

Functions

Foliensatz 6

Jasmin Noll
wi20086@lehre.dhbw-stuttgart.de

05.05.2022

Was euch heute erwartet

- Funktionen
 - Argumente
 - *args
 - Keyword-Argumente
 - *kwargs
 - Returns
 - Defaultwerte
 - Typestring
 - Docstrings
 - Import
- Lambda

Funktionen

05.05.2022

- Codeblock wird nur ausgeführt, wenn dieser mit dem Funktionsnamen aufgerufen wird
 - Funktionsname sollte dabei aussagekräftig sein
- Argumente können übergeben werden, die frei im Codeblock verwendet werden können
- Codeblock kann Ergebnisse mit `return` wiedergeben
- Funktionen machen Sinn, wenn z.B. bestimmter Code immer wiederholt werden muss
 - Damit kann man Zeilen sparen und auch Code leserlicher gestalten
- Wird erstellt mit `def` und in `()` werden alle Argumente übergeben und mit `:` beendet
 - Beispiel:

```
def addition(argument_1, argument_2):  
    return argument_1 + argument_2
```
- Beim Aufrufen müssen alle Argumente übergeben werden
 - Beispiel:

```
result = function_name(3, 2)
```

Argumente aka Parameter

- In einer Funktion muss können keine bs beliebig viele Parameter übergeben werden
- Parameter können von jedem Datentyp sein
- Parameter wird benannt und mit diesem Namen wird der übergebene Wert im Codeblock verarbeitet.
 - Name existiert nicht global sondern nur im Rahmen der Funktion
- Es muss die selbe Anzahl an Argumenten in der entsprechenden Reihenfolge beim Aufruf übergeben werden wie sie beim erstellen der Funktion definiert worden sind
 - Ausnahmen:
 - `*args`
 - `**kwargs`

args und kwargs sind Platzhalter

*args

- Verwenden, wenn man nicht weiß, wie viele Argumente übergeben werden
 - Funktion erwartet ein Tupel mit den einzelnen Werten die übergeben werden sollten
 - Beispiel:

- Funktion erstellen

```
def addition(*numbers):  
    return numbers[0] + numbers[1]
```

- Funktion Aufrufen

```
result = addition(1, 2)
```

```
def addition(*numbers):  
    result = 0  
    for number in numbers:  
        result += number  
    return result
```

Keyword Argumente

- Beim Aufrufen können die Argumente als *key* mit dem entsprechenden Wert übergeben werden

- Dadurch kann auch die Reihenfolge der Übergabe verändert werden

- Beispiel:

```
def addition(number_1, number_2):  
    result = number_1 + number_2  
    return result
```

```
result_1 = addition(number_1 = 3, number_2 = 2)
```

```
result_2 = addition(number_2 = 2, number_1 = 3)
```

****kwargs**

- Verwenden, wenn man nicht weiß wie viele Argumente übergeben werden
 - Werte können wie ein Dictionary im Codeblock verwendet werden
 - Die übergebenen Werte müssen trotzdem den im Codeblock verwendeten Namen entsprechen
 - Beispiel:

- Funktion erstellen

```
def addition(**numbers):  
    result = numbers["number_1"] + numbers["number_2"]  
    return result
```

- Funktion aufrufen

```
result = addition(number_1 = 3, number_2 = 2)
```

Man kann auch mehr Parameter übergeben, als tatsächlich in der Funktion verwendet werden

```
result = addition(number_1 = 3, number_2 = 2, number_3 = 3)
```


Returns

- Ergebnisse die in der Funktion berechnet werden können über **return** am Ende der Funktion aufrufen

- Eine Funktion kann kein und auch mehrere *return-statements* im Codeblock haben
 - Funktion “bricht ab”, sobald das erste *return-statement* erreicht wird

- Beispiel:

```
def calc(number_1, number_2):  
    if number_1%2 == 0:  
        return number_1 * number_2  
    return number_1 + number_2
```

- Es können auch mehrere Ergebnisse mit einem return-statement ausgegeben werden

- Beispiel:

- Funktion erstellen

```
def calc(number_1, number_2):  
    result_1 = number_1 + number_2  
    result_2 = number_1 - number_2  
    return result_1, result_2
```

- Funktion aufrufen

```
result_1, result_2 = calc(5, 2)
```

Defaultwerte

- Defaultwerte können beim erstellen der Funktion mitgegeben werden
- Dieser Wert wird verwendet, wenn beim aufrufen der Funktion für diesen Argumente kein Wert übergeben werden
 - Beispiel:
 - Funktion erstellen

```
def addition(number_1 = 3, number_2 = 2):  
    return number_1 + number_2
```
 - Funktion aufrufen

```
result = addition(4)  
result = addition(number_2 = 5)
```

Typestrings

- Geben der Funktion an welcher Datentypen für die einzelnen Argumente zu erwarten sind
 - Wird für jedes Argument einzeln angegeben
 - Hilft beim Verständnis, wenn andere den Code verwenden
 - Reine Dokumentation
 - Beispiel:
 - Funktion erstellen

```
def addition(number_1:int, number_2:float):  
    return number_1 + number_2
```
 - Funktion aufrufen

```
result = addition(3, 2.0)
```

Docstrings

- Docstrings sind zur Dokumentation der Funktion da
- Beinhalten eine kurze Erklärung wofür die Funktion da ist, welche Argumente übergeben werden und was als Ergebnis wieder zurück gegeben wird
- Wird direkt in die ersten Zeile der Funktion geschrieben.

- Es werden """ """ dafür verwendet

- Beispiel:

```
def addition(number_1, number_2):  
    """Diese Funktion addiert zwei Werte miteinander  
    Params:  
        number_1 (int): Erste Nummer  
        number_2 (int): Zweite Nummer  
    Returns:  
        Ergebnis aus der Addition der zwei Nummern  
    """  
    return number_1 + number_2
```

**Es gibt für VSCode
Extensions die es einem
einfacher machen einen
korrekten Docsting zu
schreiben
Beispiel: autoDocsting**

Import

- Erstellte Funktionen können in einer (separaten) Datei gespeichert werden und in anderen Dateien über import verwendet werden
 - Beispiele:
 - `import function_file`
 - `from function_file import function_1`

```
def addition(number_1:int = 3, number_2:int = 4):  
    """Diese Funktion addiert zwei Werte miteinander  
  
    Params:  
        number_1 (int): Erste Nummer  
        number_2 (int): Zweite Nummer  
  
    Returns:  
        Ergebnis aus der Addition der zwei Nummern  
    """  
  
    return number_1 + number_2
```

```
result = addition(5)
```

Lambda

05.05.2022

15

- Eine anonyme Funktion
- Kann beliebig viele Argumente nehmen, aber nur einen Ausdruck
- Eingeleitet wird die Lambdafunktion mit `lambda`, danach kommen alle benötigten Argumente. Diese werden mit `:` von dem Ausdruck getrennt
 - Beispiel:
 - Lambdafunktion erstellen
`func = lambda number_1, number_2 : number_1 + number_2`
 - Lambdafunktion aufrufen
`result = func(3, 4)`
- Lambdafunktionen können auch in regulären Funktionen als anonyme Funktion aufgerufen werden

Übungen

