Module und Erweiterte Iteration

Richard Müller, Tom Felber

2. Dezember 2021

Python-Kurs

Gliederung

1. Wiederholung
2. Module
3. Keyword-Arguments
4. Itertools - Hilfe bei der Iteration

Gesamtübersicht

Gesamtübersicht

Themen der nächsten Stunden

- Referenzen Erklärung
- Klassen
- Imports
- Nützliche Funktionen zur Iteration
- Lambda
- File handeling
- Listcomprehension
- Unpacking
- Dekoratoren

Wiederholung

Wiederholung

Beim letzten Mal:

- Vererbung

```
class NadelBaum(Baum):
    pass
```

- Module Einführung

```
import mein_modul
i = mein_modul.ImportiereMich()
```

Module

Grundlegendes

Erinnerung an Importe:

```
1 import my_modul
2 i = my_modul.MeineKlasse()
```

oder:

```
1 from my_modul import MeineKlasse
2 i = MeineKlasse()
```

Importe funktionieren aus dem selben und aus niedrigeren Verzeichnissen ohne Probleme. Will man etwas aus einem höheren Verzeichniss importieren, so wird deutlich mehr Konfigurationsaufwand benötigt. Moderne IDEs nehmen das allerdings ab.

__name__ == '__main__'

Importiert man ein Modul, so wird jeder Code sofort ausgeführt, der nicht in Funktionen oder Klassen verpackt ist.

Um das zu verhindern, gibt es ein 'Protection Statement':

```
1 if __name__ == "__main__":
2     print("Wird nur ausgeführt, wenn dieses Script direkt
      aufgerufen wurde")
```

Die `__name__`-Variable nimmt nur den Wert `'__main__'` an, wenn das Modul direkt ausgeführt wird.

Wird es nur importiert, so hat die Variable einen anderen Wert. Dadurch wird das durch die `if`-Abfrage geschützte Codesegment nicht mit ausgeführt.

Standardbibliothek

Die Standardbibliothek ist eine Sammlung von Modulen, die bereits nach der Installation von Python vorhanden ist.

Zwei Beispiele

time:

```
1 import time
2 # hält das Programm für 5s an
3 time.sleep(5)
```

random:

```
1 import random
2 # Zufallszahl ziwschen 0 und 9
3 r = random.randint(0, 9)
```

In der Doku findet man alle verfügbaren Module.

Keyword-Arguments

Keyword-Arguments

Wenn man die Reihenfolge von Argumenten in einer Funktion übergehen will, kann man den Namen des Parameters direkt benutzen. Diese Argemente werden dann als **Keyword-Arguments** bezeichnet.

```
1 def aktion(argument_1, argument_2="Y", argument_3="Z"):
2     print(f"{argument_1} {argument_2} {argument_3}")
```

Ziel: X Y Z

```
1 aktion("X")
```

Ziel: X Y Hallo

```
1 aktion("X", "Y", "Hallo")
```

besser mit Keyword-Arguments

```
1 aktion("X", argument_3="Hallo")
```

Keyword-Arguments

Es kann auch erzwungen werden, dass Argumente Keyword-Arguments sind. Alle Argumente nach `, *,` sind zwangsweise Keyword-Arguments.

```
1 def aktion(argument_1, *, argument_2, argument_3):
2     print(f"{argument_1} {argument_2} {argument_3}")
```

Ziel: `X Y Z`

```
1 aktion("X", argument_2="Y", argument_3="Z")
```

Itertools - Hilfe bei der Iteration

Itertools

Itertools ist eine Bibliothek, um die Iteration zu erleichtern. Unter anderem kann sie effizient große Iterationen durchführen.

```
1 import itertools
```

Beispiel:

```
1 numbers = [1, 2, 3, 4, 5, 6, 7]
```

```
1 for i in numbers:
2     for j in numbers:
3         t = (i, j)
4         print(t)
```

Mit Itertools:

```
1 for t in itertools.product(numbers, repeat=2):
2     print(t)
```

Itertools

```
numbers = [1, 2, 3, 4, 5, 6, 7]
```

```
for i in numbers:
    for j in numbers:
        for k in numbers:
            t = (i, j, k)
            print(t)
```

Mit Itertools:

```
for t in itertools.product(numbers, repeat=3):
    print(t)
```