a und b'''
3     return a+b
4
5 def summe(n):
6     '''berechnet die Summe der ersten n natürlichen
7         Zahlen'''
8     ergebnis = 0
9     for i in range(n+1):
10        ergebnis += i
11    return ergebnis
```

Beispiel import

Programm, mit dem die Summe der ersten n natürlichen Zahlen berechnet werden kann

```
1 import rechnen # importiere Modul rechnen
2
3 while True:
4     a = input("Geben Sie eine Ganzzahl > 0 ein: ")
5     if a.isdigit():
6         break
7
8 a_int = int(a)
9 result = rechnen.summe(a_int)
10 print("Die Summe von 1 bis", a , "ist:", result)
```

```
..
Geben Sie eine Ganzzahl > 0 ein: 5
Die Summe von 1 bis 5 ist: 15
>>>
```


Beispiel import *

```

1 from rechnen import * # importiere alles von Modul
  rechnen
2 def summe(a,b):
3     return(a + b)
4
5 while True:
6     a = input("Geben Sie eine Ganzzahl > 0 ein: ")
7     if a.isdigit():
8         break
9 a_int = int(a)
10 result = summe(a_int)
11 print("Die Summe von 1 bis", a ,"ist:",result)

```

Geben Sie eine Ganzzahl > 0 ein: 5

Traceback (most recent call last):

File "/home/ronja/Uni/Lernzentrum/Vorkurs/WS1920/Material/Folien,
y", line 13, in <module>

result = summe(a_int)

TypeError: summe() missing 1 required positional argument: 'b'

>>>

Beispiel import *

```
1 import rechnen # importiere Modul rechnen
2 def summe(a,b):
3     return(a + b)
4
5 while True:
6     a = input("Geben Sie eine Ganzzahl > 0 ein: ")
7     if a.isdigit():
8         break
9 a_int = int(a)
10 result = rechnen.summe(a_int)
11 print("Die Summe von 1 bis", a ,"ist:",result)
12 summe = summe(a_int,6)
13 print("Die Summe von", a ,"und 6 ist:",summe)
```

Geben Sie eine Ganzzahl > 0 ein: 5

Die Summe von 1 bis 5 ist: 15

Die Summe von 5 und 6 ist: 11

>>>

Dateien lesen und schreiben

`open()` : öffnet eine Datei in angegebenem Modus

- 'r': Lesemodus
- 'w': Schreibmodus !Datei wird überschreiben !
- 'a': Schreibmodus, neue Daten werden am Ende hinzugefügt

`read()` : Lese den Inhalt der Datei; komplett, oder die angegebene Anzahl an Bytes

`write()` : Schreibt Daten in Datei. Zeilenumbruch muss explizit angegeben werden

`close()` : schließt Datei.

Beispiel

```
1 # Dateizugriff
2
3 fo = open("/home/ronja/Demo/beispiel.txt","w") #Datei
   öffnen zum Schreiben
4 fo.write("Nun zieren Sie sich doch nicht so! \n")
5 fo.write("""Sie haben doch einen Spenderausweis.
6
7 Aber das ist doch nur im Todesfall!!!""") #mehrzeilig
   ohne \n-Angabe
8 fo.close() #Datei schliessen
```

Datei beispiel.txt

```
1 Nun zieren Sie sich doch nicht so!
2 Sie haben doch einen Spenderausweis.
3
4 Aber das ist doch nur im Todesfall!!!
```

Beispiel

```
1 fo = open("/home/ronja/Demo/beispiel.txt","w") #Datei  
    öffnen zum Schreiben  
2 fo.write("Keine Angst. Das hat noch keiner überlebt")  
3 fo.close()
```

Datei beispiel.txt

```
1 Keine Angst. Das hat noch keiner überlebt
```

Beispiel

```
1 fo = open("/home/ronja/Demo/beispiel.txt","a") #Datei  
    öffnen zum Schreiben  
2 fo.write("Keine Angst. Das hat noch keiner überlebt")  
3 fo.close()
```

Datei beispiel.txt

```
1 Nun zieren Sie sich doch nicht so!  
2 Sie haben doch einen Spenderausweis.  
3  
4 Aber das ist doch nur im Todesfall!!!Keine Angst. Das  
    hat noch keiner überlebt
```

Rekursion

*Um Rekursion zu verstehen,
muss man erstmal Rekursion verstehen*

- Methode etwas durch sich selbst zu definieren

Beispiel (Summe)

Die Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ sei gegeben durch

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

*Rekursionsanfang
Rekursionsschritt*

rekursive Programmierung

- Funktionen die sich selbst aufrufen (auch verschachtelt)
- Abbruchbedingung muss auch erreicht werden (Gefahr der Endlosschleife)

```
1 # Summe rekursiv
2
3 def sum_rek(n):
4     if n==0:
5         return 0
6     else:
7         return n + sum_rek(n-1)
```


Beispiel Summe

```
1 # Summe iterativ
2
3 def sum_iter(n):
4     result = 0
5     for i in range(n+1):
6         result += i
7     return result
```

```
1 # Summe rekursiv
2
3 def sum_rek(n):
4     if n==0:
5         return 0
6     else:
7         return n + sum_rek(n-1)
```

Fragen?

?