Programmierung in Python

Univ.-Prof. Dr. Martin Hepp, Universität der Bundeswehr München

Einheit 3: Objektorientierte Programmierung in Python

Version: 2019-12-05

http://www.ebusiness-unibw.org/wiki/Teaching/PIP

1 Funktionen und Modularisierung

1.1 Motivation

- Programmteile, die oft wiederholt werden, sollten nicht mehrfach vorhanden sein.
- Stattdessen sollte man sie über einen Namen aufrufen können.
- Einen Programmteil, den man über seinen Namen aufrufen kann, nennt man in der Programmierung **Funktion** oder **Methode**.
- Programme werden dadurch kürzer und übersichtlicher.

Beispiel: Im Folgenden wird zwei Mal 'Hallo UniBwM' ausgegeben:

In [60]:
```
print('Hallo UniBwM')
print('Hallo UniBwM')
```

```
Hallo UniBwM
Hallo UniBwM
```

Besser wäre es, wenn man diese Funktion ein Mal definieren und dann bei Bedarf aufrufen könnte.

1.2 Definition von Funktionen

Funktionen werden in Python mit dem Schlüsselwort `def` definiert.

In [61]:
```
def say_hello():
    print('Hallo UniBwM')
```

Anschließend kann man Sie jederzeit über ihren Namen aufrufen:

In [62]:
```
say_hello()
say_hello()
```

```
Hallo UniBwM
Hallo UniBwM
```

Hier spart man zwar nicht wirklich viel an Programmlänge. Aber wenn man den Begrüßungstext ändern möchte, muss man das nur an einer Stelle tun.

1.3 Übergabeparameter

Eine Funktion kann so definiert werden, dass man ihr Werte als Parameter übergibt, die dann das Verhalten der Funktion verändern.

Dazu definiert man in runden Klammern eine Liste von Namen, über die der jeweils übergebene Wert **innerhalb der Funktion** verfügbar ist.

Beispiel:

In [66]:
```
def say_text(text):
    print(text)
```

In [67]:
```
say_text('Hallo UniBwM')
```

```
Hallo UniBwM
```

Natürlich können auch mehrere Parameter übergeben werden:

In [65]:
```
def multipliziere(wert_1, wert_2):
    """Multipliziert <wert_1> mit <
    ergebnis = wert_1 * wert_2
    print(ergebnis)
```

In [64]:
```
# Aufruf mit Werten
multipliziere(3, 5)
```

```
15
```

In [63]:
```
# Aufruf mit Variablen
a = 3
b = 5
multipliziere(a, b)
```

```
15
```

1.3.1 Arten von Parametern

Beim Aufruf müssen die übergebenen Parameter zu den Parametern in der Definition passen. Die Parameter einer Funktion oder Methode nennt man auch Signatur.

Hier gibt es mehrere Möglichkeiten.

1.3.1.1 Positional Arguments

Parameter können einfach über ihre Reihenfolge bestimmt sein:

In [3]:
```
def funktion(parameter_1, parameter_
    print(parameter_1, parameter_2)

funktion(1, 5)
```

```
1 5
```

In [70]:
```
# Die Reihenfolge bestimmt die Zuor
teile(3, 5)
```

```
0.6
```

In [69]:
```
def teile(dividend, divisor):
    print(dividend / divisor)

teile(15, 3)
```

```
5.0
```

1.3.1.2 Named Arguments

Alternativ kann man auch einen Namen vorgeben, der verwendet werden muss. Dies erlaubt auch Default-Werte.

In [71]:
```
def funktion(parameter_name='UniBwM'):
    print(parameter_name)
```

In [72]:
```
# Aufruf mit Parameter
funktion(parameter_name='hallo')
```

```
hallo
```

In [43]:
```
# Variable text wird als Wert überg
text = 'TUM'
funktion(parameter_name=text)
```

```
TUM
```

In [74]:
```
# Wenn der Parameter fehlt, wird der Default-Wert verwendet:
funktion()
```

```
UniBwM
```

1.4 Rückgabewerte

Eine Funktion kann einen Wert als Ergebnis zurückliefern. Dazu dient das Schlüsselwort `return`.

Dann muss man den Aufruf der Funktion einer Variable zuweisen, ausgeben oder auf andere Weise in einem Ausdruck verwenden:

In [77]:
```
def verdopple(parameter_1):
    ergebnis = parameter_1 * 2
    return ergebnis

print(verdopple(10))
mein_wert = verdopple(10)
print(mein_wert)
```

```
20
20
```

Der Aufruf einer Funktion mit Rückgabewert kann wie jeder andere Wert oder Ausdruck verwendet werden.

```
In [76]: print(verdopple(verdopple(4)))

         16
```

```
In [75]: print(verdopple(10) * 1.19)

         23.799999999999997
```

Wenn man mehrere Werte zurückliefern möchte, muss man einen Datentyp verwenden, der Unterelemente enthält. Beispiele:

- Tupel
- Dictionary
- Benutzerdefinierte Objekte

In [9]:
```
def zwei_werte(parameter_1):
    wert_1 = parameter_1 * 2
    wert_2 = parameter_1 * 3
    return (wert_1, wert_2)

ergebnis = zwei_werte(4)
print(ergebnis)
a, b = ergebnis
print(a, b)
```

```
(8, 12)
```

Übungsaufgabe: Schreiben Sie eine Funktion, die die Quersumme einer als Parameter übergebenen Ganzzahl zurückliefert.

In [56]:
```
def quersumme(zahl):
    zahl = str(zahl)
    ergebnis = 0
    for ziffer in zahl:
        ergebnis = ergebnis + int(ziffer)
    return ergebnis

print(quersumme(426))
```

```
12
```

2 Bibliotheken und Import von Modulen

2.1 Import von Modulen mit `import`

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

2.2 Standard-Bibliothek von Python

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

2.3 Vorinstallierte Bibliotheken in Anaconda

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

2.4 Installation zusätzlicher Bibliotheken

Dieses Thema wird im Rahmen der Vorlesung nicht behandelt und hier nur der Vollständigkeit halber erwähnt.

3 Objektorientierung als Programmierparadigma

3.1 Motivation

3.2 Klassen

3.3 Instanzen

3.4 Methoden

3.5 Beispiel

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

4 Definition einer Klasse

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

4.1 Grundgerüst

- Klasse
- Parameter
- DOCSTRING
- Konstruktor-Methode
- Klassenmethoden
- Klassenvariablen
- Instanzenvariablen

4.2 Variablen auf Instanzebene

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

4.3 Konstruktor-Methode

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

4.4 Methoden auf Klassenebene

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de

5 Quellenangaben und weiterführende Literatur

[Pyt2019] Python Software Foundation. Python 3.8.0 Documentation. https://docs.python.org/3/.

Vielen Dank!

http://www.ebusiness-unibw.org/wiki/Teaching/PIP

Univ.-Prof. Dr. Martin Hepp, martin.hepp@unibw.de