

Einführung in die Programmierung mit Python

Lucerne University of
Applied Sciences and Arts

**HOCHSCHULE
LUZERN**

Syllabus

1. Python installieren; Jupyter Notebooks
2. Grundlagen: Arithmethik, Variablen und Datentypen
3. Methoden; Listen; Bedingte Anweisungen
4. Schleifen
- 5. Benutzerdefinierte Funktionen**
6. Datenaufbereitung und Grafische Darstellungen

Funktionen

- Funktionen sind eine Möglichkeit, Code zu organisieren. Die Grundidee ist, dass Sie ein Stück Code, das wahrscheinlich mehr als einmal verwendet wird, in eine Funktion auslagern, damit es wiederverwendet werden kann.
- Wir haben bereits einige Funktionen kennengelernt, wie z.B. `print` , `sum` , etc.

```
In [1]: sum([1, 2, 3])
```

```
Out[1]: 6
```

- Eine Funktion akzeptiert null oder mehr Eingaben, sogenannte *Argumente*. `sum` oben nimmt (mindestens) eines (probieren Sie aus, was passiert, wenn Sie keine Argumente übergeben).
- Andere Funktionen nehmen beliebig viele Argumente, wie `print`:

In [2]:

```
print()  
print(1)  
print(1, 2)
```

```
1  
1 2
```

- Wenn ein Benutzer die Funktion aufruft, macht sie (meistens) etwas mit ihren Argumenten und muss dann das Ergebnis dieser Operation für den Benutzer oder den umgebenden Code verfügbar machen.
- Es gibt zwei Möglichkeiten, das Ergebnis verfügbar zu machen:
 - "Nebenwirkungen", und
 - Rückgabewerte
- Es ist wichtig, den Unterschied zwischen den beiden zu verstehen.

Betrachten Sie erneut die `print`-Funktion. Sie macht das Ergebnis ihrer Aktion sichtbar, indem sie auf den Bildschirm ausgibt:

In [3]: `print(5)`

5

Andere Nebenwirkungen könnten das Ändern einer Datei, das Abspielen eines Tons, das Herunterfahren des Computers usw. umfassen.

Die `sum`-Funktion ist anders; sie gibt nichts aus:

```
In [4]: a = sum([1, 2, 3])
```

Stattdessen *retourniert* sie ihr Ergebnis. Das bedeutet, dass das Ergebnis einer Variable zugewiesen werden kann, wie oben. Wir können dann natürlich die Variable ausgeben:

```
In [5]: print(a)
```

```
6
```

Die `print`-Funktion hingegen gibt nichts zurück (oder besser gesagt, sie gibt `None` zurück, einen speziellen Datentyp, der Nichts repräsentiert):

```
In [6]: b = print()  
print(b)
```

```
None
```

Hinweis

In Jupyter kann der Unterschied zwischen dem Ausgeben und dem Zurückgeben eines Ergebnisses schwer zu erkennen sein, da Jupyter-Notebooks automatisch den Wert ausgeben, der von dem letzten Ausdruck in einer Zelle zurückgegeben wird, es sei denn, Sie beenden die Zeile mit einem Semikolon, um die Ausgabe zu unterdrücken. Betrachten Sie Folgendes:

```
In [7]: sum([1, 2, 3])
```

```
Out[7]: 6
```

```
In [8]: sum([1, 2, 3]);
```

```
In [9]: print(6)
```

```
6
```

```
In [10]: print(6);
```

```
6
```

Funktionen definieren

- Benutzerdefinierte Funktionen werden mit dem Schlüsselwort `def` deklariert:

```
In [11]: def mypower(x, y): # null oder mehr Argumente, hier zwei  
         return x**y
```

```
In [12]: b = mypower(2, 3)  
        print(b)
```

8

Beachten Sie, wie die vom Benutzer übergebenen Argumente innerhalb der Funktion als die Variablen `x` und `y` verfügbar sind.

Übung

Schreiben Sie eine Funktion `area`, die zwei Zahlen als Eingabe nimmt (die die Seitenlängen eines Rechtecks darstellen) und die Fläche des Rechtecks zurückgibt.

In []:

Mehrere Ausgaben

- Funktionen können mehr als ein Ausgabeargument haben:

```
In [13]: def plusminus(a, b):  
    return a+b, a-b
```

```
In [14]: c, d = plusminus(1, 2);  
print(c, d)
```

```
3 -1
```

Keine Ausgaben

Eine Funktion ohne eine `return`-Anweisung gibt `None` zurück, wie die `print`-Funktion.

```
In [15]: def greet(name):
    print("Hallo", name)
```

```
In [16]: a = greet("Simon")
```

Hallo Simon

```
In [17]: print(a)
```

None

Mit anderen Worten, das Fehlen einer `return`-Anweisung ist gleichbedeutend mit

```
return None
```

Schlüsselwortargumente

- Anstelle von *positionalen Argumenten* können wir auch *Schlüsselwortargumente* übergeben:

```
In [18]: def mypower(x, y):  
    return x**y  
print(mypower(3, 2))  
print(mypower(y=2, x=3) )
```

```
9  
9
```

Defaultargumente

- Funktionen können *Defaultargumente* angeben:

```
In [19]: def mypower(x, y=2): # Defaultargumente müssen am Ende stehen
          return x**y
mypower(3)
```

```
Out[19]: 9
```

```
In [20]: mypower(3, 3)
```

```
Out[20]: 27
```

Übung, Fortsetzung

Schreiben Sie eine Funktion `area`, die zwei Zahlen als Eingabe nimmt (die die Seitenlängen eines Rechtecks darstellen) und die Fläche des Rechtecks zurückgibt. Wenn die zweite Eingabe nicht angegeben wird, soll die Funktion die Fläche eines Quadrats mit der Seitenlänge der ersten Eingabe berechnen.

Hinweis: Lassen Sie die zweite Eingabe standardmäßig auf `None` gesetzt.

Erwartete Ausgabe:

```
area(2, 3) # sollte 6 zurückgeben  
area(2)    # sollte 4 zurückgeben
```

In []:

Zusammenfassung / weiterführende Literatur (optional)

- Optionales Notebook "Mehr über Funktionen"
- https://www.w3schools.com/python/python_functions.asp
- <https://python-course.eu/python-tutorial/functions.php>

Hausaufgabe

- Anfängerübung 16 von <https://holypython.com/beginner-python-exercises/>
- Übungen 2, 3, 6, 9 (schwierig), 10, 16 von <https://www.w3resource.com/python-exercises/python-functions-exercises.php>