Higher Order Funktionen

Funktionen höherer Ordnung

Was genau sind Funktionen höherer Ordnung?

Funktionen, deren Parameter selbst wieder Funktionen sind oder die Funktionen als Rückgabewert zurückgeben, heißen Higher Order-Functions – Funktionen höherer Ordnung.

Wozu braucht man das?

z.b.

Mengen sortieren

Mengen filtern

Mengen reduzieren

1. Funktionale Programmierung

Von seinen funktionalen Vorbildern (z. B. der Programmiersprache Lisp) hat Python die Besonderheit geerbt, dass Funktionen einfach Werte sind.

Genauso, wie wir eine Zahl einer Variable zuordnen können, können wir das auch mit Funktionen tun.

```
price = 3
add = lambda x,y: x + y
```

2. First class citizens

Die funktionale Programmierung spricht von sogenannten "first class citizens". Damit ist gemeint, dass die Sprache Funktionen genauso behandelt wie andere Werte.

Sie kann mit Variablen referenziert werden und sogar als Argumente an Funktionen übergeben werden.

Gerade letzteres lässt sich oft gut gebrauchen.

1. Higher Order Funktion map

Die map() (engl. abbilden) Funktion wendet auf jedes Element eines Iterables die bereitgestellte Funktion an und gibt das Ergebnis in einem neuen Map-Objekt zurück. Dieses Map-Objekt selbst ist ein Iterator!

```
map(fn, iterable)
```

```
def fn(n):
    return n ** 2

numbers = [1, 4, 9, 16]
numbers = map(fn, numbers)
```

Das Mapping kann natürlich auch wie filter mit List-Comprehensions gelöst werden.

2. Higher Order Funktion filter

filter() erstellt ein Filter-Objekt mit allen Elementen, die den von der bereitgestellten Funktion implementierten Test bestehen. Der Rückgabewert von filter() ist das Filter-Objekt, welches ein Iterator ist.

```
def fn(word):
    return len(word) > 5

words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present']

filtered = filter(fn, words)
```

3. Higher Order reduce

Die Funktion reduce aus dem Modul functools reduziert eine Sequenz anhand einer Funktion auf einen Wert. Optional lässt sich ein Startwert übergeben. Im Beispiel wird das Produkt von nums und dem Startwert errechnet.

```
def fn(x, y):
    return x * y


nums = [1, 4, 9]
product = reduce(fn, nums, 10) # 10 * 1 * 4 * 9
```