Decorators

Dominik Pataky

13. November 2017

Python-Kurs 2017

Gliederung

1. Fakten über Funktionen

2. Decorator
   - einfache Decorator
   - Decorator mit Argumenten

Fakten über Funktionen

Fakten über Funktionen

- 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
```

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.

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


# unser decorator
def tell_the_world(func):
    def complete_sentence(date):
        return "Hello World{}".format(func(date))
    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.
```