TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Alexander Jenke, Theodor Straube

# Übung8: Environment, Iterators, Generators

Programmierkurs Python // Mittwoch, 18. Dezember 2019

# ToC

- Virtual Environment
- Shebang
- Iterators
- Generators

# Venv - Virtual Environment

- Installierte Packages in eine eigene Umgebung kapseln
  - ➢ Verschiedene Versionen für verschiedene Projekte möglich

- Erstellen mittel dem venv-Package
- Aktivieren über *source* PATH_TO_ACTIVATE
- Deaktivieren über *deactivate*

```
alexanderjenke@MacBook-Pro ~ python3 -m venv PATH_TO_VENV
```

```
alexanderjenke@MacBook-Pro ~ source test-env/bin/activate
(test-env) alexanderjenke@MacBook-Pro ~
```

- Innerhalb des Venv können mittels Pip Packages installiert und verwaltet werden
- PyCharm bietet beim Wählen des Interpreters das Erstellen eines Venv an

# Shebang

- In einer Python-Datei kann definiert werden mit welchem Interpreter diese ausgeführt werden soll
- Danach kann die Datei ausführbar gemacht werden (*chmod u+x PATH_TO_FILE*)
- Interpreter kann auch in einem Venv liegen
  - ➢ dann wird die Datei innerhalb des Venv ausgeführt

- **#!/usr/bin/env python3** für den Standard-Interpreter
- **#!test-env/bin/python3** für unseren Interpreter im test-env Venv

# Iterators

- Ermöglichen das Iterieren über mehrere Objekte
  - Z.b. nacheinander alle Elemente einer Liste in einer for-Schleife zurückbekommen

- *__iter__:* Setup für Iteration, gibt Iterator zurück
- *__next__:* gibt nächstes Element
- *raise StopIteration*: signalisiert Ende der Liste, stoppt Iteration
- *iter(*Iteration*):* Ruft __iter__ auf

```
class Reverse:
    def __init__(self, l: list):
        self.list = l

    def __iter__(self):
        self.i = len(self.list)
        return self

    def __next__(self):
        self.i -= 1
        if self.i < 0:
            raise StopIteration
        return self.list[self.i]


l = [0, 1, 2, 3, 4, 5]
it = Reverse(l)
for i in it:
    print(i)
```

# Generators

- Ermöglichen das Iterieren über eine Menge genau wie ein Iterator
- Aber die Menge muss vorher nicht geladen sein, sondern die Elemente können bei Anforderung generiert werden

- Funktion mit *yield* statt *return*
- Nächstes Element mit *next()*

```
def fibonacci(n):
    a, b = 0, 1
    for i in range(n):
        if i < 2:
            yield i
        a, b = b, a + b
        yield b


fib = fibonacci(1000)
print(fib)
for i in range(11):
    print(next(fib))
```

```
class Fibonacci:
    def __init__(self):
        self.a = 0
        self.b = 1
        self.i = 0

    def __iter__(self):
        return self

    def __next__(self):
        self.i += 1
        if self.i < 3:
            return self.i - 1

        self.a, self.b = self.b, self.a + self.b

        return self.b


fib = Fibonacci()
c = 0
for i in fib:
    print(i)
    c += 1
    if c > 10: break
```

# nächstes Mal

- Mehr Zeit für Aufgaben
- Zeit für Fragen
- Lösungen

- **Nächster Termin: 08. Jan 2020**