**Prozedurale Programmierung**

# Python Einführung

## Teil 1: Experimentieren mit der Python-Konsole

Öffnen Sie die Python-Konsole (Terminal) in Spyder oder VSCode und probieren Sie in dieser die folgenden Beispiele aus. Experimentieren Sie mit einfachen Rechenoperationen und der Ausgabefunktion **print**.

### Spyder

Die Entwicklungsumgebung Spyder (Anaconda3) finden Sie im Startmenü.

### Visual Studio Code

Zur Nutzung der Anaconda-Installation in VS Code öffnen Sie Anaconda-Command-Prompt und starten Sie VS Code durch Eingabe des Befehls **code.**

### 1.1. Variablen, Rechenoperationen und Ausgabe

In Python können neue Variablen beliebig eingeführt werden ohne diese vorher zu deklarieren. Der Typ einer Variable wird festgelegt, sobald ein Wert zugewiesen wird und kann sich auch ändern.
Probieren Sie folgendes Beispiel aus:

```
x = 5
y = 7.5 * x
print( y )
```

```
x = "hallo python"
x = 3.141
z = x * y
```

Welchen Datentyp haben die Variablen x und y? Nutzen Sie die **print**-Funktion sowie die Funktion **type** um sich Werte und Datentypen in der Konsole anzeigen zu lassen.

### 1.2. Ausgabe mehrerer Werte

Die **print**-Funktion kann beliebig viele Werte mit einem Aufruf ausgeben (ähnlich zu **cout** bei C++). Wir können sowohl Text- und Zahlenkonstanten, als auch Variablen auflisten:

```
jahr = 1765
name = "Bergakademie"
print( name , "seit" , jahr )
```

Erweitern Sie die Ausgabe vom 1. Beispiel. Nutzen Sie auch die Sonderzeichen **\t** und **\n** um Ihre Ausgabe wie folgt aussehen zu lassen:

```
x     = 3.141
y     = 37.5
z     = 117,7875
```

### 1.3. Weitere Operatoren und Funktionen

Die Schreibweise `x = x + 5` können Sie auch abkürzen zu `x += 5`. Auch mit Texten (Strings) können wir verschiedene Operationen durchführen, wie Multiplikation, Konkatenation, Länge bestimmen oder alles zu Groß- oder Kleinschreibung konvertieren.

```
x -= y
z /= y
print( "Hurra " * 5 )
text = "TU " + "Bergakademie"
print( len(text) )
print( text.lower(), text.upper() )
```

### 1.4. Import von Modulen

Für die Nutzung von Funktionen, die nicht im Standard-Befehlssatz enthalten sind, müssen wir zusätzliche Module einbinden. Der **import**-Befehl erlaubt verschiedene Schreibweisen. Abhängig von der gewählten Variante sind die importierten Funktionen und Konstanten evtl. in einen Namensraum gekapselt. Das Raute-Symbol # wird in Python für Kommentare genutzt.

Probieren Sie die folgenden Beispiele:

```
import math         # Zugriff auf mathematische Funktionen über "math."
math.e              # Eulersche Zahl e
math.exp(3)         # (e^3)
math.pow(5.1,4)     # (5.1)^4 (pow() ist auch Standardfunktion!)
5.1**4
math.fabs(-23.5)    # Absolutwert einer Zahl (als Gleitkommazahl)
```

```
from math import e      # nur e (Konstante) importiert
e                       # direkter Zugriff
exp(3)                  # Fehler! (denn: nur e importiert)
```

```
from math import *      # Alles importieren, Aber: Namenskonflikte möglich
log(e)
log10(1000)
sin(pi/2)
```

```
import math as m  # Namensraum math umbenennen zu m → guter Kompromiss
m.sqrt(100.0)
m.pi
```

# Teil 2: Kleine Beispielprogramme

Erstellen Sie in Ihrer Entwicklungsumgebung eine neue Code Datei und speichern diese mit unter dem Namen **pykurs1.py** ab.

### 2.1. Selektion mit if / else

Eine Selektion in Python beinhaltet immer einen **if**-Zweig, beliebig viele alternative **elif**-Zweige und maximal einen **else**-Zweig.

```
a = 12
b = 10
if a>b:
    print(a)
elif b>a:
    print(b)
else:
    print(a, "=", b)
```

### 2.2. Eingaben mit input

Die **input**-Funktion erlaubt die Angabe einer Eingabeaufforderung als Parameter und liefert als Ergebnis **immer** einen String! D.h. für die Eingabe von Zahlen müssen wir das Ergebnis noch konvertieren. Erweitern Sie das Programm aus 2.1. um Eingaben für a und b.

```
a = input("Bitte a eingeben:")
a = float(a)

# oder a = float( input("Bitte a eingeben:") )
```

Bauen Sie eine zusätzliche Selektion ein, um nur bei Eingaben größer oder gleich Null den Vergleich durchzuführen.  Achten Sie bei Verschachtelung ganz besonders auf korrekte Einrückung!

```
…
if a >= and b >= 0:
    if a > b:
        …
```

### 2.3. Schleifen

Die Zählschleife ist in Python etwas anders implementiert als in anderen Programmiersprachen - das Prinzip ist aber gleich. In Python durchläuft eine Laufvariable die Elemente einer Sequenz und arbeitet dadurch den Zählschleifenkörper ab. Sequenzen sind z.B. Listen, Tupel oder **range**-Objekte. Das range-Objekt kann man dabei in Python mit der Standardfunktion range() erzeugen.

Erstellen Sie ein neues Python-Dokument und experimentieren Sie mit verschiedenen Möglichkeiten zur Anwendung von **for**-Schleifen:

```
for i in range(10):            # Zaehlschleife von 0 bis 9!
    print(i)
for a in range(1 ,11):
    print(a)
for b in range(2 ,12 ,3):      # von 2 bis 11, Schrittweite 3
    print(b)
for c in "ABCDE":
    print(c)
for i in [6,7,8]:
    print(i)
for i in (6,7,8):
    print(i)
```

Als zweite Schleifenart gibt es in Python die **while**-Schleife.

```
i=1
while i<=10:
    print(i)
    i=i+1

while 5:
    print("Hello")
# Jede Zahl ungleich 0 wird als True interpretiert
# Endlosschleife ! Abbruch mit Strg+C
# oder auch: while True:
```

### 2.4. Funktionen

Zum Abschluss erstellen Sie sich ein drittes Python-Dokument und implementieren eine einfache Funktion, die die Häufigkeit eines Zeichens in einem Text zählt.

```
def count_char(text, character):
    count = 0
    …
    return count
```

Bauen Sie auch Eingaben mit **input** ein und testen Sie den Aufruf Ihrer Funktion. Zum guten Stil gehört es auch für Hauptprogramme bzw. Tests einen **Testrahmen** zu nutzen:

```
if __name__ == "__main__":
    …
    count_car(text, "a")
```