Funktionen

Funktionen sind kleine Programme die eine einfache Aufgabe ausführen und durch ihren Namen aufgerufen werden können. In anderen Sprachen werden sie auch Unterprogramme oder Subroutinen genannt.

An **Funktionen** können Parameter übergeben werden.
Funktionen können Ergebnisse zurückgeben.

Python bringt **eingebaute Funktionen** mit und ermöglicht die **Definition eigener Funktionen**.

Eingebaute Funktionen

Micropython bringt eingebaute Funktionen mit. Hier eine kleine Auswahl:

```
>>> import builtins
>>> dir(builtins)
[... , 'abs', 'all', 'any', 'bool', 'bytearray', 'bytes', 'callable', 'chr',
'classmethod', 'dict', 'dir', 'divmod', 'eval', 'exec', 'getattr', 'globals',
'hasattr', 'hash', 'id', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'list', 'locals', 'map', 'next', 'object', 'open', 'ord', 'pow', 'print',
'range', 'repr', 'round', 'set', 'setattr', 'sorted', 'staticmethod', 'str',
'sum', 'super', 'tuple', 'type', 'zip', ... 'bin', 'compile', 'complex',
'delattr', 'enumerate', 'execfile', 'filter', 'float', 'frozenset', 'help',
'hex', 'input', 'max', 'memoryview', 'min', 'oct', 'property', 'reversed',
'slice']
```

Eingebaute Funktionen

abs()

```
>>> abs(-25)
25

>>> l
[1, 2, 3]
```

max()

```
>>> max(l)
3
```

min()

```
>>> min(l)
1
```

pow()

```
>>> pow(2, 3)
8
```

sum()

```
>>> sum(l)
6
```

Eingebaute Funktionen

range()

```
>>> range(5)
Range(0, 5)

>>> for x in range(5):
        print(x)

0
1
2
3
4

>>> for x in range(0, 30, 10):
        print(x)

0
10
20
```

Eingebaute Funktionen

round()

```
>>> x = 5.5678
>>> round(x, 2)
5.57
```

len()

```
>>> s = 'Hallo Micropython'
>>> len(s)
17

>>> l = [1, 2, 3, 4, 5]
>>> len(l)
5
>>>
```

Eingebaute Funktionen

bin()

```
>>> bin(42)
'0b101010'
```

hex()

```
>>> hex(42)
'0x2a'
```

oct()

```
>>> oct(42)
'0o52'
>>>
```

bool()

```
>>> bool(0)
False
>>> bool(42)
True
>>> bool('Hallo')
True
```

Eingebaute Funktionen

float()

```
>>> float(42)
42.0
>>> float('42')
42.0
```

int()

```
>>> int(3.14)
3

>>> int('3.14')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid syntax for integer
with base 10

>>> int('3')
3
```

Eingebaute Funktionen

str(zahl)

```
>>> str(3.14)
'3.14'
>>> s = str(3.14)
>>> type(s)
<class 'str'>
>>>
```

del(var)

```
>>> d = dict({'x1':0, 'y1':0, 'x2':10, 'y2':20})
>>> d
{'x1': 0, 'y1': 0, 'y2': 20, 'x2': 10}
>>> del(d['y1'])
>>> d
{'x1': 0, 'y2': 20, 'x2': 10}
>>> del(d)
>>> d
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'd' isn't defined
```

Eingebaute Funktionen

list()

```
>>> list('Hallo')
['H', 'a', 'l', 'l', 'o']
```

dict()

```
>>> dict({'x1':0, 'y1':0, 'x2':10, 'y2':20})
{'x1': 0, 'y1': 0, 'y2': 20, 'x2': 10}
```

set()

```
>>> sr = set({1,2,3})
>>> sr
{1, 2, 3}
>>> type(sr)
<class 'set'>
```

Funktionen

`print('Ausgabe Text')`

Ist die Standard-ausgabefunktion.

Sie gibt den Ausgabetext auf die Standardausgabe aus.

In der REPL funktioniert das, wenn Scripte ausgeführt werden aber nicht.

```
>>> print('Hallo')
Hallo
>>> text = 'Micropython'
>>> print(text)
Micropython
>>>
```

Funktionen

`Input()`

Mit der Inputfunktion können Eingaben von der Standard-eingabe geholt werden.

Input liefert immer einen String zurück!

```
>>> ip = input('Wie heißt Du?')
Wie heißt Du?Peter
>>> ip
'Peter'
>>> ip = input('Gebe eine Zahl ein:')
Gebe eine Zahl ein:42
>>> ip
'42'
>>>
```

Eigene Funktionen schreiben

Eine Funktion wird mit **def** erstellt. Es folgt der Name der Funktion und Klammern **()**. Diese sind ein Kennzeichen von Funktionen. In den Klammern werden die Parameter übergeben.

Mit **return** wird die Funktion beendet und der Rückgabewert angegeben.

```
>>> def quadrat(x):
        return x * x

>>> quadrat(5)
25


>>> def test():
        pass
```

Funktionen

Eine Funktion ist ein eigener Namensraum.

Das globale **x** in der REPL und das **x** in der Funktion sind verschiedene Variablen!

Um eine globale Variable innerhalb einer Funktion zu verwenden gibt es das Schlüsselwort **global**.

```
>>> def quadrat(x):
        return x * x

>>> quadrat(5)
25
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' isn't defined
>>>

>>> def quadrat_global():
        global x
        return x * x

>>> x = 3
>>> quadrat_global()
9
>>>
```

Funktionen

Parameter können durch ihre **Position** oder durch ein **Schlüsselwort** identifiziert werden.

```
>>> def volumen(breite, tiefe, hoehe):
        vol = breite * tiefe * hoehe
        return vol

>>> volumen(2, 3, 4)
24

>>> volumen(tiefe = 3, hoehe = 4, breite = 3)
36
```

Funktionen

Parameter können mit einem **default** Wert belegt werden.

```
>>> def volumen(breite=2, tiefe=3, hoehe=4):
        vol = breite * tiefe * hoehe
        return vol

>>> volumen()
24
>>> volumen(5)
60
>>> volumen(,5)
Traceback (most recent call last):
  File "<stdin>", line 1
SyntaxError: invalid syntax

>>> volumen(tiefe=7)
56
>>>
```

Funktionen

Rückgabewert einer Funktion.

Üblicherweise werden Funktionen mit der **`return`** Anweisung beendet.

Es geht aber auch ohne **`return`**. Dann wird **`None`** zurückgegeben.

```
>>> def anzeigen(text):
        print(text)

>>> anzeigen('Hallo')
Hallo
>>> r = anzeigen('Hallo')
Hallo
>>> r
>>> type(r)
<class 'NoneType'>
>>>
```

Funktionen

Funktionen mit Rückgabewert.

Mit Hilfe von **`return`** kann ein Wert zurückgegeben werden.

Wenn nichts hinter **`return`** angegeben wird, wird auch **`None`** zurückgegeben.

```
>>> def test():
        return
>>> type(test())
<class 'NoneType'>

>>> def quadrat(x):
        y = x * x
        return y


>>> def quadrat(x):
        return x * x
```

Funktionen

Funktionen mit mehreren Rückgabewerten.

Hinter **`return`** kann nicht nur ein Wert angegeben werden. Es können mit Komma getrennt mehrere Werte zurückgegeben werden.

```
>>> def quadrat_zahlen(x):
        return x, x * x

>>> quadrat_zahlen(5)
(5, 25)
>>> type(quadrat_zahlen(5))
<class 'tuple'>

>>> a, b = quadrat_zahlen(5)
>>> a
5
>>> b
25
>>>
```

Funktionen