



# Funktionen, Parameter, Attribute

# Python3 – Funktionen



Bislang trotz Kontrollstrukturen  
von oben nach unten

- Funktionen lagern Teil aus
- zur Gliederung
- wiederverwendbar

# Python3 – Funktionen



Bislang trotz Kontrollstrukturen  
von oben nach unten

- Funktionen lagern Teil aus
- zur Gliederung
- wiederverwendbar

```
1 def druckfunktion():  
2     print("Ausgabe")  
3  
4 print('Programmstart')  
5 druckfunktion()  
6 print('Programmende')
```

Ausgabe:  
Programmstart  
Ausgabe  
Programmende

# Python3 – Funktionen



Definition vs. Aufruf

Beispiele bisher

- `print()`
- `range()`

```
1 def druckfunktion():  
2     print("Ausgabe")  
3  
4 print('Programmmanfang')  
5 druckfunktion()  
6 print('Programmende')
```

Ausgabe:  
Programmmanfang  
Ausgabe  
Programmende

# Python3 – Funktionen



range liefert Wert zurück

- können eigene

Funktionen definieren

```
1 bereich = range(3,6)
2
3 for x in bereich:
4     print(x, end="")
```

Ausgabe:

345

# Python3 – Funktionen



## Funktionen

- haben Namen
- haben Schnittstellen  
(Parameter)
- liefern Werte zurück  
(Default: None)

# Python3 – Funktionen



## Funktionen

- haben Namen
- haben Schnittstellen  
(Parameter)
- liefern Werte zurück  
(Default: None)

```
1 x = print("333")  
2 print(x)
```

Ausgabe:

```
333  
None
```

# Python3 – Funktionen



## Funktionen -- Definition

- Schlüsselwort def

```
def funktionsname(parameter_1, ..., parameter_n):  
    Anweisung_1  
    Anweisung_2  
    ...
```

- Parameterliste kann leer sein
- Anweisungen wie sonst
- Parameter zugänglich über ihre Namen



# Python3 – Funktionen



## Funktionen -- Definition

```
def funktionsname(parameter_1, ..., parameter_n):  
    Anweisung_1  
    Anweisung_2  
    ...
```

```
1 def addiere_drei(a, b, c):  
2     summe = a + b + c  
3     print(summe)  
4  
5 addiere_drei(2, 3, 4)
```

Ausgabe:

9

# Python3 – Funktionen



## Funktionen -- Definition

- Der Wert von `summe` ist verloren

wenn Funktionsausführung beendet

```
1 def addiere_drei(a, b, c):  
2     summe = a + b + c  
3     print(summe)  
4  
5 addiere_drei(2, 3, 4)
```

Ausgabe:

9

# Python3 – Funktionen



## Funktionen -- Definition

- mit `return` wird summe

an Hauptprogramm gesendet:

```
1 def addiere_drei(a, b, c):  
2     summe = a + b + c  
3     return summe  
4  
5 print(addiere_drei(2, 3, 4))  
6
```

Ausgabe:

9

# Python3 – Funktionen



## Funktionen

- Rückgabe mit return besser im Sinne der Kapselung
- Ein-/Ausgabe und Berechnungen trennen
- allgemeiner wiederverwendbar

# Python3 – Funktionen



mehrere return möglich

- sobald eines ausgeführt: Abbruch

```
1 def addiere_drei(a, b, c):  
2     if a == 0:  
3         return None  
4     summe = a + b + c  
5     return summe  
6  
7 print(addiere_drei(2, 3, 4))  
8 print(addiere_drei(0, 3, 4))
```

Ausgabe:

9

None

# Python3 – Funktionen



mehrere return möglich

- statt "return None" auch nur "return"

```
1 def addiere_drei(a, b, c):  
2     if a == 0:  
3         return  
4     summe = a + b + c  
5     return summe  
6  
7 print(addiere_drei(2, 3, 4))  
8 print(addiere_drei(0, 3, 4))
```

Ausgabe:

9

None

# Python3 – Funktionen



mehrere Funktionen möglich

```
1 def betrag(zahl):  
2     if zahl < 0: return -zahl  
3     else: return zahl  
4  
5 def addiere_drei(a, b, c):  
6     summe = betrag(a) + betrag(b) + betrag(c)  
7     return summe  
8  
9 print(addiere_drei(2, 3, 4))  
10 print(addiere_drei(-2, 3, -4))
```

Ausgabe:

9

9

# Python3 – Funktionen



Was gibt folgendes Programm aus?

```
1 def betrug(zahl):  
2     if zahl > 5: return 4  
3     else: return 7  
4  
5 def addiere_drei(a, b, c):  
6     summe = betrug(a) + betrug(b) + betrug(c)  
7     return summe  
8  
9 print(addiere_drei(6, 4, 6), end=" ", "  
10 print(addiere_drei(3, 7, -123))
```

a) 18, 21      b) 15, -113      c) 15, 18      d) 7, 17



# Python3 – Funktionen



## Aufgabe

Schreiben Sie eine Funktion, die

- zwei Argumente bekommt,
- das kleinere vom größeren abzieht
- und das Ergebnis zurückliefert

# Python3 – Funktionen



## Funktionsobjekte

- Variable kann Funktion(sinstanz) aufnehmen

```
1 def betrug(zahl):  
2     if zahl > 5: return 4  
3     else: return 7  
4  
5 f = betrug  
6 print(f(2))
```

Ausgabe:

7

# Python3 – Funktionen



## Optionale Parameter

print():

- `end=""` und `sep=","`  
sind mal angegeben, mal nicht
- also **optional**
- auch bei selbst definierten Funktionen möglich

```
>>> print('a','B')
a B
>>> print('a','B',sep=" ")
a 'B
>>> print('a','B',sep=" ", end=" ")
a 'B>>>
```

# Python3 – Funktionen



## Optionale Parameter

- z.B. für Einstellungen
- entweder Default  
oder Eingabe
- Default angeben im Kopf

```
1 def anw(name, v="ist",  
2         d="heute", aw="da"):  
3     print(name, v, d, aw, sep=" ")  
4  
5 anw("Gerd")  
6 anw("Mia", aw="nicht da")  
7 anw("Sepp", "war", "gestern")
```

Ausgabe:

```
Gerd ist heute da  
Mia ist heute nicht da  
Sepp war gestern da
```

# Python3 – Funktionen



## Optionale Parameter

- Angabe des Namens  
nur nötig  
wenn "nach Lücke"

```
1 def anw(name, v="ist",  
2         d="heute", aw="da"):  
3     print(name, v, d, aw, sep=" ")  
4  
5 anw("Gerd")  
6 anw("Mia", aw="nicht da")  
7 anw("Sepp", "war", "gestern")
```

Ausgabe:

```
Gerd ist heute da  
Mia ist heute nicht da  
Sepp war gestern da
```

# Python3 – Funktionen



Was gibt folgendes Programm aus?

```
1 def sum(a, b, c=3, d=4):  
2     return a+b+c+d  
3  
4 print(sum(1,2,3),  
5       sum(1,2), sep=", ")
```

a) 10, 10      b) Syntax-Fehler      c) 6,3      d) 6, 3

# Python3 – Funktionen



## Aufgabe

Erstellen Sie eine Funktion, die

- einen normalen und einen optionalen Parameter nimmt
- die beiden Werte multipliziert und das Ergebnis zurückliefert
- ist der zweite Parameter nicht angegeben, wird der erste mit 2 multipliziert

# Python3 – Funktionen



## Schlüsselwortparameter

- beim Aufruf Übergabe mit Namen
- Zuordnung eindeutig

```
1 def sum(a, b, c, d):  
2     return a+b+c+d  
3  
4 print(sum(1,2,3,4))  
5 print(sum(c=3,b=2,a=1,d=4))
```



# Python3 – Funktionen



## Schlüsselwortparameter

- nicht als solche definiert
- Übergabeart

```
1 def sum(a, b, c, d):  
2     return a+b+c+d  
3  
4 print(sum(1,2,3,4))  
5 print(sum(c=3,b=2,a=1,d=4))
```

# Python3 – Funktionen



## Schlüsselwortparameter

- Mischung der Übergabearten möglich
- keine Positions- nach Schlüsselwortparametern
- nur solche, die nicht bereits übergeben

```
1 def sum(a, b, c, d):  
2     return a+b+c+d  
3  
4 print(sum(1,2,3,4))  
5 print(sum(1,2,d=3,c=4))
```

# Python3 – Funktionen



## Parameter variabler Anzahl

- Beispiel print
- beliebig viele Elemente zur Ausgabe
- auch in selbst definierten

```
1 print('a')  
2 print('a', 'b')  
3 print('a', 'b', 'c')  
4 print('a', 'b', 'c', 'd')
```

# Python3 – Funktionen



## Parameter variabler Anzahl

- in Tupel
- Stern vor Namen
- nur an letzter Stelle
- auch mit Schlüsselwort  
(dann Dictionary, s. Buch)

```
1 def mein_print(s, *mehr):  
2     print(s)  
3     for m in mehr:  
4         print(m)  
5  
6 mein_print("a", "b", "c")
```

Ausgabe:

```
a  
b  
c
```

# Python3 – Funktionen



## Reine Schlüsselwortparameter

- nach Parameter  
variabler Anzahl
- kann nur mit Namen  
übergeben werden

```
1 def mein_print(s, *mehr, ende=""):  
2     print(s, end=ende)  
3     for m in mehr:  
4         print(m, end=ende)  
5  
6 mein_print("a", "b", "c")
```

Ausgabe:  
abc

# Python3 – Funktionen



## Reine Schlüsselwortparameter

- nach Parameter variabler Anzahl
- kann nur mit Namen übergeben werden
- obligatorisch wenn nicht Default

```
1 def mein_print(s, *mehr, ende):  
2     print(s, end=ende)  
3     for m in mehr:  
4         print(m, end=ende)  
5  
6 mein_print("a", "b", "c", ende = "")
```

Ausgabe:

abc

# Python3 – Funktionen



Welche der folgenden Aufrufe führen zu Fehlern?

- a) `mein_print("text")`
- b) `mein_print("text1", "text2", "ende")`
- c) `mein_print("text1", ende = "ende")`
- d) `mein_print("text1", "text2", ende)`

```
1 def mein_print(s, *mehr, ende):  
2     print(s, end=ende)  
3     for m in mehr:  
4         print(m, end=ende)
```