Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

FS24 CAS PML - Python

Niklaus Johner

niklausbernhard.johner@bfh.ch

FS24 CAS PML - Python

11. Funktionen

Vorwort: Namespace

- Ein *namespace* kann man einfach als ein *dictionary* welcher Variablenname:Wert Paare enthält.

```
In [1]: a = 3; b = 5

In [2]: locals().keys()
Out[2]: dict_keys(['__name__', '__doc__', '__package__',
 '__loader__', '__spec__', '__builtin__', '__builtins__'
, '_ih', '_oh', '_dh', 'In', 'Out', 'get_ipython', 'exit
', 'quit', 'open', '_', '__', '___', '_i', '_ii', '_iii'
, '_i1', 'a', 'b', '_i2'])

In [3]: locals()["a"]
Out[3]: 3
```

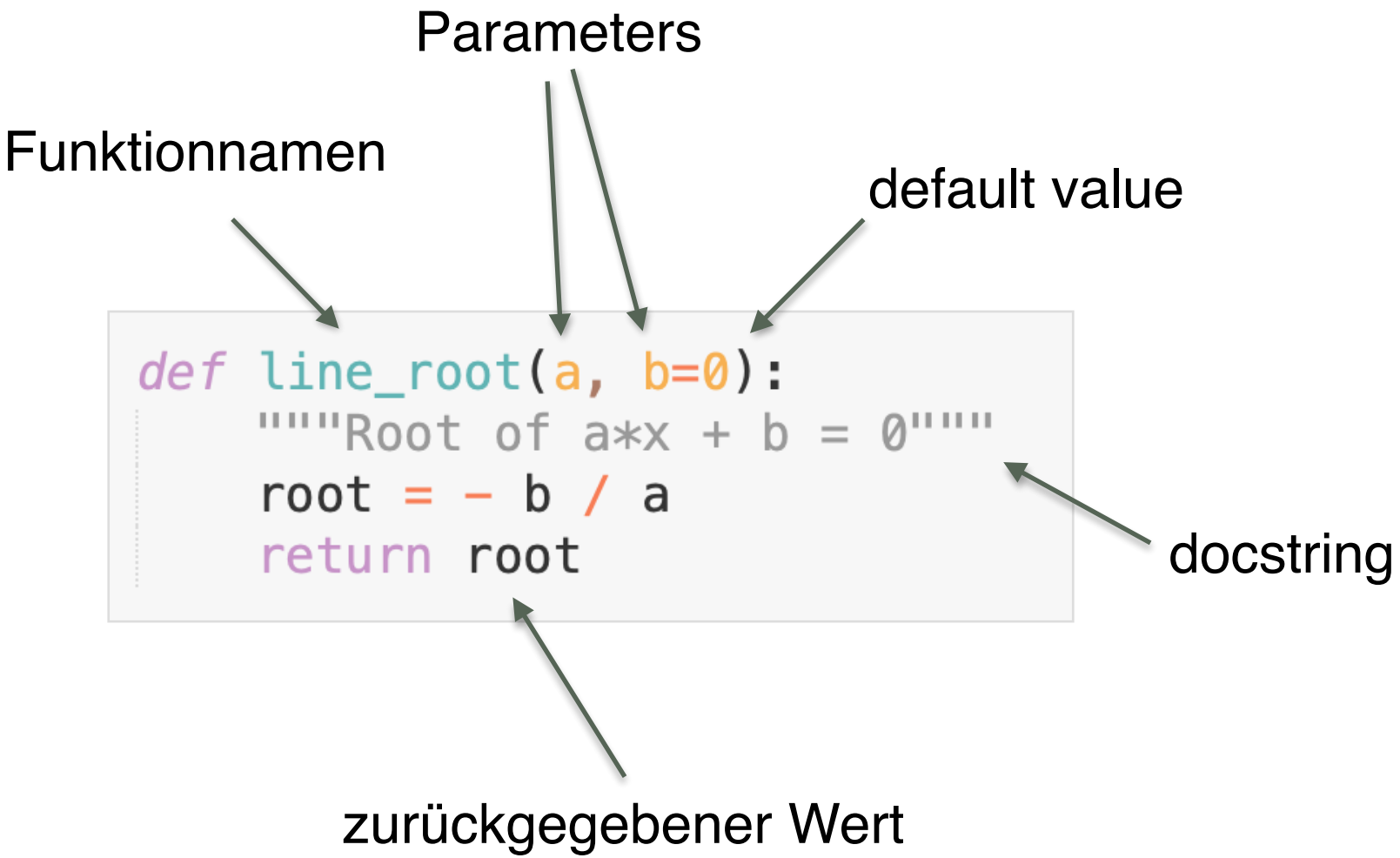
Funktionen

- Funktion:
  - Serie von Anweisungen die dem Aufrufenden einen Wert zurück gibt.
  - Kann einen oder mehrere Parameter annehmen
  - Wird mit *def* definiert
  - *return* definiert den Wert der zurückgegeben wird

```
def mean(v):
    return sum(v) / len(v)

l = [1, 3, 2]
print("mean =", mean(l))
```

Funktionen


Parameters
Funktionnamen
default value
def line_root(a, b=0):
"""Root of a*x + b = 0"""
root = - b / a
return root
docstring
zurückgegebener Wert

Funktionen

- *Funktionnamen* wird im namespace wo die Funktion definiert ist, kreiert
- *Zurückgegebener Wert*: wird dem Aufrufenden zurückgegeben
- *Argumente* werden im namespace der Funktion den entsprechenden Parametern zugewiesen

```
def line_root(a, b=0):
    """Root of a*x + b = 0"""
    root = - b / a
    return root
```

```
In [8]: res = line_root(2, 3)
   ...: print(f"root = {res}")
root = -1.5
```

Funktionen: Parameter

- Parameter welche keinen default Wert haben sind obligatorisch
- Parameter die ein default Wert haben sind fakultativ

```
In [17]: line_root(2, 3)
Out[17]: -1.5

In [18]: line_root(2)
Out[18]: 0.0

In [19]: line_root()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 line_root()

TypeError: line_root() missing 1 required positional argument: 'a'
```

Funktionen: Parameter

- Argumente werden angegeben als
  - *positional-argument* (werden über ihre Position gemapped)
  - *keyword-argument* (werden über ihre Key gemapped)
- Beim Aufrufen müssen zuerst alle *positional-arguments* dann die *keyword-arguments* angegeben werden

```
In [14]: def print_args(a, b, c=None, d=None):
    ...:     print("a={}, b={}, c={}, d={}".format(a, b, c, d))
    ...:

In [15]: print_args(1, 2, 3, d=4)
a=1, b=2, c=3, d=4

In [16]: print_args(1, c=4, b=5)
a=1, b=5, c=4, d=None
```

Funktionen: docstring

- docstring wird von der help Funktion angezeigt

```
def power(a):
    """Calculates successive powers of a variable
    Input: a: variable from which powers are calculated
    Return: list containing a, a**2 and a**3"""
    return [a, a*a, a*a*a]
```

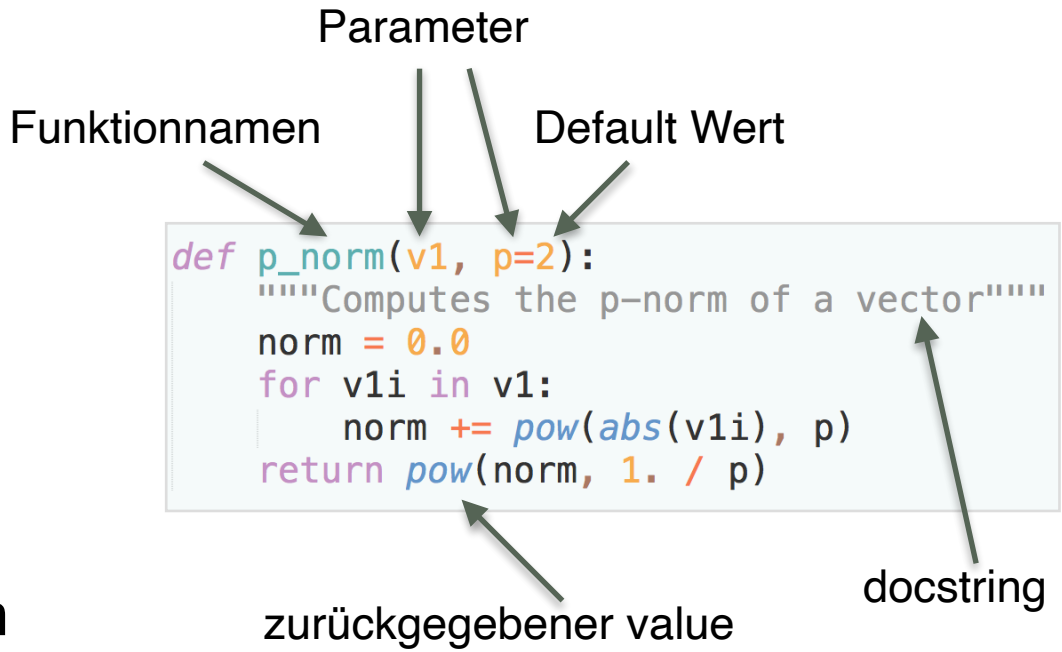
```
In [5]: help(power)
   ...:

Help on function power in module __main__:

power(a)
    Calculates successive powers of a variable
    Input: a: variable from which powers are calculated
    Return: list containing a, a**2 and a**3
```

Übungen

- *Funktionnamen* wird im namespace wo die Funktion definiert ist, kreiert
- *Argumente* werden im namespace der Funktion den *Parameter* zugewiesen
- *Zurückgegebener Wert*: wird dem Aufrufenden zurückgegeben
- *docstring* wird von der *help* Funktion gebraucht



Zusätzliche Folien

Funktionen: *args und **kwargs

```
def print_args(a, b, c=None, d=None):
    print("a={}, b={}, c={}, d={}".format(a, b, c, d))
```

- **list* unpacks the elements from *list*

```
In [71]: print_args(*[1, 2])# print_args(1, 2)
a=1, b=2, c=None, d=None

In [72]: print_args(*[1, 2], d=4)# print_args(1, 2, d=4)
a=1, b=2, c=None, d=4
```

- ***dict* unpacks the items from *dict*

```
In [73]: # print_args(1, 2, c=2, d=4)
    ...: print_args(1, 2, **{"c": 2, "d": 3})
a=1, b=2, c=2, d=3
```

Funktionen: *args und **kwargs

- In einer Funktiondefinition kann man die * und ** Syntax auch brauchen:
  - Konventionell braucht man **args* und ***kwargs*
  - Alle zusätzlichen *positional arguments* werden in *args* gepackt
  - Alle zusätzlichen *keyword arguments* werden in *kwarg* gepackt

```
In [99]: def print_args(a, *args, k=1, **kwargs):
    ...:     print("args:", args, "kwargs:", kwargs)
    ...:

In [100]: print_args(1, k=2)
args: () kwargs: {}

In [101]: print_args(1, 2, c=2)
args: (2,) kwargs: {'c': 2}
```