PYTHON

Funktionen + Generator/Coroutine

Mohammad Sameer Raha

Proseminar Python

14.06.2022

INHALTSVERZEICHNIS

WAS SIND FUNKTIONEN?

- Auslagerung von einzelnen Funktionalitäten eines Programms in unterschiedliche Funktionen
- Mehrmals aufrufbar → übersichtlicherer Code
- Funktion → Unterprogramm

- „print“

➢ Built-in Function → in Python schon verfügbare Funktion

```
print("Bester Vortrag")
```

WIE DEFINIERT MAN FUNKTIONEN?

```
1 def gruss():
2     print("Hallo Python-Kurs")
3     print("Willkommen zu meinem Vortrag")
```

```
1 def gruss():
2     print("Hallo Python-Kurs")
3     print("Willkommen zu meinem Vortrag")
4
5 gruss()
```

Ausgabe:

Hallo Python-Kurs
Willkommen zu meinem Vortrag

```
1  def gruss():
2      print("Hallo Python-Kurs")
3      print("Willkommen zu meinem Vortrag")
4
5  def gruss2():
6      print("Hallo Wlad")
7      print("Willkommen zu meinem Vortrag")
8
9  def gruss3():
10     print("Hallo Elias")
11     print("Willkommen zu meinem Vortrag")
```

FUNKTIONEN MIT PARAMETER

```
1 def gruss(name):
2     print("Hallo " + name)
3     print("Willkommen zu meinem Vortrag")
4
5 gruss("Wlad")
```

Ausgabe:

Hallo Wlad
Willkommen zu meinem Vortrag

MEHRERE PARAMETER

```
1 def count(start, steps):
2     i = 0
3     while start > i:
4         i += steps
5         print(i)
6
7 count(5, 1)
```

Ausgabe:
1
2
3
4
5

- Parameter mit Komma trennen
- Parameter heißen Argumente, wenn ihnen ein Wert zugeteilt wird

OPTIONALE & OBLIGATORISCHE PARAMETER

```
1 def count(start, steps=1):
2     i = 0
3     while start > i:
4         i += steps
5         print(i)
6
7 count(5)
```

Ausgabe:
1
2
3
4
5

- Optionale Parameter werden beim Funktionsaufruf nicht angegeben
- Python benutzt Standartwert für nicht übergebene Parameter
- Parameter ohne Standartwert sind obligatorisch

FUNKTION MIT RÜCKGABEWERT

```
1 def minimum(a, b):
2     if a < b:
3         return a
4     else:
5         return b
6
7 print(minimum(3, 7))
```

Ausgabe: 3

- Schlüsselwort „return“ notwendig
- Ausführung des return-Befehls führt zum Ende der Funktion

*ARGS

- * → symbolisiert eine undefinierte Anzahl an Argumenten

```
1 def add_more(*args):
2     result = 0
3
4     for i in args:
5         result += i
6
7     return result
8
9 print(add_more(2, 3, 2, 7))
```

Ausgabe: 14

KEYWORD ARGUMENTS

```
1 def favorite_colour(favorite = "blau"):
2     print("Meine Lieblingsfarbe ist " + favorite)
3
4 favorite_colour()
5 favorite_colour(favorite = "rot")
```

Ausgabe:

Meine Lieblingsfarbe ist blau
Meine Lieblingsfarbe ist rot

**KWARGS

- Undefinierte Anzahl an keyword arguments → **kwargs

```
1 def favorite_food(**kwargs):
2     print("Mein Lieblingsessen ist " + kwargs["favorite"])
3
4 favorite_food(favorite = "Reis", second_favorite = "Pizza")
```

Ausgabe:

Mein Lieblingsessen ist Reis

GENERATOREN

- Funktion, wobei das Iterieren über Objekte ermöglicht wird

- yield statt return statements

➢ return beendet eine Funktion

➢ yield hält Funktion an, speichert alle Zustände und fährt von dort aus bei weiteren Aufrufen fort

GENERATOREN

```
1 def numbers(nums):
2     result = []
3     for i in nums:
4         result.append(i*i)
5     return result
6
7 x = numbers([1,2,3,4,5])
8
9 print(x)
```

Ausgabe:

[1, 4, 9, 16, 25]

```
1 def numbers(nums):
2     for i in nums:
3         yield (i*i)
4
5 x = numbers([1,2,3,4,5])
6
7 print(x)
```

Ausgabe:

<generator object numbers at 0x000002A113D5E270>

```
1  def numbers(nums):
2      for i in nums:
3          yield (i*i)
4
5  x = numbers([1,2,3,4,5])
6
7  print(next(x))
8  print(next(x))
9  print(next(x))
10 print(next(x))
11 print(next(x))
```

Ausgabe:

1
4
9
16
25

```
1  def numbers(nums):
2      for i in nums:
3          yield (i*i)
4
5  x = numbers([1,2,3,4,5])
6
7  print(next(x))
8  print(next(x))
9  print(next(x))
10 print(next(x))
11 print(next(x))
12 print(next(x))
```

Ausgabe:

1
4
9
16
25
Traceback (most recent call last):
  File
"C:\Users\User\IdeaProjects\FirstTimePython\
Vortrag.py", line 60, in <module>
    print(next(x))
StopIteration

```
1 def numbers(nums):
2     for i in nums:
3         yield (i*i)
4
5 x = numbers([1,2,3,4,5])
6
7 for num in x:
8     print(num)
```

Ausgabe:

1
4
9
16
25

COROUTINE

- Verallgemeinerung von Unterprogrammen

- Verwendung für kollaboratives Multitasking
  - ➢ Prozess gibt aus eigenem Willen oder im Leerlauf die Kontrolle ab
    - ➢ gleichzeitiges Ausführen mehrerer Anwendungen möglich

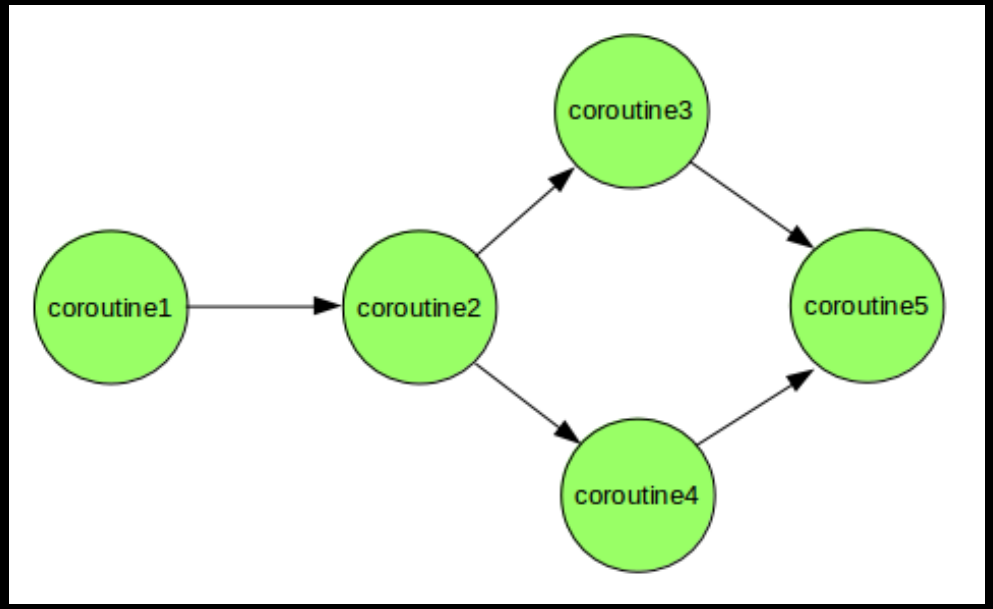
- Unterschiede zu Unterprogrammen:

➢ mehrere Einstiegspunkte

➢ Ausführung kann unterbrochen werden + Steuerung auf andere Coroutine übertragbar + Ausführung von dort aus fortsetzbar

➢ keine Hauptfunktion vorhanden

➢ Coroutinen können eine Pipeline bilden



WIE DEFINIERT UND PAUSIERT MAN EINE COROUTINE?

```
async def main():
    await awaitable_object
```

```
1 async def countries():
2     print("Germany")
3     await asyncio.sleep(2)
4     print("France")
```

WIE VERLÄSST MAN EINE COROUTINE?

```
loop = asyncio.get_event_loop()
loop.run_until_complete(countries())
loop.close()
```

Shortcut:

```
asyncio.run(countries())
```

```python
import asyncio

async def cor1():
    print("cor1 start")
    await asyncio.sleep(2)
    print("cor1")

async def cor2():
    print("cor2 start")
    await asyncio.sleep(1)
    print("cor2")


loop = asyncio.get_event_loop()
cors = asyncio.wait([cor1(), cor2()])
loop.run_until_complete(cors)
loop.close()
```

Ausgabe:

cor1 start
cor2 start
cor2
cor1

QUELLENVERZEICHNIS

- Zu Funktionen:
  - https://www.youtube.com/watch?v=mgA-Ytr32Ys&list=WL&index=35
  - https://studyflix.de/informatik/python-funktion-3221
  - https://www.youtube.com/watch?v=af9ORp1Pty0&list=WL&index=36&t=1s
  - https://www.youtube.com/watch?v=dbT7PpRz2nc&list=WL&index=39

- Zu Generatoren:
  - https://www.python-kurs.eu/generatoren.php
  - https://www.youtube.com/watch?v=bD05uGo_sVI&list=WL&index=40&t=407s
  - https://lernenpython.com/generator/

- Zu Coroutine:
  - https://de.acervolima.com/coroutine-in-python/
  - https://www.youtube.com/watch?v=c6uoxhaenHg&ab_channel=IndianPythonista
  - https://riptutorial.com/Download/python-language-de.pdf