

Funktionales Programmieren in Python

Prof. Dr. Rüdiger Weis

Beuth Hochschule für Technik Berlin

- 1 lambda Funktionen
- 2 apply
- 3 zip
- 4 filter
- 5 map
- 6 reduce
- 7 List Comprehension

Funktionales Programmieren

*"Wer nicht funktional programmiert,
programmiert disfunktional."*

- Eleganter Syntax beeinflusst von Haskell
<http://www.haskell.org/>
- Optional

MIT wechselt zu Python

Waseem S. Daher, EECS Revamps Course Structure

" The difference is that programming will be done in Python and not Scheme. "

<http://www-tech.mit.edu/V125/N65/coursevi.html>

lambda

lambda

lambda [<arg>[, <arg>...]]: <ausdruck>

- Anonyme Funktion
- Ausdruck

Beispiel

```
>>> lambda x : x ** 2
<function <lambda> at 0xb7df7e2c>
>>> f = lambda x : x ** 2
>>> f
<function <lambda> at 0xb7df7e64>
>>> f(2)
4
>>> f = [lambda x : x ** 2, lambda x : x ** 3]
>>> f[0](42)
1764
```

apply

apply

```
apply(<funktion >, <tupel >)
```

- Äquivalent

```
<funktion >(<tupel >)
```

- Primitive für Funktionales Programmieren

zip Funktion

zip

zip(<sequenz >, [<sequenz >, ...])

- zip ist eingebaute Funktion.
- zip liefert eine Liste von Tupeln zurück in der das i -te Tupel aus den i -ten Elementen der Eingabesequenzen besteht.

Beispiel: zip

```
>>> zip([1, 2, 3], ['a', 'b', 'c'])  
[(1, 'a'), (2, 'b'), (3, 'c')]  
>>> zip([1, 2], ['a', 'b', 'c'])  
[(1, 'a'), (2, 'b')]  
>>> zip([1, 2])  
[(1, ), (2, )]
```

filter

filter

filter(<funktion >, <sequenz >)

- filter ist Schlüsselwort.
- filter gibt eine Liste von Elementen der Eingabesequenz zurück, für welche die Funktionanwendung True liefert.

Beispiel: filter

```
>>> # Liste der geraden Zahlen zwischen 0 und 24
... filter(lambda x : x % 2 == 0, range(25))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
>>> # Liste der ungeraden Zahlen zwischen 0 und 24
... filter(lambda x : x % 2 != 0, range(25))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23]
```

Beispiel: Primzahlen

```
def prime(n):  
    if n == 0 or n == 1:  
        return False  
    i = 2  
    while i * i <= n:  
        if n % i == 0:  
            return False  
        i = i + 1  
    return True
```

```
>>> filter(prime, range(50))  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

map

map

```
map(<funktion >, <sequenz >, [<sequenz >, ...])
```

- map ist Schlüsselwort.
- map wendet eine Funktion auf alle Elemente einer Sequenz an.
- Als Ergebnisse wird eine Liste mit den Funktionsergebnissen zurückgeliefert.

Beispiel: map

```
>>> map(ord, 'Spam')  
[83, 112, 97, 109]
```

```
>>> map(lambda x : x * x * x, range(10))  
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

reduce

reduce

reduce(<funktion >, <sequenz >)

- reduce ist Schlüsselwort.
- reduce wendet von links nach rechts nacheinander eine Funktion mit zwei Argumenten auf alle Elemente der Sequenz an bis ein einzelner Wert entsteht.
- Falls die Sequenz nur einen einzigen Wert besitzt, wird der einzige Sequenzwert zurückgeliefert

Beispiel: reduce

```
>>> reduce(lambda x, y : x + y, range(1, 6))
```

```
15
```

```
>>> (((((1 + 2) + 3) + 4) + 5)
```

```
15
```

```
>>> reduce(lambda x, y : x * 1000 + y, [42])
```

```
42
```


reduce mit Initialwert

reduce mit Initialwert

```
reduce(<funktion >, <sequenz >[, <initwert >])
```

- Mit mit <initwert> kann ein Initialwert angegeben werden. Dieser wird der Sequenz vorangestellt.
- Falls die Sequenz nur einen einzigen Wert besitzt und kein Initialwert angegeben wird, wird der einzige Sequenzwert zurückgeliefert.

Beispiel: reduce mit Initialwert

```
>>> reduce(lambda x, y : x * 1000 + y, [23, 42])  
23042  
>>> reduce(lambda x, y : x * 1000 + y, [42], 23)  
23042  
>>> reduce(lambda x, y : x * 1000 + y, [42])  
42
```

List Comprehension

List Comprehension

```
[<ausdruck> [ for <var> in <seq> ...  
              [ if <bedingung> ... ] ]  
]
```

- Listenbildung mit for und if Anweisung.

Äquivalente Schreibweisen

```
l = [expression for expr in sequence1
      for expr2 in sequence2 ...
      for exprN in sequenceN
      if condition]
```

```
l = []
for expr1 in sequence1:
    for expr2 in sequence2:
        ...
        for exprN in sequenceN:
            if (condition):
                l.append(expression)
```

map, filter

- $\text{map}(f, \text{seq}) \equiv [f(i) \text{ for } i \text{ in } \text{seq}]$
- $\text{filter}(f, \text{seq}) \equiv [i \text{ for } i \text{ in } \text{seq} \text{ if } f(i)]$

Beispiel: List Comprehension

```
>>> [x * x * x for x in range(1, 11)]  
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]  
>>> [x * x * x for x in range(1, 11) if x % 3 == 0]  
[27, 216, 729]
```

©opyleft

©opyleft

- Erstellt mit Freier Software
- © Rüdiger Weis, Berlin 2005 – 2018
- unter der GNU Free Documentation License.