

# Funktionen und Modularisierung

---

Claas de Boer, Tilman Hinnerichs

10. Dezember 2020

Python-Grundlagen

- Datentypen: `int`, `str`, `list`, `set`, `dict`, ...
- Komperatoren: `==`, `!=`, ...
- Schleifen: `while`, `for`
- Ein-/Ausgabe: `input`, `print`

# Heute: Funktionen

---

- Eine **Funktion** ist ein in sich geschlossener Codeblock, der eine bestimmte Aktion ausführt.
- **Funktionen** können mit **Parametern** aufgerufen werden und können **Rückgabewerte** liefern.
- **Funktionen** erlauben es Code zu modularisieren, zu organisieren und wiederverwendbar zu machen.

# Built-In Funktionen

```
1 # Die Anzahl der Elemente eines Containers bestimmen
2 numbers = [1, 2, 3, 4, 5]
3 len(numbers)
4
5 # Den Typ einer Variable herausfinden
6 n_float = 42.8
7 type(n_float)
8
9 # Informationen über ein Objekt bekommen
10 set_of_even_numbers = set(n for n in range(1, 10) if n % 2 == 0)
11 help(set_of_even_numbers)
12
13 # Weitere: sorted(), enumerate(), ...
```

Hier findet ihr eine Liste der Python 3.9.1 Built-in Funktionen.

# Eigene Funktionen I

```
1 # Funktionsdefinition startet mit dem Keyword 'def'
2
3 def simple_function():
4
5     # Hier beginnt der Körper (body) der Funktion
6
7     print("The mitochondria is the powerhouse of the cell.")
8
9     # Hier endet der Körper der Funktion
```

- Kein Parameter, kein Rückgabewert

# Eigene Funktionen II

```
1 # Der Wert eines übergebenen Parameters wird an eine
2 # Variable im Körper der aufgerufenen Funktion gebunden
3
4 def format_greeting(name):
5     return f"Hello {name}. Nice to meet you again!"
6
7 # Der Rückgabewert einer Funktion kann
8 # in einer Variable gespeichert werden
9
10 result = format_greeting("Sun")
11 print(result)
```

- Ein Parameter, ein Rückgabewert

# Eigene Funktionen III

```
1 def calculate_price(item, price, amount=5):  
2     costs = amount * price  
3     return f"{amount} {items}'s will cost you {costs} $."
```

- Funktionen lassen sich auf verschiedene Arten aufrufen.

```
1 >>> calculate_price("Apple", 0.3, 10)  
2 # 10 Apple's will cost you 3.0 $  
3  
4 >>> calculate_price(price=0.3, amount=10, item="Apple")  
5 # 10 Apple's will cost you 3.0 $  
6  
7 >>> calculate_price("Apple", 0.3)  
8 # 5 Apple's will cost you 1.5 $
```



# Parametrisieren will gekonnt sein!

Bei den Parametern einer Funktion unterscheidet man zwischen verschiedenen Typen.

- **Positionelle** Parameter (Erforderliche Parameter)
- **Keyword** Parameter
- **Optionale** Parameter (Parameter mit Default Wert)

# Mini-Exkurs: Documentation

```
1  def complicated_funktion(gamma, offset, inertia):
2      """This function computes the result
3      of calculation foobar for some given
4      parameters.
5
6      A function can get really complicated!
7      But we can describe what a function does in
8      plain text in the function docstring.
9      """
10     ...
11
```

- Code wird häufiger gelesen als geschrieben.
- Dokumentiert euren Code und ihr müsst weniger lesen!
- Gebt euren Funktionen und ihren Parametern **deskriptive** Namen!

# Einige Aufgaben™6

1. Schreibe eine Funktion, welche zwei Zahlen addiert.
2. Schreibe eine Funktion, welche die Anzahl der Vokale einer Zeichenkette berechnet.
3. Schreibe eine Funktion, welche die Länge eines Videos im Format `mm:ss` (bspw. `02:54`) entgegen nimmt und die Länge des Videos in Sekunden zurückgibt.
4. Schreibe eine Funktion, welche die **Fakultät** einer Ganzzahl berechnet. Die Fakultät einer Ganzzahl ist das Produkt der Ganzzahl mit allen kleineren positiven Ganzzahlen.
5. Schreibe eine Funktion, welche die **n-te** Zahl der **Fibonacci-Folge** berechnet.

- Real Python Artikel: Defining your own python function
- Advent of Code