

7. Funktionen

- Funktionen wie in der Mathematik: Abbildung von einem Definitionsbereich in einem Bildbereich.
- Funktion erwartet Argumente (aus dem Definitionsbereich) und gibt einen Funktionswert (=Rückgabewert) zurück.
- mehr wie in der Mathematik: Funktionen können Effekte haben, z.B. Ausgabe erzeugen, Eingabe lesen, etc.
- Viele Standardfunktionen in Python vordefiniert.

7.1 Konvertierungsfunktionen:

- `int`, `float`, `complex`, `str`: **passende** Werte in den Typ konvertieren.
- `chr`: konvertiert int in ein Unicode Zeichen. In Python sind Zeichen einstellige Strings.
- `ord`: konvertiert in die umgekehrte Richtung, Unicode in Char.

```
In [44]: int(-1.3)
```

```
Out[44]: -1
```

```
In [45]: int('vier')
```

```
-----  
ValueError                                Traceback (most recent call last)  
/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7_Funktionen/1-7_Funktionen.ipynb Cell 5 line 1  
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#W4sZmlsZQ%3D%3D?line=0'>1</a> int('vier')  
  
ValueError: invalid literal for int() with base 10: 'vier'
```

```
In [ ]: complex('42')
```

```
Out[ ]: (42+0j)
```

```
In [ ]: float(4)
```

```
Out[ ]: 4.0
```

```
In [ ]: str(42)
```

```
Out[ ]: '42'
```

```
In [ ]: chr(42)
```

Out[]: '*'

```
In [ ]: ord('*')
```

Out[]: 42

```
In [ ]: ord('HA')
```

```
-----  
TypeError                                Traceback (most recent call last)  
/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7_Funktionen/Funktionen.ipynb Cell 11 line 1  
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/Funktionen.ipynb#X32sZmlsZQ%3D%3D?line=0'>1</a> ord('HA')  
  
TypeError: ord() expected a character, but string of length 2 found
```

7.2 Ein- und Ausgabe-Funktionen:

- `print` : Gibt Wert aus
- `input` : liest Stringwerte ein.

```
In [ ]: input('Gib mir was ein :')
```

Out[]: 'Keks'

```
In [ ]: input (42)
```

Out[]: '42'

- Eingabe von Zahlen: Typkonvertierung notwendig

```
In [ ]: cm_in_m = 100  
länge = input("Eingabe der Länge in cm: ")  
#Länge ist ein String  
länge_cm = float(länge)  
länge_m = länge_cm * cm_in_m  
print(länge + 'cm', "=", str(länge_m) + 'm')
```

20cm = 2000.0m

7.3 Numerische bzw. mathematische Funktionen

- `abs` : Absolutbetrag, aber Vorsicht!
- `round` : kaufmännisches runden

```
In [ ]: abs(-2)
```

Out[]: 2

```
In [ ]: round(2.501)
```

```
Out [ ]: 3
```

Mathematische Funktionen stehen nicht direkt zur Verfügung, sondern müssen von einem Modul importiert werden.

```
In [ ]: import math
```

Im `math` Modul findet man viele Funktionen wie:

- Pi
- Exponentialfunktionen
- trigonometrische Funktionen

Funktionen aus dem Modul `math` können mit vorangestellten `math.` verwendet werden.

```
In [ ]: math.pi
```

```
Out [ ]: 3.141592653589793
```

```
In [ ]: math.exp(math.log(5))
```

```
Out [ ]: 4.999999999999999
```

```
In [ ]: math.sin(math.cos(2))
```

```
Out [ ]: -0.4042391538522658
```

Vorteile der Punkt-Schreibweise: Namenskollisionen werden verhindert

Der Bezeichner einer mathematischen Funktion kann direkt importiert werden: `from modulname import name`

Direkter Import aller Bezeichner eines Moduls: `from modul import *`

Bemerkung:

- Import kann nicht rückgängig gemacht werden.
- Ein Import kann aber überschrieben werden.

Beispiel:

```
In [ ]: from math import *
```

- Eine große Zuweisung
- alle lokalen Variablen werden mit den Definitionen in `math` überschrieben.

- lokale bezeichner können dann verwendet werden.
- Vorsicht: Bei Module update könnten lokale (eigene) Variablen plötzlich überschrieben werden, wenn zum Beispiel neue Definitionen hinzukommen.
- Besser diese Zuweisung vermeiden oder die Funktionen aufzählen, die man konkret haben möchte.

```
In [ ]: cos(pi)
```

```
In [ ]: from math import *
        cos(pi)
```

```
Out[ ]: -1.0
```

7.4 Eigene Funktionen definieren

7.4.1 Syntax und Eigenschaften

```
`def` Funktionsname (Paramterliste) `:`
    #Funktionsblock
    Anweisung 1 / Funktionsaufruf 1
    Anweisung 2 / Funktionsaufruf 2 ...
```

- Funktionsname muss gültiger Bezeichnern sein.
- Rumpf: alle Anweisungen sind **gleich weit** eingerückt.
- Funktionsdefinition ist auch eine Zuweisung an einen Bezeichner (analog zur Variable)

Beispiel:

```
In [ ]: def print_hymne():
        print("Einigkeit und Recht und Freiheit")
        print("Für das deutsche Vaterland")

        #print_nationalhymne()
```

- Einrückung und Anzahl der eingerückten Zeichen ist entscheiden in Python.
- Vier Leerzeichen pro Ebene notwendig
- keine Tabulatorzeichen machen! Editoren machen häufig aus der Tabulatortaste Leerzeichen.
- Anweisungen werden sequentiell ausgewertet. Damit müssen Funktionen, die verwendet werden sollen, vor der Anwendung definiert werden.
- andere Programmiersprache verwenden geschweifte Klammern für Blöcke

Beispiel: Funktionen innerhalb von Funktionen aufrufen

```
In [ ]: def repeat_hymne():
        print_hymne()
        print_hymne()

repeat_hymne()
```

Einigkeit und Recht und Freiheit
Für das deutsche Vaterland

7.4.2 Argumente

- häufig verwendet
- Funktionsdefinition verwendet **formale Parameter** (Variablennamen.)
- beim Funktionsaufruf erhalten die formalen Parameter die Argumentwerte
- `return` beendet die Ausführung der Funktion
- Wert des Ausdrucks nach `return` ist der Wert, den die Funktion zurückgibt.

Beispiel: Konverter Geschwindigkeit km/h in miles per hour

```
In [53]: KMH_PER_MIH = 1.6

def conv_kmh_in_mih(kmh):
    return kmh/KMH_PER_MIH

var1 = input('Gib die Geschwindigkeit in km pro Stunde ein')
conv_kmh_in_mih(int(var1))
```

Out[53]: 187.5

- Parameter sind nur innerhalb der Funktion sichtbar => **lokale Variable**

```
In [60]: def cat_words (part1, part2):
        cat = part1 + part2
        print (cat)

cat_words ('Hi ', 'Folks')

# Ein Print der Variable cat führt zur Fehlermeldung: name 'cat' ist not def
print(cat)
```

Hi Folks

```

-----
NameError                                Traceback (most recent call last)
/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7_Funktionen/1-7_Funktionen.ipynb Cell 41 line 8
      <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#X65sZmlsZQ%3D%3D?line=4'>5</a> cat_words ('Hi ', 'Folks')
      <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#X65sZmlsZQ%3D%3D?line=6'>7</a> # Ein Print der Variable cat führt zu einer Fehlermeldung
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#X65sZmlsZQ%3D%3D?line=7'>8</a> print(cat)

NameError: name 'cat' is not defined

```

7.4.3 Rückgabewerte

- Alle Funktionen geben einen Wert zurück.
- Die `print`-Funktion hat nur einen Effekt, gibt aber auch den Wert `None` zurück, der aber nicht sichtbar ist.

```

In [5]: ergebnis = print("Hallo")
        ergebnis
        print(ergebnis)

```

```

Hallo
None

```

- Schlüsselwort `return` definiert den Rückgabewert

```

In [7]: def summe(a, b, c):
        return a+b+c

        summe(1,2,3)

```

```

Out[7]: 6

```

```

In [11]: def printsumme(a,b,c):
        print (a+b+c)

        printsumme(1,2,3)

        #printsumme(1,2,3)+4

```

```

6
6

```

```

-----
TypeError                                Traceback (most recent call last)
/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7_Funktionen/1-7_Funktionen.ipynb Cell 45 line 6
      <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#Y100sZmlsZQ%3D%3D?line=1'>2</a>    print (a+b+c)
      <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#Y100sZmlsZQ%3D%3D?line=3'>4</a>    printsumme(1,2,3)
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funktionen/1-7_Funktionen.ipynb#Y100sZmlsZQ%3D%3D?line=5'>6</a>    printsumme(1,2,3)+4

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'

```

7.4.4 Scopes

- Rumpf einer Funktion bildet ein Scope (Umgebung)
- der Scope bindet die lokal definierten Variablen und Parameter, d.h. sie sind auch nur innerhalb der Funktion sichtbar und können auch nur dort verwendet werden.
- alle weiteren Variablen sind frei im Funktionsrumpf (freie Variablen) und beziehen sich nur auf den umschließenden Scope => global definierte Dinge können benutzt werden, zum Beispiel die Funktion `print`
- Beim Aufruf der Funktion wird ein stack frame (=Kellerrahmen) für die Werte der Variablen angelegt.
- Visualisierung des Kellerrahmens: <https://pythontutor.com/python-compiler.html#mode=edit>

Funktionen sollen:

- lokale Variablen und Parameter verwenden.
- globale Variablen lesen können.
- Vorsicht: shadowing (Überschreiben der Variablen)

```

In [2]: var1= 123

def deep():
    return var1

print("Die Funktion deep gibt zurück: ", deep())

def high(var1):
    return var1

print("Die Funktion high(42) gibt zurück: ", high(42))

def middle():
    var1=321
    return var1

```

```
print("Die Funktion middle gibt zurück: ", middle(), "und die Variable var1
```

Die Funktion deep gibt zurück: 123

Die Funktion high(42) gibt zurück: 42

Die Funktion middle gibt zurück: 321 und die Variable var1 hat den Wert: 1
23

In []: