Funktionen in Python

Prof. Dr. Rüdiger Weis

Beuth Hochschule für Technik Berlin

def Anweisung

def

```
def funktionsname([<arg>]):
    ['']Dokumentationsstring'']

    <anweisungen>
    [return <ausdruck>]
```

- `def` erstellt ein Funktionsobjekt mit 0 oder mehr Parametern.
- Dokumentationsstring ist optional aber empfehlenswert.
- `return` liefert Ausdruck. None wenn nicht vorhanden.

Beispiel: TuNix

```
>>> def TuNix(): pass
 ...
>>> TuNix
<function TuNix at 0xb7df7e64>
>>> print TuNix()
None
```

Argumentübergabe

Argumente werden an lokale Variablen der Funktion übergeben.

Argumentübergabe

- Unveränderliche Objekte $\approx$ call by value
- call by reference am besten mit Tupel als Funktionsrückgabe.

Beispiel: 'call by value'

```
def f1(x,y):
"callby value"

    print('Parameter x :'), x
    print('Parameter y :'), y
    x = x * 3
    y = y * 2
    print('Aendere Parameter x :'), x
    print('Aendere Parameter y :'), y
```

Beispiel: 'call by value'

```
>>> x = 1
>>> y = 2
>>> f1(x, y)
Parameter x : 1
Parameter y : 2
Aendere Parameter x : 3
Aendere Parameter y : 4
>>> x
1
>>> y
2
```

Beispiel: Veränderliche Datentypen

```
def f2(x,y):
"Veraenderliche Datentypen"

    print('Parameter x :'), x
    print('Parameter y :'), y
    x[0] = 'Hoppla'
    y.append(42)
    print ('Aendere Parameter x :'), x
    print('Aendere Parameter y :'), y
```

Beispiel: Veränderliche Datentypen (II)

```
>>> x = [1, 2, 3]
>>> y = [11, 12, 13]
>>> f2(x, y)
Parameter x : [1, 2, 3]
Parameter y : [11, 12, 13]
Aendere Parameter x : ['Hoppla', 2, 3]
Aendere Parameter y : [11, 12, 13, 42]
>>> x
['Hoppla', 2, 3]
>>> y
[11, 12, 13, 42]
```

Empfehlung: 'call by reference'

http://www.python.org/doc/faq/programming/

How do I write a function with output parameters (call by reference)?

By returning a tuple of the results:

```
def func2(a, b):
    a = 'new-value'          # a and b are local names
    b = b + 1                # assigned to new objects
    return a, b              # return new values

x, y = 'old-value', 99
x, y = func2(x, y)
print x, y                   # output: new-value 100
```

This is almost always the clearest solution.

Gültigkeitsbereich

Gültigkeitsbereich

- Funktionen definieren einen eigenen Gültigkeitsbereich.
- Lokale Variablen

Beispiel: Lokale Variablen 1

```
>>> def f(x):
...     y = 2
...     return(x + y)
...
>>> f(1)
3
>>> y
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'y' is not defined
```

Beispiel: Lokale Variablen 2

```
>>> y = 42
>>> def f(x):
...         y = 2
...         return(x + y)
...
>>> f(1)
3
>>> y
42
```

global

- Das Schlüsselwort `global` erlaubt den Zugriff auf globale Variablen.
- Sparsam verwenden.

LGB-Regel

LGB-Regel

1. **L**ocal
2. **G**lobal
3. **B**uilt-In

Beispiel: Fibonacci Zahlen

Rekursive Folge

- $f_0 = 0$
- $f_1 = 1$
- $f_n = f_{n-1} + f_{n-2}, \forall n \geq 2$

`http://de.wikipedia.org/wiki/Fibonacci-Zahlen`

Exkurs: Fibonacci rekursiv

```
def fibonacci(n):
"Fibonacci rekursiv"

    if n >= 2:
        return(fibonacci(n - 1) + fibonacci(n - 2))
    elif n == 1:
        return 1
    elif n == 0:
        return 0
```

Exkurs: Fibonacci mittels Generator

```
def fib():
"Fibonacci mittels Generator"

    a, b = 0, 1
    while True:
        yield b
        a, b = b, a + b
```

http://www.python.org/dev/peps/pep-0255/

Default Argumente

Default Parameter

```
def funktionsname([<arg>], [<arg>=<vorgabewert>]):

    <anweisungen>
```

- Vorgabe Parameter müssen **hinter** den Stellungsparametern stehen.
- Werden beim Aufruf ein Parameter weggelassen, wird der Vorgabewert verwendet.

Beispiel

```
>>> def f(antwort=42): print antwort
...
>>> f()
42
>>> f('Kein Plan!')
Kein Plan!
```

Gültigkeit Default Parameter (1)

Gültigkeit Default Parameter

Parameter sind bei der Definition der Default-Werte nicht nutzbar.

```
>>> def mische(sequenz, start=0, ende=len(sequenz)): pass
...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'sequenz' is not defined
```

Gültigkeit Default Parameter (2)

Gültigkeit Default Parameter (2)

- Default-Werte werden beim ersten Funktionsaufruf ausgewertet.
- Vorsicht bei veränderlichen Datentypen.

Beispiel: Gültigkeit Default Parameter

```
def verkette(a, b=[]):

    b.append("Spam")
    return a + b


print verkette([1])
print verkette([1])

# Ergibt:
#[1, 'Spam']
#[1, 'Spam', 'Spam']
```

Workaround: Defaultparameter None

- Workaround: `None` und anschliessende Zuweisung.

```
def verkette(a, b=None):

    if b == None:
        b = []
    b.append("Spam")
    return a + b


print verkette([1])
print verkette([1])

# Ergibt:
#[1, 'Spam']
#[1, 'Spam']
```

Aufruf mit Schlüsselwort

Aufruf mit Schlüsselwort

```
f([<arg>][, ..., ][<arg>=<wert>])
```

- Werte können beim Aufruf explizit einem Parameter-Namen zugewiesen werden.
- Alle Werte ohne Default-Wert müssen als Schlüsselwort-Parameter oder Stellungs-Parameter übergeben werden.

Variable Argumente

Variable Argumente

- *<name> sammelt Position-Parameter in Tupel.
- **<name> sammelt Schlüsselwort-Parameter in Dictonary.

- Verwendung am Ende der Parameter-Liste.
- Reihenfolge:
  - *
  - **

Beispiel: Positionsparameter

```
>>> def f(*parameter): print parameter
...
>>> f(1, 2, 3)
(1, 2, 3)
>>> f('Brian', 'Judith')
('Brian', 'Judith')
>>> f(('Brian', 'Judith'))
(('Brian', 'Judith'),)
```

Beispiel: Schlüsselworte

```
>>> def f(**schluesselworte): print schluesselworte
...
>>> f(fruehstueck='spam')
{'fruehstueck': 'spam'}
>>> f(fruehstueck='spam', mittagessen='spam with eggs')
{'fruehstueck': 'spam', 'mittagessen': 'spam with eggs'}
>>> f(x=23, y=42, essen=('spam', 69))
{'y': 42, 'x': 23, 'essen': ('spam', 69)}
```

Beispiel

```
# Falsche Reihenfolge :
#
>>> def f(**worte, *para): print para, worte
  File "<stdin>", line 1
    def f(**worte, *para): print para, worte
                  ^
SyntaxError: invalid syntax
#
# Richtige Reihenfolge: Position vorne
#
>>> def f(*para, **worte): print para, worte
...
```

Beispiel

```
# Falsche Reihenfolge
#
>>> f(a=1, 11, 2)
SyntaxError: non-keyword arg after keyword arg
#
# Richtige Reihenfolge: Position vorne
#
>>> f(11, 2, a=1)
(11, 2) {'a': 1}
```

©opyleft

©opyleft

- Erstellt mit Freier Software
- © Rüdiger Weis, Berlin 2005 – 2011
- unter der GNU Free Documentation License.