7. Funktionen

- Funktionen wie in der Mathematik: Abbildung von einem Definitionsbereich in einem Bildbereich.
- Funktion erwartet Argumente (aus dem Definitionsbereich) ud 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`: konertiert int in ein Unicode zeichen. In Python sind Zeichen einstellige Strings.
- `ord`: konvetiert in die umgehkehrte 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 Funtktionen/1-7_Funkionen.ipynb Cell 5 line 1
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funtktionen/1-7_Funkionen.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 Funtktionen/Funkionen.ipynb Cell 11 line 1
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funtktionen/Funkionen.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 nich direkt zur Verfügung, sondern müssen von einem Modul imporitert werden.

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

Im `math` Modul findet man viele Funktionen wie:

- Pi
- Exponentialfunktionen
- trigonmetrische 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 mathemtatischen Funkiton kann direkt importeert 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 Defintionen 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 vermeinden 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)

Beipsiel:

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 ausgwertet. 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 Paramter** (Variablennamen.)
- beim Funktionsaufruf erhalten die formalen Paramtere 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
```

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

In [ ]:
```
```

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

cat_words ('Hi ', 'Folks')

# Ein Print der Varialbe cat fürht 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 Funtktionen/1-7_Funkionen.ipynb Cell 41 line 8
      <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funtktionen/1-7_Funkionen.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%20Funtktionen/1-7_Funkionen.ipynb#X65sZmlsZQ%3D%3D?line=6'>7</a> # Ein Print der Varialbe cat fürht zu einer Fehlermeldung
----> <a href='vscode-notebook-cell:/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/1_Python_Programmieren/1-7%20Funtktionen/1-7_Funkionen.ipynb#X65sZmlsZQ%3D%3D?line=7'>8</a> print(cat)

NameError: name 'cat' is not defined
```

7.4.3 Scopes

- Rumpf einer Funktion bildet ein Scope (Umgebung)
- der Scope bindet die lokal definierten Variablen und Parameter
- alle weiteren Variablen sind frei im Funktionsrumpf 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.

In [ ]: