Decorators

Felix Döring, Felix Wittwer

1. Dezember 2021

Python-Kurs

Gliederung

1. Funktionen höherer Ordnung
    - Beispiele

2. Decorator
    - einfache Decorator
    - Decorator mit Argumenten

Funktionen höherer Ordnung

Funktionen sind Objekte

Funktionen sind Objekte mit speziellen Methoden.

```
# def greet(name):
#     return "Hello {0}".format(name)

class GreetFunction:
    # wird beim Funktionsaufruf aufgerufen
    def __call__(self, name):
        return "Hello {0}".format(name)

greet = GreetFunction()

# Hello World
print(greet("World"))
```

Folgen

- Funktionen können Variablen zugewiesen werden,

```
# Funktionen Variablen zuweisen
def return_hello():
    return "Hello"

say_hello = return_hello

print(say_hello())
# => Hello
```

- Sie können in Funktionen definiert werden,

```
# Funktionen in Funktionen definieren
def greet(name):
    def return_hello():
        return "Hello "
    greeting = return_hello() + name
    return greeting

print(greet("World"))
# => Hello World
```

Fakten über Funktionen

- Sie können andere Funktionen zurückgeben,

```
# Rueckgabe von Funktionen
def greet():
    def say_hello():
        return "Hello"
    return say_hello

print(greet()())
# Hello
```

Fakten über Funktionen

- Sie können als Parameter mitgegeben werden

```
1  # Uebergabe von Funktionen
2  def say_date(date):
3      return "Today it's {date}".format(date=date)
4
5  def which_date(function):
6      date = "25th June 2015"
7      return function(date)
8
9  print(which_date(say_date))
10 # => Today it's 25th June 2015
```

Beispiele - map

`map(function, iterable)` wendet eine Funktion auf alle Elemente eines Iterators an.

```
# gibt Fizz wenn die Zahl durch 3 und Buzz
# wenn die Zahl durch 5 teilbar ist zurück
def fizzbuzz(number):
    teilbar_3 = number % 3 == 0
    teilbar_5 = number % 5 == 0
    if teilbar_3 and teilbar_5:
        return "FizzBuzz"
    elif teilbar_3:
        return "Fizz"
    elif teilbar_5:
        return "Buzz"
    else:
        return number

# gibt FizzBuzz, 1, 2, Fizz, 4, Buzz, Fizz, 7, 8 und Fizz aus
for number in map(fizzbuzz, range(0, 10)):
    print(number)
```

Beispiele - filter

`filter(function, iterable)` gibt die Elemente eines Iterators zurück, für welche die Funktion `True` zurückgibt.

```
1 # gibt zurück ob eine Zahl gerade ist
2 def even(number):
3     return number % 2 == 0
4
5 # gibt die Zahlen 0, 2, 4, 6 und 8 aus
6 for number in filter(even, range(0, 10)):
7     print(number)
```

Decorator

einfache Decorator

Decorator sind Wrapper über existierende Funktionen. Dabei werden die zuvor genannten Eigenschaften verwendet.
Eine Funktion, die eine weitere als Argument hat, erstellt eine neue Funktion.

```
1 def get_date(date):
2     return ", today it's {}.".format(date)
3
4
5 # unser decorator
6 def tell_the_world(func):
7     def complete_sentence(date):
8         return "Hello World{}".format(func(date))
9     return complete_sentence
```

einfache Decorator

```
# normaler Aufruf:
print(tell_the_world(get_date)("25th June 2015"))

# als decorator
get_date = tell_the_world(get_date)

print(get_date("25th June 2015"))
# => Hello World, today it's 25th June 2015.
```

einfache Decorator

Durch das @ Symbol lässt sich der Decorator wesentlich einfacher verwenden.

```
# Nutzung des @ Syntaxes
def tell_the_world(func):
    def complete_sentence(date):
        return "Hello World, {}".format(func(date))
    return complete_sentence


@tell_the_world
def get_date(date):
    return "today it's {}.".format(date)

print(get_date("25th June 2015"))
# => Hello World, today it's 25th June 2015.
```

Es können auch mehrere Decorator übereinander geschrieben werden.

Decorator mit Argumenten

Decorator erwarten Funktionen als Argumente. Aus diesem Grund kann man nicht einfach andere Argumente mitgeben, sondern man muss eine Funktion schreiben, die dann den Decorator erstellt.

```
1  def tell_the_date_to(name):
2      def name_decorator(func):
3          def complete_sentence(date):
4              return "Hello {name}, {date}".format(name=name, date
       =func(date))
5          return complete_sentence
6      return name_decorator
7
8
9  @tell_the_date_to("John Doe")
10 def get_date(date):
11     return "today it's {}.".format(date)
12
13 print(get_date("25th June 2015"))
14 # => Hello John Doe, today it's 25th June 2015.
```