

Einführung in die Programmierung

Helmut Schmid

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilian-Universität München
`schmid@cis.uni-muenchen.de`

Stand 7. Oktober 2026

Termine

- Vorlesungen

- ▶ Python (Schmid): **Dienstag 10-12 Uhr** (c.t.) in BU101

Inhalt: Einführung in die Python-Programmierung

- ▶ Linux (Schäfer): **Freitag 14-15 Uhr** (c.t.) in BU101

Inhalt: Einführung in das Linux-Betriebssystem und Linux-Tools

- Übungen (Schmid): **Mittwoch 10-12 Uhr** (c.t.) in L155

Inhalt: Programmierübungen

- Tutorien ab der 2. Vorlesungswoche

- ▶ Tutorium 1 (Quan Kuang): **Donnerstag 10-12 Uhr** (c.t.) in BU101

- ▶ Tutorium 2 (Su-Min Choe): **Freitag 12-14 Uhr** (c.t.), in ???

Inhalt: Stoffwiederholung, Programmierübungen, Fragen

Organisatorisches

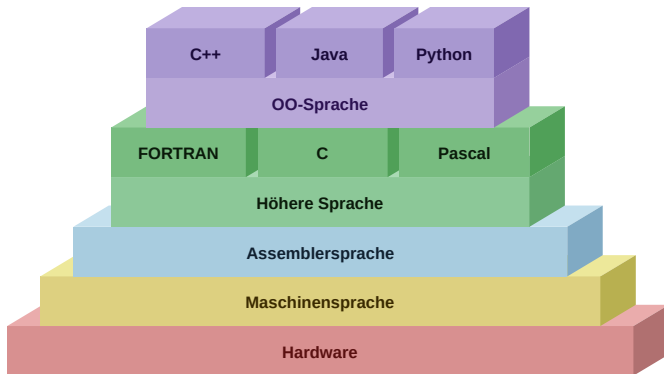
- Kursanmeldung
 - ▶ Sie sollten bereits zu **Kurs** und **Übung** angemeldet sein.
 - ▶ Vergessen Sie nicht, sich Anfang Januar für die **Prüfung** anzumelden.
- voraussichtlicher **Klausurtermin**: in der Woche nach Vorlesungsende
- Kursseite mit Vorlesungsfolien, Übungsaufgaben etc.
<https://www.cis.uni-muenchen.de/~schmid/lehre/EIP>

Inhalte

- Vorlesung
 - ▶ Theorie
 - ▶ Praktische Beispiele
- Übungen
 - ▶ Interaktives Programmieren von Übungsaufgaben
- Tutorium
 - ▶ Besprechung der Übungsaufgaben
 - ▶ Hilfe bei der Programmierung

Ihre Fragen sind immer willkommen!

Programmiersprachen



Assembler & Maschinensprache

- Programm in Maschinensprache

- ▶ Zahlenfolge im Speicher `B8 05 00 00 00 83 C0 03` (hexadezimal)
- ▶ kann direkt vom Prozessor ausgeführt werden
- ▶ Jede Prozessorenfamilie verwendet ihre eigene Maschinensprache.
- ▶ Beispielsprachen:
 - ★ 32 Bit: Intel x86, ARMv7
 - ★ 64 Bit: AMD x86-64, ARMv8

- Assembler-Programm

- ▶ Folge von Befehlen mit Argumenten `MOV EAX, 5` `ADD EAX, 3`
- ▶ Assembler-Code ist viel lesbarer als Maschinencode und wird 1:1 in Maschinencode übersetzt: `B8 05 00 00 00` `83 C0 03`

Compilersprachen

- verwenden komplexe Befehle wie `printf("Hello world\n")`
- Der Code wird von einem Compiler in Maschinencode übersetzt.
- keine 1:1-Übersetzung
- Beispielsprachen: C, C++, C#, Rust
- Vor- und Nachteile
 - + Die Verarbeitung ist sehr schnell.
 - + Syntaxfehler werden bereits beim Kompilieren erkannt.
 - + Der Quellcode des Programmes muss nicht an Anwender weitergegeben werden.
 - Der kompilierte Code läuft nur auf Rechnern mit bestimmten Prozessoren und Betriebssystemen.
 - Programmfehler führen zu unkontrollierten Programmabstürzen.

Interpreter-Sprachen

- Der Programmcode wird nicht in Maschinencode kompiliert sondern direkt von einem **Interpreter**-Programm verarbeitet.
- Beispielsprachen: **Python**, Java, PHP, Ruby
- Der Interpreter simuliert einen Computer, der bspw. Python ausführen kann.
- Vor- und Nachteile (komplementär zu Compilersprachen)
 - + Plattform-Unabhängigkeit: Der Code läuft auf allen Rechnern, auf denen ein Interpreter installiert ist.
 - + Programmfehler werden vom Interpreter abgefangen, der eine informative Fehlermeldung ausgibt.
 - Der Interpreter braucht viel Speicherplatz.
 - Die Verarbeitung ist langsamer.
 - Syntaxfehler werden nicht sofort erkannt.
 - Anwender erhalten den Quellcode des Programmes und können ihn einfach modifizieren.

- Der Python-Interpreter übersetzt Programme zunächst in **Bytecode**.
- Der Bytecode wird dann von der **Python Virtual Machine** ausgeführt.
- Python ist also sowohl eine Compilersprache als auch eine Interpretersprache.
- Um den Bytecode nicht bei jedem Start neu kompilieren zu müssen, wird er in einer Datei mit der Endung `.pyc` gespeichert.

Objektorientierung

- Python ist eine objektorientierte Programmiersprache.
- Alle Python-Daten sind **Objekte**.
- Ein Objekt fasst **Daten** (Attribute) und **Funktionen** (Methoden), welche auf die Daten zugreifen, zu einer Einheit zusammen.
- Das Innere von Objekten wird gegenüber dem übrigen Programm **abgekapselt**, wodurch Programme übersichtlicher werden.

Bits und Bytes

- Computer repräsentieren alle Informationen als **Bitfolgen**.
- Ein Bit hat die möglichen Werte **True** (1) und **False** (0)
- 8 Bits werden zu einem **Byte zusammengefasst**.
- Ein 64-Bit-Computer fasst 4 Bytes (=64 Bits) zu einem **Maschinenwort** zusammen.
- Computer lesen, schreiben und verarbeiten Maschinenwörter immer als Einheit.
- Ältere Prozessoren verarbeiteten 32 Bit (Intel 80386), 16 Bit (Intel 8086) oder auch nur 8 Bit (Intel 8008) auf einmal.

Dezimalzahlen und Binärzahlen

Dezimalzahl 183

	1. Ziffer	2. Ziffer	3. Ziffer
Wert	1	8	3
Multiplikator	100	10	1
entspricht	10^2	10^1	10^0

Wert: $1 \times 100 + 8 \times 10 + 3 \times 1 = 183$

Dezimalzahlen und Binärzahlen

Dezimalzahl 183

	1. Ziffer	2. Ziffer	3. Ziffer
Wert	1	8	3
Multiplikator	100	10	1
entspricht	10^2	10^1	10^0

Wert: $1 \times 100 + 8 \times 10 + 3 \times 1 = 183$

Binärzahl 10110111

Wert	1	0	1	1	0	1	1	1
Multiplikator	128	64	32	16	8	4	2	1
entspricht	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Wert: $1 \times 128 + 0 \times 64 + 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 183$

Hexadezimal- und Oktalzahlen

16 Ziffern

Ziffer	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Wert	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hexadezimalzahl B7

Ziffer	B	7
Wert	11	7
Multiplikator	16	1
entspricht	16^1	16^0

Wert: ???

Hexadezimal- und Oktalzahlen

16 Ziffern

Ziffer	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Wert	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hexadezimalzahl B7

Ziffer	B	7
Wert	11	7
Multiplikator	16	1
entspricht	16^1	16^0

Wert: $11 \times 16 + 7 \times 1 = 183$

Hexadezimal- und Oktalzahlen

16 Ziffern

Ziffer	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Wert	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hexadezimalzahl B7

Ziffer	B	7
Wert	11	7
Multiplikator	16	1
entspricht	16^1	16^0

Wert: $11 \times 16 + 7 \times 1 = 183$

Oktalzahl 267

Ziffer	2	6	7
Wert	2	6	7
Multiplikator	64	8	1
entspricht	8^2	8^1	8^0

Wert: ???

Hexadezimal- und Oktalzahlen

16 Ziffern

Ziffer	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Wert	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hexadezimalzahl B7

Ziffer	B	7
Wert	11	7
Multiplikator	16	1
entspricht	16^1	16^0

Wert: $11 \times 16 + 7 \times 1 = 183$

Oktalzahl 267

Ziffer	2	6	7
Wert	2	6	7
Multiplikator	64	8	1
entspricht	8^2	8^1	8^0

Wert: $2 \times 64 + 6 \times 8 + 7 \times 1 = 183$

$$183 = 0xB7 = 0o267 = 0b10110111$$

Mega, Giga, Terra, Peta, Exa

Einheit	Abk.	Wert
Byte	B	8 Bits
Kilobyte	kB	1000 B
KByte	KB	1024 B
Megabyte	MB	1024 KB
Gigabyte	GB	1024 MB
Terrabyte	TB	1024 GB
Petabyte	PB	1024 TB
Exabyte	EB	1024 PB

Python-Zahlen

Python kennt folgende Zahlentypen:

- **bool** mit den Werten True und False (bzw. 1 und 0)
- **int** für ganze Zahlen beliebiger Länge
- **float** für Fließkommazahlen mit 64-Bit nach dem IEEE 754-Standard

Wertebereich: ca. $\pm 1.79 \times 10^{308}$ ($\pm 1.79\text{e}308$)

kleinste Zahl größer 0: ca. 5×10^{-324} ($5\text{e-}324$)

Underflow: Der Python-Ausdruck $5\text{e-}324 / 2$ liefert den Wert `0.0`

Overflow: Der Python-Ausdruck $1\text{e+}308 * 2$ liefert den Wert `inf`

Fließkommazahlen (Float)

Beispiel:

1.7976931348623157 e +308
Mantisse Exponent

Nach dem IEEE-Standard 754 gilt für 64-Bit-Float:

- Mantisse 52 Bit
- Exponent 11 Bit
- Vorzeichen 1 Bit

Zeichenkodierung

Python speichert Buchstaben, Ziffern usw. intern als Zahlen und verwendet dafür **Unicode**-Kodierungen.

Beispiele:	Zeichen	Unicode-Kodierung
	A	65
	B	66
	Z	90
	a	97
	z	122
	0	48
	9	57
	ä	228
	😊	128578
	"Null"	0
	"Tabulator"	9

Wie wird der String "Abba" kodiert? ???

Zeichenkodierung

Python speichert Buchstaben, Ziffern usw. intern als Zahlen und verwendet dafür **Unicode**-Kodierungen.

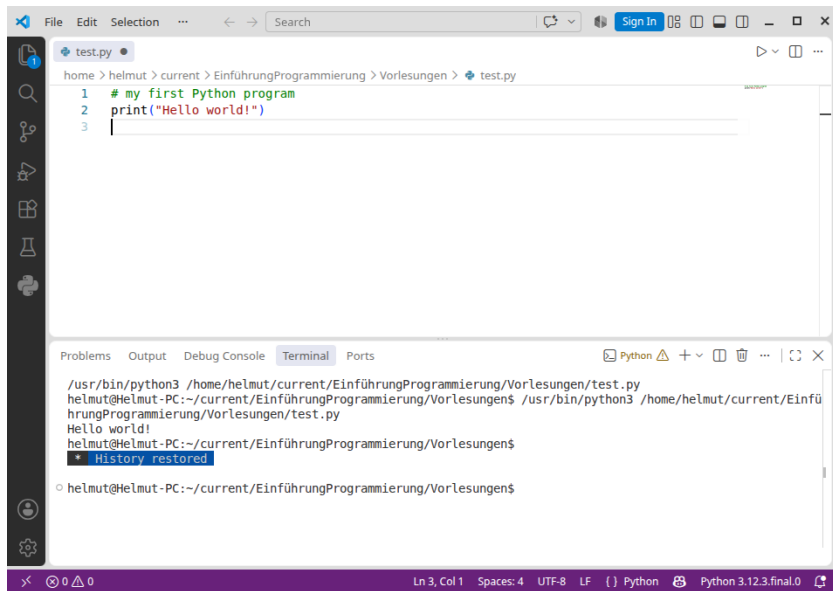
Beispiele:	Zeichen	Unicode-Kodierung
	A	65
	B	66
	Z	90
	a	97
	z	122
	0	48
	9	57
	ä	228
	😊	128578
	"Null"	0
	"Tabulator"	9

Wie wird der String "Abba" kodiert? 65 98 98 97

Womit schreibt man ein Python-Programm?

- Ein Python-Programm ist eine unformatierte **Textdatei**.
- Zum Erstellen einer solchen Datei brauchen Sie einen **Editor**.
- Empfehlung: Installieren sie den Open-Source-Editor **Visual Studio Code**.
- In VS Code können Sie das Programm auch ausführen, testen und debuggen.
- Lassen Sie sich von Ihrer Lieblings-KI die **Installation** von VS Code mit Python auf Ihrem Betriebssystem erklären und installieren Sie es bis nächste Woche.

Visual Studio Code



The screenshot displays the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, and a search bar. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, Extensions, Testing, and Settings. The main editor area shows a file named `test.py` with the following content:

```
1 # my first Python program
2 print("Hello world!")
3
```

Below the editor is the integrated terminal, which shows the execution of the Python script:

```
/usr/bin/python3 /home/helmut/current/EinführungProgrammierung/Vorlesungen/test.py
helmut@Helmut-PC:~/current/EinführungProgrammierung/Vorlesungen$ /usr/bin/python3 /home/helmut/current/EinführungProgrammierung/Vorlesungen/test.py
Hello world!
helmut@Helmut-PC:~/current/EinführungProgrammierung/Vorlesungen$
* History restored
helmut@Helmut-PC:~/current/EinführungProgrammierung/Vorlesungen$
```

The status bar at the bottom indicates the current position (Ln 3, Col 1), spacing (Spaces: 4), encoding (UTF-8), line feed (LF), language (Python), and the Python version (Python 3.12.3.final.0).

Ein erstes Pythonprogramm

Starten Sie den Editor bspw. mit dem Befehl `code test.py` und schreiben Sie:

```
test.py
```

```
print("Hello world!")
```

Speichern Sie das Programm. Öffnen Sie ein Terminal. Wechseln Sie in das Verzeichnis, in dem `test.py` gespeichert wurde, und führen den Befehl `python test.py` aus:

```
> python test.py  
Hello world!
```

In VS Code können Sie das Programm auch mit *Run* → *Run Without Debugging* starten.

Wenn Linux den Befehl `python` nicht kennt, müssen Sie stattdessen `python3` schreiben oder am Ende der Datei `~/.bashrc` folgende Zeile hinzufügen: `alias python=python3`

Einige Linux-Befehle

- `cd ~` wechselt ins Homeverzeichnis bspw. `/home/helmut/`
- `cd src` wechselt ins Unterverzeichnis `src`
- `pwd` gibt den Pfad zum aktuellen Verzeichnis aus
- `cd ..` wechselt ins übergeordnete Verzeichnis
- `ls` zeigt den Inhalt des aktuellen Verzeichnisses an
- `ls sub` zeigt den Inhalt des Unterverzeichnisses `sub` an
- `ls -la` zeigt den Inhalt des aktuellen Verzeichnisses ausführlich an
- `man ls` zeigt Informationen über den Befehl `ls` an

Python interaktiv

Sie können Python auch interaktiv nutzen.

Öffnen Sie dazu ein Terminalfenster und geben Sie den Befehl `python` ein.

```
> python
>>> print("Hello world!")
Hello world!
>>> 3+4*7
31
>>> help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
...
```

Beenden mit STRG+d

Einfache Datenstrukturen

- Wahrheitswerte (Boolean): `True False`
- ganze Zahlen (Integer): `17 -179`
- Fließkommazahlen (Float): `1.75 3.17e-23`
- Zeichenketten (Strings): `'Wie' "geht's" 'geht\'s'`

Einfache Datenstrukturen

- Wahrheitswerte (Boolean): `True False`
- ganze Zahlen (Integer): `17 -179`
- Fließkommazahlen (Float): `1.75 3.17e-23`
- Zeichenketten (Strings): `'Wie' "geht's" 'geht\'s'`

Operatoren

- arithmetische Operatoren: `+ - * / **`
- Zuweisungsoperator: `=`
- Vergleichsoperatoren: `== != > < >= <=`
- logische Operatoren: `and or not`

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```


Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

Python-Ausdrücke

```
> python
>>> 2 ** 3
8
>>> (3 + 4) * 5
35
>>> 2 + 3 == 5
True
>>> 13 / 5
2.6
>>> 13 // 5
2
>>> 13 % 5
3
>>> divmod(13, 5)
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

```
5.0
```

```
>>> 5 == 5.0
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

```
5.0
```

```
>>> 5 == 5.0
```

```
True
```

```
>>> 'abc' + "def"
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

```
5.0
```

```
>>> 5 == 5.0
```

```
True
```

```
>>> 'abc' + "def"
```

```
'abcdef'
```

```
>>> 'cd' in 'abcdef'
```


Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

```
5.0
```

```
>>> 5 == 5.0
```

```
True
```

```
>>> 'abc' + "def"
```

```
'abcdef'
```

```
>>> 'cd' in 'abcdef'
```

```
True
```

```
>>> 'cd' in 'abcdef' and 1 >= 1.5
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

```
5.0
```

```
>>> 5 == 5.0
```

```
True
```

```
>>> 'abc' + "def"
```

```
'abcdef'
```

```
>>> 'cd' in 'abcdef'
```

```
True
```

```
>>> 'cd' in 'abcdef' and 1 >= 1.5
```

```
False
```

```
>>> 1/0
```

Python-Ausdrücke

```
> python
```

```
>>> 2 ** 3
```

```
8
```

```
>>> (3 + 4) * 5
```

```
35
```

```
>>> 2 + 3 == 5
```

```
True
```

```
>>> 13 / 5
```

```
2.6
```

```
>>> 13 // 5
```

```
2
```

```
>>> 13 % 5
```

```
3
```

```
>>> divmod(13, 5)
```

```
(2, 3)
```

```
>>> 2 + 3.0
```

```
5.0
```

```
>>> 5 == 5.0
```

```
True
```

```
>>> 'abc' + "def"
```

```
'abcdef'
```

```
>>> 'cd' in 'abcdef'
```

```
True
```

```
>>> 'cd' in 'abcdef' and 1 >= 1.5
```

```
False
```

```
>>> 1/0
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
ZeroDivisionError: division by zero
```

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3  
>>> b = 2  
>>> a + b
```

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
```

```
>>> b = 2
```

```
>>> a + b
```

```
5
```

```
>>> a ** b
```

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
```

```
>>> b = 2
```

```
>>> a + b
```

```
5
```

```
>>> a ** b
```

```
8
```

```
>>> c = 'abc'
```

```
>>> d = 'def'
```

```
>>> c + d
```

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
```

```
>>> b = 2
```

```
>>> a + b
```

```
5
```

```
>>> a ** b
```

```
8
```

```
>>> c = 'abc'
```

```
>>> d = 'def'
```

```
>>> c + d
```

```
'abcdef'
```

```
>>> a + c
```

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3                                >>> type(a)
```

```
>>> b = 2
```

```
>>> a + b
```

```
5
```

```
>>> a ** b
```

```
8
```

```
>>> c = 'abc'
```

```
>>> d = 'def'
```

```
>>> c + d
```

```
'abcdef'
```

```
>>> a + c
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```


Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
>>> b = 2
>>> a + b
5
```

```
>>> type(a)
<class 'int'>
>>> type(c)
```

```
>>> a ** b
8
```

```
>>> c = 'abc'
>>> d = 'def'
>>> c + d
'abcdef'
>>> a + c
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
>>> b = 2
>>> a + b
5
>>> a ** b
8
>>> c = 'abc'
>>> d = 'def'
>>> c + d
'abcdef'
>>> a + c
```

```
>>> type(a)
<class 'int'>
>>> type(c)
<class 'str'>
>>> id(a)
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
>>> b = 2
>>> a + b
5
>>> a ** b
8
>>> c = 'abc'
>>> d = 'def'
>>> c + d
'abcdef'
>>> a + c
```

```
>>> type(a)
<class 'int'>
>>> type(c)
<class 'str'>
>>> id(a)
11763944
>>> id(3)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
>>> b = 2
>>> a + b
5
>>> a ** b
8
>>> c = 'abc'
>>> d = 'def'
>>> c + d
'abcdef'
>>> a + c
```

```
>>> type(a)
<class 'int'>
>>> type(c)
<class 'str'>
>>> id(a)
11763944
>>> id(3)
11763944
>>> id(4)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Variablen

Variablen speichern Python-Objekte:

```
>>> a = 3
>>> b = 2
>>> a + b
5
>>> a ** b
8
>>> c = 'abc'
>>> d = 'def'
>>> c + d
'abcdef'
>>> a + c
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> type(a)
<class 'int'>
>>> type(c)
<class 'str'>
>>> id(a)
11763944
>>> id(3)
11763944
>>> id(4)
11763976
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

`type(x)` liefert den Typ von x.

`id(x)` liefert eine Nummer, welche das Objekt x eindeutig identifiziert.

Typisierung von Variablen

Python verwendet **dynamische** Typisierung der Variablen

- Eine Variable ist ein Verweis auf ein Objekt.
- Der Typ des Objektes ist beliebig.

```
n = 3
```

```
n = 'Hallo'
```

Statische Typisierung (bspw. in C++)

- Jede Variable muss mit einem bestimmten Typ (int, float, string, etc.) **deklariert** werden, bevor sie verwendet werden kann.

```
int n;
```

```
float f;
```

```
string c;
```

- Die Variable kann keine Werte eines anderen Typs speichern.

```
int n;
```

```
n = 1.7; // Der Compiler meldet hier einen Fehler
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)
```

```
'0b101010'
```

```
>>> hex(42)
```


Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)
'0b101010'
>>> hex(42)
'0x2a'
>>> 0b101010
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

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

>>> hex(42)
'0x2a'

>>> 0b101010
42

>>> 0x2a
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

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

>>> hex(42)
'0x2a'

>>> 0b101010
42

>>> 0x2a
42

>>> int(3.7)
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

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

>>> hex(42)
'0x2a'

>>> 0b101010
42

>>> 0x2a
42

>>> int(3.7)
3

>>> int('17')
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)
```

```
'0b101010'
```

```
>>> hex(42)
```

```
'0x2a'
```

```
>>> 0b101010
```

```
42
```

```
>>> 0x2a
```

```
42
```

```
>>> int(3.7)
```

```
3
```

```
>>> int('17')
```

```
17
```

```
>>> round(3.14159265359, 2)
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)
```

```
'0b101010'
```

```
>>> hex(42)
```

```
'0x2a'
```

```
>>> 0b101010
```

```
42
```

```
>>> 0x2a
```

```
42
```

```
>>> int(3.7)
```

```
3
```

```
>>> int('17')
```

```
17
```

```
>>> round(3.14159265359, 2)
```

```
3.14
```

```
>>> int('3.7')
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)                                >>> float(3)
'0b101010'
>>> hex(42)
'0x2a'
>>> 0b101010
42
>>> 0x2a
42
>>> int(3.7)
3
>>> int('17')
17
>>> round(3.14159265359, 2)
3.14
>>> int('3.7')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)                >>> float(3)
'0b101010'                 3.0
>>> hex(42)                >>> float('13e4')
'0x2a'
>>> 0b101010
42
>>> 0x2a
42
>>> int(3.7)
3
>>> int('17')
17
>>> round(3.14159265359, 2)
3.14
>>> int('3.7')
... invalid literal ...
```


Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)                >>> float(3)
'0b101010'                 3.0
>>> hex(42)                >>> float('13e4')
'0x2a'                     130000.0
>>> 0b101010              >>> str(13.70)
42
>>> 0x2a
42
>>> int(3.7)
3
>>> int('17')
17
>>> round(3.14159265359, 2)
3.14
>>> int('3.7')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)           >>> float(3)
'0b101010'           3.0
>>> hex(42)           >>> float('13e4')
'0x2a'               130000.0
>>> 0b101010         >>> str(13.70)
42                  '13.7'
>>> 0x2a              >>> 10 + 3.70
42
>>> int(3.7)
3
>>> int('17')
17
>>> round(3.14159265359, 2)
3.14
>>> int('3.7')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)           >>> float(3)
'0b101010'           3.0
>>> hex(42)           >>> float('13e4')
'0x2a'               130000.0
>>> 0b101010         >>> str(13.70)
42                   '13.7'
>>> 0x2a              >>> 10 + 3.70
42                   13.7
>>> int(3.7)          >>> ord('a')
3
>>> int('17')
17
>>> round(3.14159265359, 2)
3.14
>>> int('3.7')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)                >>> float(3)
'0b101010'                 3.0
>>> hex(42)                >>> float('13e4')
'0x2a'                     130000.0
>>> 0b101010              >>> str(13.70)
42                          '13.7'
>>> 0x2a                  >>> 10 + 3.70
42                          13.7
>>> int(3.7)              >>> ord('a')
3                           97
>>> int('17')            >>> chr(97)
17
>>> round(3.14159265359, 2)
3.14
>>> int('3.7')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)           >>> float(3)
'0b101010'           3.0
>>> hex(42)           >>> float('13e4')
'0x2a'               130000.0
>>> 0b101010         >>> str(13.70)
42                   '13.7'
>>> 0x2a              >>> 10 + 3.70
42                   13.7
>>> int(3.7)          >>> ord('a')
3                    97
>>> int('17')         >>> chr(97)
17                   'a'
>>> round(3.14159265359, 2) >>> chr(ord('a') + 1)
3.14
>>> int('3.7')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)           >>> float(3)
'0b101010'           3.0
>>> hex(42)           >>> float('13e4')
'0x2a'               130000.0
>>> 0b101010         >>> str(13.70)
42                   '13.7'
>>> 0x2a              >>> 10 + 3.70
42                   13.7
>>> int(3.7)          >>> ord('a')
3                    97
>>> int('17')         >>> chr(97)
17                   'a'
>>> round(3.14159265359, 2) >>> chr(ord('a') + 1)
3.14                 'b'
>>> int('3.7')        >>> ord('ab')
... invalid literal ...
```

Zahlenkonvertierungen (bin, hex, int, float, round, str, ord, chr)

```
>>> bin(42)
```

```
'0b101010'
```

```
>>> hex(42)
```

```
'0x2a'
```

```
>>> 0b101010
```

```
42
```

```
>>> 0x2a
```

```
42
```

```
>>> int(3.7)
```

```
3
```

```
>>> int('17')
```

```
17
```

```
>>> round(3.14159265359, 2)
```

```
3.14
```

```
>>> int('3.7')
```

```
... invalid literal ...
```

```
>>> float(3)
```

```
3.0
```

```
>>> float('13e4')
```

```
130000.0
```

```
>>> str(13.70)
```

```
'13.7'
```

```
>>> 10 + 3.70
```

```
13.7
```

```
>>> ord('a')
```

```
97
```

```
>>> chr(97)
```

```
'a'
```

```
>>> chr(ord('a') + 1)
```

```
'b'
```

```
>>> ord('ab')
```

```
... expected a character ...
```

Listen

```
>>> liste = [1,2,4,8,16]  
>>> liste[0]
```


Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

```
16
```

```
>>> liste[-1]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

```
16
```

```
>>> liste[-1]
```

```
16
```

```
>>> liste[-2]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

```
16
```

```
>>> liste[-1]
```

```
16
```

```
>>> liste[-2]
```

```
8
```

```
>>> liste[1:3]
```

Listen

```
>>> liste = [1,2,4,8,16]    >>> liste[:3]
>>> liste[0]
1
>>> liste[2]
4
>>> len(liste)
5
>>> liste[len(liste)-1]
16
>>> liste[-1]
16
>>> liste[-2]
8
>>> liste[1:3]
[2, 4]
```

Listen

```
>>> liste = [1,2,4,8,16]
>>> liste[0]
1
>>> liste[2]
4
>>> len(liste)
5
>>> liste[len(liste)-1]
16
>>> liste[-1]
16
>>> liste[-2]
8
>>> liste[1:3]
[2, 4]
```

```
>>> liste[:3]
[1, 2, 4]
>>> liste[3:]
```


Listen

```
>>> liste = [1,2,4,8,16]
>>> liste[0]
1
>>> liste[2]
4
>>> len(liste)
5
>>> liste[len(liste)-1]
16
>>> liste[-1]
16
>>> liste[-2]
8
>>> liste[1:3]
[2, 4]
```

```
>>> liste[:3]
[1, 2, 4]
>>> liste[3:]
[8, 16]
>>> liste[-1:]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

```
16
```

```
>>> liste[-1]
```

```
16
```

```
>>> liste[-2]
```

```
8
```

```
>>> liste[1:3]
```

```
[2, 4]
```

```
>>> liste[:3]
```

```
[1, 2, 4]
```

```
>>> liste[3:]
```

```
[8, 16]
```

```
>>> liste[-1:]
```

```
[16]
```

```
>>> liste[0:5:2]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

```
16
```

```
>>> liste[-1]
```

```
16
```

```
>>> liste[-2]
```

```
8
```

```
>>> liste[1:3]
```

```
[2, 4]
```

```
>>> liste[:3]
```

```
[1, 2, 4]
```

```
>>> liste[3:]
```

```
[8, 16]
```

```
>>> liste[-1:]
```

```
[16]
```

```
>>> liste[0:5:2]
```

```
[1, 4, 16]
```

```
>>> liste[::-2]
```

Listen

```
>>> liste = [1,2,4,8,16]
```

```
>>> liste[0]
```

```
1
```

```
>>> liste[2]
```

```
4
```

```
>>> len(liste)
```

```
5
```

```
>>> liste[len(liste)-1]
```

```
16
```

```
>>> liste[-1]
```

```
16
```

```
>>> liste[-2]
```

```
8
```

```
>>> liste[1:3]
```

```
[2, 4]
```

```
>>> liste[:3]
```

```
[1, 2, 4]
```

```
>>> liste[3:]
```

```
[8, 16]
```

```
>>> liste[-1:]
```

```
[16]
```

```
>>> liste[0:5:2]
```

```
[1, 4, 16]
```

```
>>> liste[::2]
```

```
[1, 4, 16]
```

```
>>> liste[:]
```

Listen

```
>>> liste = [1,2,4,8,16]
>>> liste[0]
1
>>> liste[2]
4
>>> len(liste)
5
>>> liste[len(liste)-1]
16
>>> liste[-1]
16
>>> liste[-2]
8
>>> liste[1:3]
[2, 4]
```

```
>>> liste[:3]
[1, 2, 4]
>>> liste[3:]
[8, 16]
>>> liste[-1:]
[16]
>>> liste[0:5:2]
[1, 4, 16]
>>> liste[::2]
[1, 4, 16]
>>> liste[:]
[1, 2, 4, 8, 16]
>>> liste[5]
```

Listen

```
>>> liste = [1,2,4,8,16]
>>> liste[0]
1
>>> liste[2]
4
>>> len(liste)
5
>>> liste[len(liste)-1]
16
>>> liste[-1]
16
>>> liste[-2]
8
>>> liste[1:3]
[2, 4]
```

```
>>> liste[:3]
[1, 2, 4]
>>> liste[3:]
[8, 16]
>>> liste[-1:]
[16]
>>> liste[0:5:2]
[1, 4, 16]
>>> liste[::2]
[1, 4, 16]
>>> liste[:]
[1, 2, 4, 8, 16]
>>> liste[5]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

`len` eingebaute Python-Funktion, anwendbar auf Listen, Tupel, Sets, Dictionaries u.a.

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]  
>>> liste
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]  
>>> liste  
[1, 2, 4, [8, 16]]  
>>> liste2 = liste[-1]  
>>> liste2
```


Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]
>>> liste
[1, 2, 4, [8, 16]]
>>> liste2 = liste[-1]
>>> liste2
[8, 16]
>>> liste2[1]
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]
>>> liste
[1, 2, 4, [8, 16]]
>>> liste2 = liste[-1]
>>> liste2
[8, 16]
>>> liste2[1]
16
>>> liste[-1][1]
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]
>>> liste
[1, 2, 4, [8, 16]]
>>> liste2 = liste[-1]
>>> liste2
[8, 16]
>>> liste2[1]
16
>>> liste[-1][1]
16
>>> liste[1][1]
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]           >>> liste = [1,2,3]
>>> liste                             >>> liste + liste
[1, 2, 4, [8, 16]]
>>> liste2 = liste[-1]
>>> liste2
[8, 16]
>>> liste2[1]
16
>>> liste[-1][1]
16
>>> liste[1][1]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'int' object is not subscriptable
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]
>>> liste
[1, 2, 4, [8, 16]]
>>> liste2 = liste[-1]
>>> liste2
[8, 16]
>>> liste2[1]
16
>>> liste[-1][1]
16
>>> liste[1][1]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'int' object is not subscriptable
```

```
>>> liste = [1,2,3]
>>> liste + liste
[1, 2, 3, 1, 2, 3]
>>> liste * 3
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]
```

```
>>> liste
```

```
[1, 2, 4, [8, 16]]
```

```
>>> liste2 = liste[-1]
```

```
>>> liste2
```

```
[8, 16]
```

```
>>> liste2[1]
```

```
16
```

```
>>> liste[-1][1]
```

```
16
```

```
>>> liste[1][1]
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: 'int' object is not subscriptable
```

```
>>> liste = [1,2,3]
```

```
>>> liste + liste
```

```
[1, 2, 3, 1, 2, 3]
```

```
>>> liste * 3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> 'abc' * 3
```

Listen mit Listen als Elementen

```
>>> liste = [1,2,4,[8,16]]
```

```
>>> liste
```

```
[1, 2, 4, [8, 16]]
```

```
>>> liste2 = liste[-1]
```

```
>>> liste2
```

```
[8, 16]
```

```
>>> liste2[1]
```

```
16
```

```
>>> liste[-1][1]
```

```
16
```

```
>>> liste[1][1]
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: 'int' object is not subscriptable
```

```
>>> liste = [1,2,3]
```

```
>>> liste + liste
```

```
[1, 2, 3, 1, 2, 3]
```

```
>>> liste * 3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> 'abc' * 3
```

```
'abccabccabc'
```

Aliasing

```
>>> liste = [1,2,4,8,16]  
[1, 7, 4, 8, 16]  
>>> liste[0] = liste[0] / 2  
>>> liste
```


Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
>>> liste
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
```

Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
>>> liste
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
>>> liste2
```

Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
>>> liste
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
>>> liste2
[1, 2, 4, 8, 16]
>>> liste2[1] = 7
>>> liste2
```

Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
>>> liste
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
>>> liste2
[1, 2, 4, 8, 16]
>>> liste2[1] = 7
>>> liste2
[1, 7, 4, 8, 16]
>>> liste
```

Aliasing

```
>>> liste = [1,2,4,8,16]    >>> liste[0] = 5
[1, 7, 4, 8, 16]           >>> liste2

>>> liste[0] = liste[0] / 2
>>> liste
[0.5, 2, 4, 8, 16]

>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
>>> liste2
[1, 2, 4, 8, 16]

>>> liste2[1] = 7
>>> liste2
[1, 7, 4, 8, 16]

>>> liste
[1, 7, 4, 8, 16]
```

Aliasing

```
>>> liste = [1,2,4,8,16]    >>> liste[0] = 5
[1, 7, 4, 8, 16]           >>> liste2
                             [5, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2 >>> liste2 = liste[:]
[0.5, 2, 4, 8, 16]         >>> liste[0:2] = [1,2]
>>> liste                   >>> liste
[1, 2, 4, 8, 16]

>>> liste2 = liste
>>> liste2
[1, 2, 4, 8, 16]
>>> liste2[1] = 7
>>> liste2
[1, 7, 4, 8, 16]
>>> liste
[1, 7, 4, 8, 16]
```

Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
>>> liste2
[1, 2, 4, 8, 16]
>>> liste2[1] = 7
>>> liste2
[1, 7, 4, 8, 16]
>>> liste
[1, 7, 4, 8, 16]
```

```
>>> liste[0] = 5
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = liste[:]
>>> liste[0:2] = [1,2]
>>> liste
[1, 2, 4, 8, 16]
>>> liste2
```

Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
[1, 2, 4, 8, 16]
>>> liste2[1] = 7
[1, 7, 4, 8, 16]
>>> liste
[1, 7, 4, 8, 16]

>>> liste[0] = 5
[5, 7, 4, 8, 16]
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = liste[:]
[1, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = list(liste)
[1, 2, 4, 8, 16]
>>> liste2[0] = 9
[9, 2, 4, 8, 16]
>>> [liste, liste2]
[[1, 2, 4, 8, 16], [9, 2, 4, 8, 16]]
```


Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
[1, 2, 4, 8, 16]
>>> liste2[1] = 7
[1, 7, 4, 8, 16]
>>> liste
[1, 7, 4, 8, 16]

>>> liste[0] = 5
[5, 7, 4, 8, 16]
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = liste[:]
[1, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = list(liste)
[1, 2, 4, 8, 16]
>>> liste2[0] = 9
[1, 2, 4, 8, 16]
>>> [liste, liste2]
[[1, 2, 4, 8, 16], [1, 9, 4, 8, 16]]
>>> liste[::2] = [0,0,0]
>>> liste
```

Aliasing

```
>>> liste = [1,2,4,8,16]
[1, 7, 4, 8, 16]
>>> liste[0] = liste[0] / 2
[0.5, 2, 4, 8, 16]
>>> liste[0:2] = [1,2]
[1, 2, 4, 8, 16]

>>> liste2 = liste
[1, 2, 4, 8, 16]
>>> liste2
[1, 7, 4, 8, 16]
>>> liste2[1] = 7
[1, 7, 4, 8, 16]
>>> liste
[1, 7, 4, 8, 16]

>>> liste[0] = 5
[5, 7, 4, 8, 16]
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = liste[:]
>>> liste[0:2] = [1,2]
>>> liste
[1, 2, 4, 8, 16]
>>> liste2
[5, 7, 4, 8, 16]
>>> liste2 = list(liste)
>>> liste2[0] = 9
>>> [liste, liste2]
[[1, 2, 4, 8, 16], [1, 9, 4, 8, 16]]
>>> liste[::2] = [0,0,0]
>>> liste
[0, 2, 0, 8, 0]
```

⇒ Objekte werden bei der Zuweisung zu einer Variablen nicht kopiert!

Methoden

- **Listen** sind wie alle Daten in Python **Objekte**.
- Jedes Objekt gehört zu einer **Klasse**.
- Objekte besitzen **Methoden** (Funktionen), welche die Objekte modifizieren und darin gespeicherte Information zurückgeben können.
- einige Methoden der Klasse `list`
 - ▶ `liste.append(e)` fügt das Element `e` zur Liste `liste` hinzu.
 - ▶ `liste.extend(l)` fügt alle Elemente aus der Liste `l` zur Liste `liste` hinzu.
 - ▶ `liste.pop(n)` entfernt das `n`-te Element aus `liste`.
 - ▶ `liste.insert(n, e)` fügt das Element `e` an Position `n` hinzu.

Methoden des Datentyps list

```
>>> []
```

Methoden des Datentyps list

```
>>> []
```

```
[]
```

```
>>> liste = list()
```

```
>>> liste
```

Methoden des Datentyps list

```
>>> []
```

```
[]
```

```
>>> liste = list()
```

```
>>> liste
```

```
[]
```

```
>>> dir(list)
```

Methoden des Datentyps list

```
>>> []
```

```
[]
```

```
>>> liste = list()
```

```
>>> liste
```

```
[]
```

```
>>> dir(list)
```

```
[..., 'append', 'clear', ...]
```

```
>>> help(list.append)
```

Methoden des Datentyps list

```
>>> []  
[]  
>>> liste = list()  
>>> liste  
[]  
>>> dir(list)  
[..., 'append', 'clear', ...]  
>>> help(list.append)  
Help on method_descriptor:  
append(self, object, /) ...  
>>> liste.append(1)  
>>> liste
```


Methoden des Datentyps list

```
>>> []  
[]  
>>> liste = list()  
>>> liste  
[]  
>>> dir(list)  
[..., 'append', 'clear', ...]  
>>> help(list.append)  
Help on method_descriptor:  
append(self, object, /) ...  
>>> liste.append(1)  
>>> liste  
[1]  
>>> liste.extend([2,3,4])  
>>> liste
```

Methoden des Datentyps list

```
>>> []                                >>> liste.pop()
[]

>>> liste = list()
>>> liste
[]

>>> dir(list)
[..., 'append', 'clear', ...]

>>> help(list.append)
Help on method_descriptor:
append(self, object, /) ...

>>> liste.append(1)
>>> liste
[1]

>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
```

Methoden des Datentyps list

```
>>> []                                >>> liste.pop()
[]                                     4
>>> liste = list()                   >>> liste
>>> liste
[]
>>> dir(list)
[..., 'append', 'clear', ...]
>>> help(list.append)
Help on method_descriptor:
append(self, object, /) ...
>>> liste.append(1)
>>> liste
[1]
>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
```

Methoden des Datentyps list

```
>>> []
[]
>>> liste = list()
>>> liste
[]
>>> dir(list)
[..., 'append', 'clear', ...]
>>> help(list.append)
Help on method_descriptor:
append(self, object, /) ...
>>> liste.append(1)
>>> liste
[1]
>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
```

```
>>> liste.pop()
4
>>> liste
[1, 2, 3]
>>> liste.pop(0)
```

Methoden des Datentyps list

```
>>> []                                >>> liste.pop()
[]                                     4
>>> liste = list()                   >>> liste
>>> liste                             [1, 2, 3]
[]                                    >>> liste.pop(0)
>>> dir(list)                         1
[... , 'append', 'clear', ...] >>> liste
>>> help(list.append)
Help on method_descriptor:
append(self, object, /) ...
>>> liste.append(1)
>>> liste
[1]
>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
```

Methoden des Datentyps list

```
>>> []                                >>> liste.pop()
[]                                     4
>>> liste = list()                  >>> liste
>>> liste                           [1, 2, 3]
[]                                   >>> liste.pop(0)
>>> dir(list)                       1
[..., 'append', 'clear', ...]      >>> liste
>>> help(list.append)               [2, 3]
Help on method_descriptor:
append(self, object, /) ...        >>> liste.insert(1, 'x')
>>> liste.append(1)                >>> liste
>>> liste
[1]
>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
```

Methoden des Datentyps list

```
>>> []
[]
>>> liste = list()
>>> liste
[]
>>> dir(list)
1
[..., 'append', 'clear', ...]
>>> help(list.append)
Help on method_descriptor:
append(self, object, /) ...
>>> liste.append(1)
>>> liste
[1]
>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
>>> liste.pop()
4
>>> liste
[1, 2, 3]
>>> liste.pop(0)
1
>>> liste
[2, 3]
>>> liste.insert(1,'x')
>>> liste
[2, 'x', 3]
>>> liste.append([6,7])
>>> liste
```

Methoden des Datentyps list

```
>>> []
[]
>>> liste = list()
>>> liste
[]
>>> dir(list)
1
[..., 'append', 'clear', ...]
>>> help(list.append)
Help on method_descriptor:
append(self, object, /) ...
>>> liste.append(1)
>>> liste
[1]
>>> liste.extend([2,3,4])
>>> liste
[1, 2, 3, 4]
>>> liste.pop()
4
>>> liste
[1, 2, 3]
>>> liste.pop(0)
1
>>> liste
[2, 3]
>>> liste.insert(1,'x')
>>> liste
[2, 'x', 3]
>>> liste.append([6,7])
>>> liste
[2, 3, 'x', 4, 5, [6, 7]]
>>> liste[5][1]
```


Methoden des Datentyps list

```
>>> []                                >>> liste.pop()
[]                                     4
>>> liste = list()                  >>> liste
>>> liste                            [1, 2, 3]
[]                                   >>> liste.pop(0)
>>> dir(list)                        1
[..., 'append', 'clear', ...] >>> liste
>>> help(list.append)                [2, 3]
Help on method_descriptor:
append(self, object, /) ...
>>> liste.append(1)                  >>> liste.insert(1, 'x')
>>> liste                            >>> liste
[1]                                   [2, 'x', 3]
>>> liste.extend([2,3,4])            >>> liste.append([6,7])
>>> liste                            >>> liste
[1, 2, 3, 4]                         [2, 3, 'x', 4, 5, [6, 7]]
>>> liste[5][1]                      >>> liste[5][1]
7
```

`dir(list)` listet die Methoden der Klasse `list` auf.

Methoden, deren Name mit `__` beginnt, sollten extern nicht verwendet werden.

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]  
>>> liste.count(1)
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
```

```
>>> liste.count(1)
```

```
2
```

```
>>> liste.index(1)
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
```

```
>>> liste.count(1)
```

```
2
```

```
>>> liste.index(1)
```

```
1
```

```
>>> liste.remove(1)
```

```
>>> liste
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
```

```
>>> liste.count(1)
```

```
2
```

```
>>> liste.index(1)
```

```
1
```

```
>>> liste.remove(1)
```

```
>>> liste
```

```
[3, 9, 7, 1, 4]
```

```
>>> liste.sort()
```

```
>>> liste
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]      >>> liste
>>> liste.count(1)
2
>>> liste.index(1)
1
>>> liste.remove(1)
>>> liste
[3, 9, 7, 1, 4]
>>> liste.sort()
>>> liste
[1, 3, 4, 7, 9]
>>> liste.reverse()
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
>>> liste.count(1)
2
>>> liste.index(1)
1
>>> liste.remove(1)
>>> liste
[3, 9, 7, 1, 4]
>>> liste.sort()
>>> liste
[1, 3, 4, 7, 9]
>>> liste.reverse()
```

```
>>> liste
[9, 7, 4, 3, 1]
>>> reversed(liste)
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
>>> liste.count(1)
2
>>> liste.index(1)
1
>>> liste.remove(1)
>>> liste
[3, 9, 7, 1, 4]
>>> liste.sort()
>>> liste
[1, 3, 4, 7, 9]
>>> liste.reverse()

>>> liste
[9, 7, 4, 3, 1]
>>> reversed(liste)
<list_reverseiterator object at ...>
>>> list(reversed(liste))
```


Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
```

```
>>> liste.count(1)
```

```
2
```

```
>>> liste.index(1)
```

```
1
```

```
>>> liste.remove(1)
```

```
>>> liste
```

```
[3, 9, 7, 1, 4]
```

```
>>> liste.sort()
```

```
>>> liste
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste.reverse()
```

```
>>> liste
```

```
[9, 7, 4, 3, 1]
```

```
>>> reversed(liste)
```

```
<list_reverseiterator object at ...>
```

```
>>> list(reversed(liste))
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
```

```
>>> liste.count(1)
```

```
2
```

```
>>> liste.index(1)
```

```
1
```

```
>>> liste.remove(1)
```

```
>>> liste
```

```
[3, 9, 7, 1, 4]
```

```
>>> liste.sort()
```

```
>>> liste
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste.reverse()
```

```
>>> liste
```

```
[9, 7, 4, 3, 1]
```

```
>>> reversed(liste)
```

```
<list_reverseiterator object at ...>
```

```
>>> list(reversed(liste))
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste
```

```
[9, 7, 4, 3, 1]
```

```
>>> sorted(liste)
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
```

```
>>> liste.count(1)
```

```
2
```

```
>>> liste.index(1)
```

```
1
```

```
>>> liste.remove(1)
```

```
>>> liste
```

```
[3, 9, 7, 1, 4]
```

```
>>> liste.sort()
```

```
>>> liste
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste.reverse()
```

```
>>> liste
```

```
[9, 7, 4, 3, 1]
```

```
>>> reversed(liste)
```

```
<list_reverseiterator object at ...>
```

```
>>> list(reversed(liste))
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste
```

```
[9, 7, 4, 3, 1]
```

```
>>> sorted(liste)
```

```
[1, 3, 4, 7, 9]
```

```
>>> liste.copy()
```

Weitere Methoden der Objektklasse list

```
>>> liste = [3,1,9,7,1,4]
>>> liste.count(1)
2
>>> liste.index(1)
1
>>> liste.remove(1)
>>> liste
[3, 9, 7, 1, 4]
>>> liste.sort()
>>> liste
[1, 3, 4, 7, 9]
>>> liste.reverse()
>>> liste
[9, 7, 4, 3, 1]
>>> reversed(liste)
<list_reverseiterator object at ...>
>>> list(reversed(liste))
[1, 3, 4, 7, 9]
>>> liste
[9, 7, 4, 3, 1]
>>> sorted(liste)
[1, 3, 4, 7, 9]
>>> liste.copy()
[9, 7, 4, 3, 1]
```

`count(e)` zählt die Vorkommen des Elementes `e`

`index(e)` liefert die Position des ersten Vorkommens von `e`

`remove(e)` löscht das erste Vorkommen von `e`

`sort()` sortiert die Liste in-place

`reverse()` kehrt die Reihenfolge der Liste um (in-place)

`copy()` liefert eine flache Kopie der Liste

`clear()` entfernt alle Elemente der Liste (existiert auch bei set, dict etc.)

str

```
>>> ''
```

str

```
>>> ''
```

```
''
```

```
>>> str()
```

str

```
>>> ''
```

```
''
```

```
>>> str()
```

```
''
```

```
>>> string = 'Hallo'
```

str

```
>>> ''
```

```
''
```

```
>>> str()
```

```
''
```

```
>>> string = 'Hallo'
```

```
>>> string[3:5]
```


str

```
>>> ''  
''  
  
>>> str()  
''  
  
>>> string = 'Hallo'  
>>> string[3:5]  
'lo'  
  
>>> string[::2]
```

str

```
>>> ''  
''  
>>> str()  
''  
>>> string = 'Hallo'  
>>> string[3:5]  
'lo'  
>>> string[:2]  
'Hl'  
>>> string[::-1]
```

str

```
>>> ''  
''  
>>> str()  
''  
>>> string = 'Hallo'  
>>> string[3:5]  
'lo'  
>>> string[::2]  
'Hlo'  
>>> string[::-1]  
'ollaH'  
>>> sorted(string)
```

str

```
>>> ''  
''  
>>> str()  
''  
>>> string = 'Hallo'  
>>> string[3:5]  
'lo'  
>>> string[:2]  
'Hl'  
>>> string[::-1]  
'ollaH'  
>>> sorted(string)  
['a', 'H', 'l', 'l', 'o']  
>>> string.upper()
```

str

```
>>> ''  
''  
>>> str()  
''  
>>> string = 'Hallo'  
>>> string[3:5]  
'lo'  
>>> string[:2]  
'Hl'  
>>> string[::-1]  
'ollaH'  
>>> sorted(string)  
['a', 'H', 'l', 'l', 'o']  
>>> string.upper()  
'HALLO'  
>>> string.lower()
```

str

```
>>> ''  
''  
>>> str()  
''  
>>> string = 'Hallo'  
>>> string[3:5]  
'lo'  
>>> string[:2]  
'Hl'  
>>> string[::-1]  
'ollaH'  
>>> sorted(string)  
['a', 'H', 'l', 'l', 'o']  
>>> string.upper()  
'HALLO'  
>>> string.lower()  
'hallo'  
>>> string[2] = 'z'
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment
```

```
>>> 'abcdef'.isalpha()
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment
```

```
>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
```


str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hlo'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment
```

```
>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hlo'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment

>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment

>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
True
>>> string.endswith('llo')
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment

>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
True
>>> string.endswith('llo')
True
>>> string.replace('l', 'L')
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'hallo'
>>> string[2] = 'z'
... does not support item assignment

>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
True
>>> string.endswith('llo')
True
>>> string.replace('l', 'L')
'haLLo'
>>> string.rjust(8)
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'halla'
>>> string[2] = 'z'
... does not support item assignment

>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
True
>>> string.endswith('llo')
True
>>> string.replace('l', 'L')
'haLLo'
>>> string.rjust(8)
'    halla'
>>> string.center(8)
'  halla  '
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'hallo'
>>> string[2] = 'z'
... does not support item assignment

>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
True
>>> string.endswith('llo')
True
>>> string.replace('l', 'L')
'haLlo'
>>> string.rjust(8)
'    hallo'
>>> string.center(8)
'  hallo  '
>>> 'hal' 'lo'
```

str

```
>>> ''
''
>>> str()
''
>>> string = 'Hallo'
>>> string[3:5]
'lo'
>>> string[:2]
'Hl'
>>> string[::-1]
'ollaH'
>>> sorted(string)
['a', 'H', 'l', 'l', 'o']
>>> string.upper()
'HALLO'
>>> string.lower()
'hallo'
>>> string[2] = 'z'
... does not support item assignment
```

```
>>> 'abcdef'.isalpha()
True
>>> '12'.isdigit()
True
>>> '12.3'.isdigit()
False
>>> string.startswith('ha')
True
>>> string.endswith('llo')
True
>>> string.replace('l', 'L')
'haLLo'
>>> string.rjust(8)
'    hallo'
>>> string.center(8)
'  hallo  '
>>> 'hal' 'lo'
'hallo'
```


tuple und set

```
>>> ()
```

tuple und set

```
>>> ()
```

```
()
```

```
>>> [(1), (1,)]
```

tuple und set

```
>>> ()
```

```
()
```

```
>>> [(1), (1,)]
```

```
[1, (1,)]
```

```
>>> tup = (1,2,4,2,1)
```

tuple und set

```
>>> ()
```

```
()
```

```
>>> [(1), (1,)]
```

```
[1, (1,)]
```

```
>>> tup = (1,2,4,2,1)
```

```
>>> tup
```

tuple und set

```
>>> ()
```

```
()
```

```
>>> [(1), (1,)]
```

```
[1, (1,)]
```

```
>>> tup = (1,2,4,2,1)
```

```
>>> tup
```

```
(1, 2, 4, 2, 1)
```

```
>>> tup[3:5]
```

tuple und set

```
>>> ()  
()  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)
```

tuple und set

```
>>> ()  
()  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)
```

tuple und set

```
>>> ()  
()  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)
```


tuple und set

```
>>> ()  
()  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)  
[1, 2, 4, 2, 1]  
>>> tup[2] = 7
```

tuple und set

```
>>> ()
()
>>> [(1), (1,)]
[1, (1,)]
>>> tup = (1,2,4,2,1)
>>> tup
(1, 2, 4, 2, 1)
>>> tup[3:5]
(2, 1)
>>> tup.count(1)
2
>>> tup.index(2)
1
>>> list(tup)
[1, 2, 4, 2, 1]
>>> tup[2] = 7
...does not support item assignment
```

```
>>> s = set('hallo')
>>> s
```

tuple und set

```
>>> ()
()
>>> [(1), (1,)]
[1, (1,)]
>>> tup = (1,2,4,2,1)
>>> tup
(1, 2, 4, 2, 1)
>>> tup[3:5]
(2, 1)
>>> tup.count(1)
2
>>> tup.index(2)
1
>>> list(tup)
[1, 2, 4, 2, 1]
>>> tup[2] = 7
...does not support item assignment
```

```
>>> s = set('hallo')
>>> s
{'h', 'a', 'l', 'o'}
>>> s.add('x')
>>> s
```

tuple und set

```
>>> ()  
(  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)  
[1, 2, 4, 2, 1]  
>>> tup[2] = 7  
...does not support item assignment
```

```
>>> s = set('hallo')  
>>> s  
{'h', 'a', 'l', 'o'}  
>>> s.add('x')  
>>> s  
{'x', 'h', 'a', 'l', 'o'}  
>>> s - {'x'}
```

tuple und set

```
>>> ()  
( )  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)  
[1, 2, 4, 2, 1]  
>>> tup[2] = 7
```

...does not support item assignment

```
>>> s = set('hallo')  
>>> s  
{'h', 'a', 'l', 'o'}  
>>> s.add('x')  
>>> s  
{'x', 'h', 'a', 'l', 'o'}  
>>> s - {'x'}  
{'h', 'a', 'l', 'o'}  
>>> s | {'y', 'z'}
```

tuple und set

```
>>> ()  
(  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)  
[1, 2, 4, 2, 1]  
>>> tup[2] = 7
```

...does not support item assignment

```
>>> s = set('hallo')  
>>> s  
{'h', 'a', 'l', 'o'}  
>>> s.add('x')  
>>> s  
{'x', 'h', 'a', 'l', 'o'}  
>>> s - {'x'}  
{'h', 'a', 'l', 'o'}  
>>> s | {'y', 'z'}  
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}  
>>> s & {'a', 'x', 'z'}
```

tuple und set

```
>>> ()
()
>>> [(1), (1,)]
[1, (1,)]
>>> tup = (1,2,4,2,1)
>>> tup
(1, 2, 4, 2, 1)
>>> tup[3:5]
(2, 1)
>>> tup.count(1)
2
>>> tup.index(2)
1
>>> list(tup)
[1, 2, 4, 2, 1]
>>> tup[2] = 7
...does not support item assignment
```

```
>>> s = set('hallo')
>>> s
{'h', 'a', 'l', 'o'}
>>> s.add('x')
>>> s
{'x', 'h', 'a', 'l', 'o'}
>>> s - {'x'}
{'h', 'a', 'l', 'o'}
>>> s | {'y', 'z'}
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}
>>> s & {'a', 'x', 'z'}
{'x', 'a'}
>>> s.intersection({'a', 'x', 'z'})
```

tuple und set

```
>>> ()
()
>>> [(1), (1,)]
[1, (1,)]
>>> tup = (1,2,4,2,1)
>>> tup
(1, 2, 4, 2, 1)
>>> tup[3:5]
(2, 1)
>>> tup.count(1)
2
>>> tup.index(2)
1
>>> list(tup)
[1, 2, 4, 2, 1]
>>> tup[2] = 7
```

...does not support item assignment

```
>>> s = set('hallo')
>>> s
{'h', 'a', 'l', 'o'}
>>> s.add('x')
>>> s
{'x', 'h', 'a', 'l', 'o'}
>>> s - {'x'}
{'h', 'a', 'l', 'o'}
>>> s | {'y','z'}
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}
>>> s & {'a','x','z'}
{'x', 'a'}
>>> s.intersection({'a','x','z'})
{'x', 'a'}
>>> s.union(['y','z'])
```


tuple und set

```
>>> ()  
(  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)  
[1, 2, 4, 2, 1]  
>>> tup[2] = 7
```

...does not support item assignment

```
>>> s = set('hallo')  
>>> s  
{'h', 'a', 'l', 'o'}  
>>> s.add('x')  
>>> s  
{'x', 'h', 'a', 'l', 'o'}  
>>> s - {'x'}  
{'h', 'a', 'l', 'o'}  
>>> s | {'y', 'z'}  
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}  
>>> s & {'a', 'x', 'z'}  
{'x', 'a'}  
>>> s.intersection({'a', 'x', 'z'})  
{'x', 'a'}  
>>> s.union(['y', 'z'])  
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}  
>>> {'o', 'a'}.issubset('hallo')
```

tuple und set

```
>>> ()  
(  
>>> [(1), (1,)]  
[1, (1,)]  
>>> tup = (1,2,4,2,1)  
>>> tup  
(1, 2, 4, 2, 1)  
>>> tup[3:5]  
(2, 1)  
>>> tup.count(1)  
2  
>>> tup.index(2)  
1  
>>> list(tup)  
[1, 2, 4, 2, 1]  
>>> tup[2] = 7
```

...does not support item assignment

```
>>> s = set('hallo')  
>>> s  
{'h', 'a', 'l', 'o'}  
>>> s.add('x')  
>>> s  
{'x', 'h', 'a', 'l', 'o'}  
>>> s - {'x'}  
{'h', 'a', 'l', 'o'}  
>>> s | {'y', 'z'}  
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}  
>>> s & {'a', 'x', 'z'}  
{'x', 'a'}  
>>> s.intersection({'a', 'x', 'z'})  
{'x', 'a'}  
>>> s.union(['y', 'z'])  
{'x', 'h', 'a', 'l', 'o', 'y', 'z'}  
>>> {'o', 'a'}.issubset('hallo')  
True
```

Der if-Befehl

test.py

```
x = 3
y = 2
if x > y:
    print('größer')
elif x < y:
    print('kleiner')
else:
    print('gleich groß')
```

Der if-Befehl

test.py

```
x = 3
y = 2
if x > y:
    print('größer')
elif x < y:
    print('kleiner')
else:
    print('gleich groß')
```

```
> python test.py
größer
```

- `elif`: ... und `else`: ... sind optional.
- `elif`: ... kann auch mehrmals wiederholt werden.
- Der `if`-Befehl wählt zwischen verschiedenen **Codeblöcken** aus.

Einrückung

In Python ist die Einrückung einer Zeile (indentation) wichtig:

```
if x > y:  
    print('größer')  
    print('-----')
```

Das ist nicht dasselbe wie:

```
if x > y:  
    print('größer')  
print('-----')
```

Im zweiten Beispiel wird `print('-----')` immer ausgeführt!

Der if-Einzeiler

Beispiel:

```
>>> x = 3
>>> y = 3
>>> erg = 'größer' if x > y else 'kleiner gleich'
>>> print(erg)
kleiner gleich
```

Der if-Einzeiler

Beispiel:

```
>>> x = 3
>>> y = 3
>>> erg = 'größer' if x > y else 'kleiner gleich'
>>> print(erg)
kleiner gleich
>>> erg = 'größer' if x > y else ('kleiner' if x < y else 'gleich')
>>> print(erg)
```

Der if-Einzeiler

Beispiel:

```
>>> x = 3
>>> y = 3
>>> erg = 'größer' if x > y else 'kleiner gleich'
>>> print(erg)
kleiner gleich

>>> erg = 'größer' if x > y else ('kleiner' if x < y else 'gleich')
>>> print(erg)
gleich
```

- Der Teil `else: ...` ist hier **obligatorisch**.
- Die Klammer kann in dem if-Ausdruck oben auch weggelassen werden.
- Diese Variante des if-Befehls wählt zwischen verschiedenen **Werten** aus.

Funktionen

```
>>> def sag_hallo():  
...     print('Hallo!')  
...  
>>> sag_hallo()
```

Funktionen

```
>>> def sag_hallo():  
...     print('Hallo!')  
...  
>>> sag_hallo()  
Hallo!  
>>> sag_hallo
```

Funktionen

```
>>> def sag_hallo():  
...     print('Hallo!')  
...  
>>> sag_hallo()  
Hallo!  
  
>>> sag_hallo  
<function sag_hallo at 0x7ea5ab738680>  
  
>>> f = sag_hallo  
>>> f()
```

Funktionen

```
>>> def sag_hallo():
...     print('Hallo!')
...
>>> sag_hallo()
Hallo!

>>> sag_hallo
<function sag_hallo at 0x7ea5ab738680>

>>> f = sag_hallo
>>> f()
Hallo!
```

```
>>> def tu_nichts()
...     pass
...
>>> tu_nichts()
```

Funktionen

```
>>> def sag_hallo():          >>> def tu_nichts()
...     print('Hallo!')      ...     pass
...                           ...
>>> sag_hallo()              ... tu_nichts()
Hallo!
>>> sag_hallo
<function sag_hallo at 0x7ea5ab738680>
>>> f = sag_hallo
>>> f()
Hallo!
```

⇒ Auch Funktionen sind in Python Objekte!

- Auf das Schlüsselwort `def` folgt der Name der Funktion und die Liste der Argumentvariablen in runden Klammern.
- Danach folgt der Code der Funktion.
- Der Befehl `pass` wird als Ersatz für einen syntaktisch notwendigen Befehl verwendet.

Funktionen mit Argumenten und Rückgabewerten

```
>>> def quadrat(x):  
...     return x * x  
...  
>>> quadrat(3)
```

Funktionen mit Argumenten und Rückgabewerten

```
>>> def quadrat(x):  
...     return x * x  
...  
>>> quadrat(3)  
9  
>>> quadrat(-4.0)
```

Funktionen mit Argumenten und Rückgabewerten

```
>>> def quadrat(x):  
...     return x * x  
...  
>>> quadrat(3)  
9  
>>> quadrat(-4.0)  
16.0  
>>> quadrat(3, 5)
```


Funktionen mit Argumenten und Rückgabewerten

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

```
>>> quadrat(3)
```

```
9
```

```
>>> quadrat(-4.0)
```

```
16.0
```

```
>>> quadrat(3, 5)
```

```
... quadrat() takes 1 ...  
argument, but 2 were given
```

```
>>> def addiere(x, y):  
...     return x + y  
...
```

```
>>> addiere(3, 5)
```

Funktionen mit Argumenten und Rückgabewerten

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

```
>>> quadrat(3)
```

```
9
```

```
>>> quadrat(-4.0)
```

```
16.0
```

```
>>> quadrat(3, 5)
```

```
... quadrat() takes 1 ...  
argument, but 2 were given
```

```
>>> def addiere(x, y):  
...     return x + y  
...
```

```
>>> addiere(3, 5)
```

```
8
```

```
>>> addiere('abc', 'def')
```

Funktionen mit Argumenten und Rückgabewerten

```
>>> def quadrat(x):  
...     return x * x  
...  
>>> quadrat(3)  
9  
  
>>> quadrat(-4.0)  
16.0  
  
>>> quadrat(3, 5)  
... quadrat() takes 1 ...  
argument, but 2 were given
```

```
>>> def addiere(x, y):  
...     return x + y  
...  
>>> addiere(3, 5)  
8  
  
>>> addiere('abc', 'def')  
' abcdef'  
  
>>> addiere(3)
```

Funktionen mit Argumenten und Rückgabewerten

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

```
>>> quadrat(3)  
9
```

```
>>> quadrat(-4.0)  
16.0
```

```
>>> quadrat(3, 5)  
... quadrat() takes 1 ...  
argument, but 2 were given
```

```
>>> def addiere(x, y):  
...     return x + y  
...
```

```
>>> addiere(3, 5)  
8
```

```
>>> addiere('abc', 'def')  
' abcdef'
```

```
>>> addiere(3)  
... missing ... argument: 'y'
```

- Die Werte, welche der Funktion beim Aufruf übergeben werden, werden den Variablen in der Liste hinter dem Funktionsnamen zugewiesen.
- Jeder Funktion muss die korrekte Zahl an Argumenten übergeben werden.
- Die übergebenen Werte werden den Argument-Variablen zugewiesen.

Funktionen

- Funktionen enthalten Code, der eine bestimmte Aufgabe erfüllt.
- Jede Funktion erhält eine Liste von **Argumenten** als Eingabe.
- Mit dem **return**-Befehl gibt sie einen Ergebniswert zurück.
- Wenn der return-Befehl fehlt, wird **None** zurückgegeben.
- Funktionen **strukturieren** den Code, indem sie kleinere Codeblöcke vom übrigen Code abtrennen.
- So wird der Code **übersichtlicher**.
- Wenn die Funktion an mehreren Stellen aufgerufen wird, verringert sich die **Redundanz** des Codes.

Lokale und globale Variablen

test.py

```
z = 3
```

```
def add(x, y):  
    z = x + y  
    return z
```

```
result = add(z, z)  
print(result, z)
```

> python test.py

Lokale und globale Variablen

test.py

```
z = 3

def add(x, y):
    z = x + y
    return z

result = add(z, z)
print(result, z)
```

> python test.py

6 3

⇒ x, y, z sind innerhalb der Funktion add **lokale Variablen**.

z ist außerhalb der Funktion add eine **globale Variable** mit anderem Wert.

Lokale Variablen

test.py

```
def plus_z(x):  
    return x + z  
  
z = 3  
result = plus_z(2)  
print(result)
```

> python test.py

Lokale Variablen

test.py

```
def plus_z(x):  
    return x + z  
  
z = 3  
result = plus_z(2)  
print(result)
```

> python test.py

5

⇒ z ist hier auch innerhalb der Funktion eine **globale Variable**.

Regel

Eine Variable ist **lokal**, wenn ihr innerhalb der Funktion ein Wert zugewiesen wird, sonst **global**.

Lokale Variablen

test.py

```
def add(x, y):  
    global z # neu  
    z = x + y  
    return z  
  
z = 3  
result = add(z, z)  
print(result, z)
```

> python test.py

6 6

⇒ x, y sind immer noch lokale Variablen.

z ist jetzt auch in der Funktion eine **globale Variable**.

Globale Variablen sollten vermieden werden. Sie machen den Code unübersichtlicher.

Kommentare

Kommentare beginnt mit dem Zeichen `#` und enden am Zeilenende

```
def addiere(x, y):  
    ''' addiert zwei Zahlen.  
  
    Args:  
        x (float): erster Summand  
        y (float): zweiter Summand  
  
    Returns:  
        float: Summe der beiden Zahlen  
    '''  
    return x + y # Hier erfolgt die Ausgabe
```

Die dreifachen Anführungszeichen `'''` umschließen einen mehrzeiligen String. Wenn dieser direkt am Anfang einer Funktion oder Klasse steht, handelt es sich – wie im Beispiel – um einen **Docstring**. Python merkt sich den DocString, damit Entwicklungsumgebungen (auch VS Code) darauf zugreifen können.

Die while-Schleife

```
i = 5
while i < 10:
    print(i)
    i = i + 1
```

> python test.py

5
6
7
8
9

Die while-Schleife wiederholt den Code, solange die Bedingung erfüllt ist.

⇒ Gefahr einer **Endlos-Schleife**!

Die while-Schleife

fakultät.py

```
def fakultät(x):  
    f = 1  
    i = 2  
    while i <= x:  
        f = f * i  
        i = i + 1  
    return f  
  
print(fakultät(0), fakultät(1), fakultät(2), fakultät(3))
```

> python test.py

1 1 2 6

```
def binär(zahl):  
    ergebnis = ''  
    while True:  
        zahl, rest = divmod(zahl, 2) # Quotient und Rest  
        ergebnis = str(rest) + ergebnis  
        if zahl == 0:  
            break # bricht die Schleife ab  
    return ergebnis
```

```
def binär(zahl):  
    ergebnis = ''  
    while True:  
        zahl, rest = divmod(zahl, 2) # Quotient und Rest  
        ergebnis = str(rest) + ergebnis  
        if zahl == 0:  
            break # bricht die Schleife ab  
    return ergebnis  
  
while True: # Diese Schleife wird nur durch break beendet.  
    zahl = input('Zahl: ') # Der Benutzer gibt Zahl ein.  
    if zahl == 'quit':  
        break  
    if not zahl.isdigit():  
        continue # direkt zur nächsten Iteration  
    print(binär(int(zahl))) # geschachtelte Funktionen
```

```
def binär(zahl):  
    ergebnis = ''  
    while True:  
        zahl, rest = divmod(zahl, 2) # Quotient und Rest  
        ergebnis = str(rest) + ergebnis  
        if zahl == 0:  
            break # bricht die Schleife ab  
    return ergebnis  
  
while True: # Diese Schleife wird nur durch break beendet.  
    zahl = input('Zahl: ') # Der Benutzer gibt Zahl ein.  
    if zahl == 'quit':  
        break  
    if not zahl.isdigit():  
        continue # direkt zur nächsten Iteration  
    print(binär(int(zahl))) # geschachtelte Funktionen
```

<code>divmod(n, b)</code>	liefert den Quotienten und den Rest der Division von n durch b.
<code>break</code>	bricht die (innerste) Schleife ab.
<code>continue</code>	springt direkt zum nächsten Schleifendurchlauf
<code>input(prompt)</code>	gibt den Prompt aus und liest eine Benutzereingabe ein

Test der Umwandlung ins Binärformat

```
> python test.py
```

```
Zahl: 13
```

```
1101
```

```
Zahl: 1
```

```
1
```

```
Zahl: 0
```

```
0
```

```
Zahl: 9
```

```
1001
```

```
Zahl: 7
```

```
111
```

```
Zahl:
```

```
Zahl: hs
```

```
Zahl: 4.5
```

```
Zahl: quit
```

Die for-Schleife

summe.py

```
def summe(liste):  
    sum = 0  
    for zahl in liste:  
        sum += zahl # kurz für sum = sum + zahl  
    return sum
```

Die for-Schleife

summe.py

```
def summe(liste):  
    sum = 0  
    for zahl in liste:  
        sum += zahl # kurz für sum = sum + zahl  
    return sum  
  
s = summe([1,7,2,9])  
s2 = sum([1,7,2,9]) # vordefinierte Python-Funktion  
print(s, s2)
```

Die for-Schleife

summe.py

```
def summe(liste):  
    sum = 0  
    for zahl in liste:  
        sum += zahl # kurz für sum = sum + zahl  
    return sum  
  
s = summe([1,7,2,9])  
s2 = sum([1,7,2,9]) # vordefinierte Python-Funktion  
print(s, s2)
```

> python summe.py

19 19

Die for-Schleife iteriert über `liste` und weist nacheinander jedes Listenelement der Schleifen-Variablen `zahl` zu.

Die range-Funktion

fakultät.py

```
def fakultät(x):  
    f = 1  
    for i in range(2, x+1):  
        f = f * i  
    return f  
  
for n in range(4, 10, 2):  
    erg = fakultät(n)  
    # formatierte Stringausgabe  
    print(f'n: {n:2}\tfakultät(n): {erg:6}')
```

> python fakultät.py

```
n:  4    fakultät(n):      24  
n:  6    fakultät(n):     720  
n:  8    fakultät(n):    40320
```

range liefert ein Objekt vom Typ **range** (ähnlich einer Liste).

Formatierte Strings

```
>>> vorname = 'Ben'  
>>> nachname = 'Schur'  
>>> f'> {vorname} {nachname}'
```

Formatierte Strings

```
>>> vorname = 'Ben'  
>>> nachname = 'Schur'  
>>> f'> {vorname} {nachname}'  
> Ben Schur'  
>>> f'Pi: {3.14159:.2f}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
```


Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'

>>> f'Rate: {0.2375:.1%}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'

>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'

>>> f'{vorname:8}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'

>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'

>>> f'{vorname:8}'
'Ben  '

>>> f'{vorname:<8}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'

>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'

>>> f'{vorname:8}'
'Ben  '

>>> f'{vorname:<8}'
'  Schur'

>>> f'{vorname:^8}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'

>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'

>>> f'{vorname:8}'
'Ben  '

>>> f'{vorname:<8}'
'  Schur'

>>> f'{vorname:^8}'
' Ben '
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'

>>> f'Pi: {3.14159:.2f}'
3.14

>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'

>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'

>>> f'{vorname:8}'
'Ben  '

>>> f'{vorname:<8}'
'  Schur'

>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
```

```
'3 + 5 = 8'
```

```
>>> f'Einwohner: {14628913:,}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},{}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
```


Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:','}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
... Vorname: {vorname}
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben  '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},{}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
... Vorname: {vorname}
... Nachname: {nachname}
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},{}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
... Vorname: {vorname}
... Nachname: {nachname}
... '''
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:','}'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
... Vorname: {vorname}
... Nachname: {nachname}
... '''
'\nVorname: Ben\nNachname: Schur\n'
>>> f'{{{vorname}}} {{{{vorname}}}}'
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
... Vorname: {vorname}
... Nachname: {nachname}
... '''
'\nVorname: Ben\nNachname: Schur\n'
>>> f'{{{vorname}}} {{{{vorname}}}}'
'{vorname} {Ben}'
>>> f"{vorname}'s"
```

Formatierte Strings

```
>>> vorname = 'Ben'
>>> nachname = 'Schur'
>>> f'> {vorname} {nachname}'
'> Ben Schur'
>>> f'Pi: {3.14159:.2f}'
3.14
>>> f'5 / 3 = {5/3:.2f}'
'5 / 3 = 1.67'
>>> f'Rate: {0.2375:.1%}'
'Rate: 23.8%'
>>> f'{vorname:8}'
'Ben  '
>>> f'{vorname:<8}'
'  Schur'
>>> f'{vorname:^8}'
' Ben '
```

```
>>> f'3 + 5 = {3 + 5}'
'3 + 5 = 8'
>>> f'Einwohner: {14628913:},{'
'Einwohner: 14,628,913'
>>> f'{vorname=}'
"vorname='Ben'"
>>> f'{32:04}'
'0032'
>>> f'''
... Vorname: {vorname}
... Nachname: {nachname}
... '''
'\nVorname: Ben\nNachname: Schur\n'
>>> f'{{{vorname}}} {{{{vorname}}}}'
'{vorname} {Ben}'
>>> f"{vorname}'s"
"Ben's"
```

Escape-Sequenzen

Für nicht-druckbare Symbole wie den Tabulator gibt es spezielle Zeichenfolgen (Escape-Sequenzen), um sie in einem Python-String zu repräsentieren:

<code>\t</code>	Tabulator
<code>\n</code>	Newline
<code>\r</code>	Wagenrücklauf (Carriage Return)
<code>\b</code>	Backspace (eine Position zurückgehen)
<code>\a</code>	Alarm (akustisches Signal ausgeben)
<code>\"</code>	"
<code>\'</code>	'
<code>\\</code>	\

Iterables

Datentypen, über die iteriert werden kann:

- list `[1, 2, 3]` `[1]`
- range `range(10)` `range(2, 8)` `range(0, 10, 2)`
- str `'abcde'`
- tuple `(1, 2, 3)` `(1,)`
- set `{'a', 'c', 'f'}`
- (dict, enumerate, zip, Generatoren, Dateien)

Rekursion

Die Fakultäts-Funktion kann auch rekursiv definiert werden:

$$0! = 1$$

$$n! = n \cdot (n-1)! \text{ für } n > 0$$

rekursive Funktion

```
def fakultät(n):  
    return 1 if n == 0 else n * fakultät(n-1)  
  
print(fakultät(3))
```

6

⇒ Funktionen können sich selbst aufrufen.

⇒ Ohne eine Abbruchbedingung entsteht eine Endlos-Rekursion!

Vergleiche und Boole'sche Werte

- Vergleiche wie $x > 3$ liefern einen **Wahrheitswert** (Boole'schen Wert)
- Es gibt nur die Wahrheitswerte **True** und **False**.
- In arithmetischen Kontexten werden True und False als 1 und 0 interpretiert.
- In logischen Kontexten werden folgende Werte als False interpretiert:
0, 0.0, '', [], (), set(), {}, None, False
- Alle anderen Werte werden als True interpretiert:
-1, 0.01, ' ', [[]], (''), {None}, True

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```


Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 12 is 13
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

```
>>> l2 is l3
```

```
True
```

```
>>> l1 is l3
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

```
>>> l2 is l3
```

```
True
```

```
>>> l1 is l3
```

```
False
```

```
>>> True and 2
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

```
>>> l2 is l3
```

```
True
```

```
>>> l1 is l3
```

```
False
```

```
>>> True and 2
```

```
2
```

```
>>> False and 2
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

```
>>> l2 is l3
```

```
True
```

```
>>> l1 is l3
```

```
False
```

```
>>> True and 2
```

```
2
```

```
>>> False and 2
```

```
False
```

```
>>> True or 2
```


Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

```
>>> l2 is l3
```

```
True
```

```
>>> l1 is l3
```

```
False
```

```
>>> True and 2
```

```
2
```

```
>>> False and 2
```

```
False
```

```
>>> True or 2
```

```
True
```

```
>>> False or 2
```

Vergleiche und Boole'sche Werte

```
>>> 1 < 2
```

```
True
```

```
>>> 1 < 2 < 3
```

```
True
```

```
>>> w = 1 < 2
```

```
>>> w
```

```
True
```

```
>>> w + 1
```

```
2
```

```
>>> l1 = [1,2]
```

```
>>> l2 = [1,2]
```

```
>>> l3 = l2
```

```
>>> l1 == l3
```

```
True
```

```
>>> l2 is l3
```

```
True
```

```
>>> l1 is l3
```

```
False
```

```
>>> True and 2
```

```
2
```

```
>>> False and 2
```

```
False
```

```
>>> True or 2
```

```
True
```

```
>>> False or 2
```

```
2
```

`x is y` liefert True, falls x und y auf dasselbe Objekt zeigen, sonst False.

`x or y` liefert x falls x wahr ist, sonst y.

`x and y` liefert x falls x falsch ist, sonst y.

Liste

```
liste[i] = 'Augustiner'
```

- i ist eine ganze Zahl.
- liste wird als ein **Array** gespeichert.

Dictionary

```
dictionary[i] = 'Augustiner'
```

- i ist ein beliebiger Datentyp.
- aber: i muss **immutable** (nicht veränderbar) sein!
- dictionary wird als eine **Hash-Tabelle** gespeichert.

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

`dictionary[key] = value` `häufigkeit['a'] = 12`

```
>>> [dict(), {}]
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```


Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':4}
```

```
>>> del d['a'], d['b']
```

```
>>> d
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d[(1,2)] = {'VfB':'top'}
```

```
>>> d
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':4}
```

```
>>> del d['a'], d['b']
```

```
>>> d
```

```
{'c':4}
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':4}
```

```
>>> del d['a'], d['b']
```

```
>>> d
```

```
{'c':4}
```

```
>>> d[(1,2)] = {'VfB':'top'}
```

```
>>> d
```

```
{'c': 4, (1, 2): {'VfB': 'top'}}
```

```
>>> d['a'] += 1
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
{}, {}
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':4}
```

```
>>> del d['a'], d['b']
```

```
>>> d
```

```
{'c':4}
```

```
>>> d[(1,2)] = {'VfB':'top'}
```

```
>>> d
```

```
{'c': 4, (1, 2): {'VfB': 'top'}}
```

```
>>> d['a'] += 1
```

```
...KeyError: 'a'
```

```
>>> d[{1,2}] = 1
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
{}, {}
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':4}
```

```
>>> del d['a'], d['b']
```

```
>>> d
```

```
{'c':4}
```

```
>>> d[(1,2)] = {'VfB':'top'}
```

```
>>> d
```

```
{'c': 4, (1, 2): {'VfB': 'top'}}
```

```
>>> d['a'] += 1
```

```
...KeyError: 'a'
```

```
>>> d[{1,2}] = 1
```

```
...unhashable type: 'set'
```

```
>>> d = {}
```

```
>>> d[frozenset({1,2})] = 1
```

Dictionary

Dictionaries bilden Schlüssel (**Keys**) auf Werte (**Values**) ab:

dictionary[key] = value häufigkeit['a'] = 12

```
>>> [dict(), {}]
```

```
[{}, {}]
```

```
>>> d = {'a':1, 'b':2}
```

```
>>> d
```

```
{'a': 1, 'b': 2}
```

```
>>> d['c'] = 3
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':3}
```

```
>>> d['c'] += 1
```

```
>>> d
```

```
{'a': 1, 'b': 2, 'c':4}
```

```
>>> del d['a'], d['b']
```

```
>>> d
```

```
{'c':4}
```

```
>>> d[(1,2)] = {'VfB':'top'}
```

```
>>> d
```

```
{'c': 4, (1, 2): {'VfB': 'top'}}
```

```
>>> d['a'] += 1
```

```
...KeyError: 'a'
```

```
>>> d[{1,2}] = 1
```

```
...unhashable type: 'set'
```

```
>>> d = {}
```

```
>>> d[frozenset({1,2})] = 1
```

```
{frozenset({1, 2}): 1}
```

```
>>> pers_info = {
```

```
...     'vorname': 'Ben',
```

```
...     'nachname': 'Schur',
```

```
...     'Geburtstag': '1.1.2000',
```

```
... }
```

Häufigkeiten mit einem Dictionary berechnen

charcounts.py

```
def character_counts(string):  
    f = {}  
    for char in string: # prüft ob char in string enthalten ist  
        f[char] = f[char] + 1 if char in f else 1  
    return f
```

Häufigkeiten mit einem Dictionary berechnen

charcounts.py

```
def character_counts(string):  
    f = {}  
    for char in string: # prüft ob char in string enthalten ist  
        f[char] = f[char] + 1 if char in f else 1  
    return f
```

statt: `f[char] + 1 if char in f else 1`

geht auch: `f.get(char, 0) + 1`

Module

```
import charcounts

while True:
    string = input('Bitte einen String eingeben: ')
    d = charcounts.character_counts(string)
    if string == 'quit':
        break
    if string == '':
        continue
    for char in d.keys():
        print(char, d[char])
```

`import datei` importiert alle Funktionen und globalen Variablen aus `datei.py`.
`d.keys()` liefert die Folge der Dictionary-Keys

Module

statt

```
import charcounts
...
d = charcounts.character_counts(string)
```

geht auch

```
from charcounts import character_counts
...
d = character_counts(string)
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')  
>>> letters
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')  
>>> letters  
{'s', 'k', 'g'}  
>>> enumerate(letters)
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')
>>> letters
{'s', 'k', 'g'}
>>> enumerate(letters)
<enumerate object ...>
>>> list(enumerate(letters))
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')
>>> letters
{'s', 'k', 'g'}
>>> enumerate(letters)
<enumerate object ...>
>>> list(enumerate(letters))
[(0, 's'), (1, 'k'), (2, 'g')]
>>> letterindex = {}
>>> for pair in enumerate(letters):
...     i, letter = pair
...     letterindex[letter] = i
...
>>> letterindex
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')           >>> letterindex.keys()
>>> letters
{'s', 'k', 'g'}

>>> enumerate(letters)
<enumerate object ...>

>>> list(enumerate(letters))
[(0, 's'), (1, 'k'), (2, 'g')]

>>> letterindex = {}
>>> for pair in enumerate(letters):
...     i, letter = pair
...     letterindex[letter] = i
...
>>> letterindex
{'s': 0, 'k': 1, 'g': 2}
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')
>>> letters
{'s', 'k', 'g'}

>>> enumerate(letters)
<enumerate object ...>

>>> list(enumerate(letters))
[(0, 's'), (1, 'k'), (2, 'g')]

>>> letterindex = {}
>>> for pair in enumerate(letters):
...     i, letter = pair
...     letterindex[letter] = i
...
>>> letterindex
{'s': 0, 'k': 1, 'g': 2}

>>> letterindex.keys()
dict_keys(['s', 'k', 'g'])
>>> list(letterindex.keys())
```


enumerate, keys, values und items

```
>>> letters = set('gkskg')
>>> letters
{'s', 'k', 'g'}
>>> enumerate(letters)
<enumerate object ...>
>>> list(enumerate(letters))
[(0, 's'), (1, 'k'), (2, 'g')]
>>> letterindex = {}
>>> for pair in enumerate(letters):
...     i, letter = pair
...     letterindex[letter] = i
...
>>> letterindex
{'s': 0, 'k': 1, 'g': 2}
```

```
>>> letterindex.keys()
dict_keys(['s', 'k', 'g'])
>>> list(letterindex.keys())
['s', 'k', 'g']
>>> list(letterindex.values())
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')
>>> letters
{'s', 'k', 'g'}
>>> enumerate(letters)
<enumerate object ...>
>>> list(enumerate(letters))
[(0, 's'), (1, 'k'), (2, 'g')]
>>> letterindex = {}
>>> for pair in enumerate(letters):
...     i, letter = pair
...     letterindex[letter] = i
...
>>> letterindex
{'s': 0, 'k': 1, 'g': 2}
>>> letterindex.keys()
dict_keys(['s', 'k', 'g'])
>>> list(letterindex.keys())
['s', 'k', 'g']
>>> list(letterindex.values())
[0, 1, 2]
>>> list(letterindex.items())
```

enumerate, keys, values und items

```
>>> letters = set('gkskg')
>>> letters
{'s', 'k', 'g'}
>>> enumerate(letters)
<enumerate object ...>
>>> list(enumerate(letters))
[(0, 's'), (1, 'k'), (2, 'g')]
>>> letterindex = {}
>>> for pair in enumerate(letters):
...     i, letter = pair
...     letterindex[letter] = i
...
>>> letterindex
{'s': 0, 'k': 1, 'g': 2}
```

```
>>> letterindex.keys()
dict_keys(['s', 'k', 'g'])
>>> list(letterindex.keys())
['s', 'k', 'g']
>>> list(letterindex.values())
[0, 1, 2]
>>> list(letterindex.items())
[('s', 0), ('k', 1), ('g', 2)]
```

Bibliotheken

Für Python gibt es umfangreiche Bibliotheken, welche für viele Aufgaben fertige Software-Lösungen bereitstellen.

Das Paket **collections** ist Teil der **Python-Standardbibliothek**, welche in jeder Python-Distribution enthalten ist.

Der Befehl

```
import collections
```

importiert alle Funktionen und Klassen aus collections.

Der Befehl

```
from collections import defaultdict
```

 importiert nur die Klasse defaultdict.

defaultdict und Counter

charcount.py

```
from collections import defaultdict

def character_counts(string):
    count = defaultdict(int)
    for char in string:
        count[char] += 1
    return count
```

noch einfacher mit einem Counter-Objekt:

charcount2.py

```
from collections import Counter

def character_counts(string):
    return Counter(string)
```

Dateien

```
datei = open('input.txt', 'rt') # zum Lesen öffnen
text = datei.read()
datei.close()

datei = open('output.txt', 'w') # zum Schreiben öffnen
datei.write(text)
datei.close()

datei = open('output.txt', 'a') # zum Anfügen öffnen
print(text, end='', file=datei)
datei.close()
```

<code>open</code>	öffnet eine Datei
<code>datei.read()</code>	liest die Datei <code>datei</code> auf einmal.
<code>datei.readline()</code>	liest die nächste Zeile der Datei <code>datei</code> .
<code>datei.readlines()</code>	liest alle Zeilen und gibt eine Liste zurück.
<code>datei.write(text)</code>	schreibt <code>text</code> in die Datei <code>datei</code>
<code>datei.close()</code>	schließt die Datei <code>datei</code>

with-Befehl

word-frequencies.py

```
freq = defaultdict(int)
with open('test.txt', 'r') as file:
    for line in file:
        line = line.strip()
        words = line.split()
        for word in words:
            freq[word] += 1
```

with erzeugt einen Kontextmanager, der die Datei automatisch schließt
strip löscht Whitespace am Beginn und Ende einer Zeile
split() zerlegt einen String mit beliebigem Whitespace als Trennzeichen
split('\t') zerlegt einen String mit Tabulator als Trennzeichen
for line in file iteriert über die Zeilen der Textdatei

Whitespace: Leerzeichen, Tabulator, Wagenrücklauf (carriage return), Zeilenvorschub (line feed)

strip,.rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
```


strip,rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
```

```
'abc'
```

```
>>> '\t abc \n\t'.strip(' \t\r\n')
```

strip,.rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
```

```
'abc'
```

```
>>> '\t abc \n\t'.strip(' \t\r\n')
```

```
'abc'
```

```
>>> '\t abc \n\t'.rstrip(' \t\r\n')
```

strip,.rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
'abc'
>>> '\t abc \n\t'.strip(' \t\r\n')
'abc'
>>> '\t abc \n\t'.rstrip(' \t\r\n')
'\t abc'
>>> '\t abc \n\t'.lstrip(' \t\r\n')
```

strip,.rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
'abc'
>>> '\t abc \n\t'.strip(' \t\r\n')
'abc'
>>> '\t abc \n\t'.rstrip(' \t\r\n')
'\t abc'
>>> '\t abc \n\t'.lstrip(' \t\r\n')
'abc \n\t'
>>> 'abcdefghaa '.strip('ba')
```

strip,.rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
'abc'
>>> '\t abc \n\t'.strip(' \t\r\n')
'abc'
>>> '\t abc \n\t'.rstrip(' \t\r\n')
'\t abc'
>>> '\t abc \n\t'.lstrip(' \t\r\n')
'abc \n\t'
>>> 'abcdefghaa '.strip('ba')
cdefgh
>>> 'abcdefghaa '.strip('ba')
```

strip,.rstrip und lstrip

```
>>> '\t abc \n\t'.strip()
'abc'
>>> '\t abc \n\t'.strip(' \t\r\n')
'abc'
>>> '\t abc \n\t'.rstrip(' \t\r\n')
'\t abc'
>>> '\t abc \n\t'.lstrip(' \t\r\n')
'abc \n\t'
>>> 'abcdefghaa '.strip('ba')
cdefgh
>>> 'abcdefghaa '.strip('ba')
'cdefghaa '
```

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Beispiele

```
open('test.txt', 'r')
```

OK

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Beispiele

<code>open('test.txt', 'r')</code>	OK
<code>open('test.txt', mode='r')</code>	OK

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Beispiele

<code>open('test.txt', 'r')</code>	OK
<code>open('test.txt', mode='r')</code>	OK
<code>open(file='test.txt', mode='r')</code>	OK

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Beispiele

<code>open('test.txt', 'r')</code>	OK
<code>open('test.txt', mode='r')</code>	OK
<code>open(file='test.txt', mode='r')</code>	OK
<code>open('test.txt', encoding='utf-8')</code>	OK

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Beispiele

<code>open('test.txt', 'r')</code>	OK
<code>open('test.txt', mode='r')</code>	OK
<code>open(file='test.txt', mode='r')</code>	OK
<code>open('test.txt', encoding='utf-8')</code>	OK
<code>open('test.txt', 'r', 'utf-8')</code>	'utf-8' wird buffering zugewiesen!

Keyword-Argumente

Der open-Befehl besitzt viele mögliche Argumente:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

Alle Argumente außer file sind optional.

Fehlende Argumente werden durch Defaultwerte ersetzt bspw. mode durch 'r'.

Beispiele

<code>open('test.txt', 'r')</code>	OK
<code>open('test.txt', mode='r')</code>	OK
<code>open(file='test.txt', mode='r')</code>	OK
<code>open('test.txt', encoding='utf-8')</code>	OK
<code>open('test.txt', 'r', 'utf-8')</code>	'utf-8' wird buffering zugewiesen!
<code>open(file='test.txt', 'r')</code>	Fehler

Default-Werte

```
>>> def increment(zahl, incr=1):  
...     return zahl + incr  
...  
>>> increment(3, 1)
```

Default-Werte

```
>>> def increment(zahl, incr=1):  
...     return zahl + incr  
...  
>>> increment(3, 1)  
4  
>>> increment(3, 2)
```

Default-Werte

```
>>> def increment(zahl, incr=1):  
...     return zahl + incr  
...  
>>> increment(3, 1)  
4  
>>> increment(3, 2)  
5  
>>> increment(3)
```


Default-Werte

```
>>> def increment(zahl, incr=1):  
...     return zahl + incr  
...  
>>> increment(3, 1)  
4  
>>> increment(3, 2)  
5  
>>> increment(3)  
4
```

Klassen

- Jedes Objekt ist eine **Instanz** einer **Klasse**.
- Es gibt vordefinierte Klassen wie `list`, `dict` und benutzerdefinierte Klassen.
- Instanzen einer Klasse besitzen **Attribute** (Variablen) und **Methoden** (Funktionen)
- Die Definition einer Klasse beginnt mit dem Schlüsselwort `class`.
- Fast alle Klassen besitzen eine Methode mit dem Namen `__init__`, welche die Instanzen **initialisiert**.
- In der Definition der Methoden taucht an 1. Stelle ein zusätzliches Argument **self** auf.
- `self` ist ein Dictionary, in dem alle Instanz-Attribute gespeichert sind.
- Für `self` wird beim Aufruf der Methode kein Argument übergeben.

Klassen

student.py

```
class Student:
    def __init__(self, mnr, vname, nname, studium):
        self.matrikelnummer = mnr
        self.vorname = vname
        self.nachname = nname
        self.studiengang = studium

    def name(self):
        return self.vorname + ' ' + self.nachname

if __name__ == "__main__":
    # wird nicht ausgeführt, wenn die Datei als Modul importiert wird
    student = Student('123456', 'Ben', 'Schur', 'BSc Computerlinguistik')
    print(student.name())
```

```
> python student.py
```

```
Ben Schur
```

Klassen, welche Funktionen definieren

incrementer.py

```
class Incrementer:
    def __init__(self, incr=1):
        self.increment = incr

    def __call__(self, zahl):
        return zahl + self.increment

increment = Incrementer(2)
print(increment.__call__(10))

increment = Incrementer()
print(increment(10))
```

Klassen, welche Funktionen definieren

incrementer.py

```
class Incrementer:
    def __init__(self, incr=1):
        self.increment = incr

    def __call__(self, zahl):
        return zahl + self.increment

increment = Incrementer(2)
print(increment.__call__(10))

increment = Incrementer()
print(increment(10))
```

> python incrementer.py

12

11

Klassen, welche Funktionen definieren

incrementer.py

```
class Incrementer:
    def __init__(self, incr=1):
        self.increment = incr

    def __call__(self, zahl):
        return zahl + self.increment

increment = Incrementer(2)
print(increment.__call__(10))

increment = Incrementer()
print(increment(10))
```

> python incrementer.py

12

11

`__call__` Methode, welche die Klasseninstanzen zu Funktionen macht.

Stringrepräsentation einer Klassen-Instanz

student.py

```
class Student:
    def __init__(self, vorname, nachname, matrikelnummer, studienfach):
        self.vorname = vorname
        self.nachname = nachname
        self.matrikelnummer = matrikelnummer
        self.studienfach = studienfach

    def __str__(self):
        return f'{self.nachname}, {self.vorname} ({self.matrikelnummer})'

student = Student('Ben', 'Schur', 1234567, 'Computerlinguistik')
print(student)
```

Stringrepräsentation einer Klassen-Instanz

student.py

```
class Student:
    def __init__(self, vorname, nachname, matrikelnummer, studienfach):
        self.vorname = vorname
        self.nachname = nachname
        self.matrikelnummer = matrikelnummer
        self.studienfach = studienfach

    def __str__(self):
        return f'{self.nachname}, {self.vorname} ({self.matrikelnummer})'

student = Student('Ben', 'Schur', 1234567, 'Computerlinguistik')
print(student)
```

```
> python student.py
Schur, Ben (1234567)
```


Stringrepräsentation einer Klassen-Instanz

student.py

```
class Student:
    def __init__(self, vorname, nachname, matrikelnummer, studienfach):
        self.vorname = vorname
        self.nachname = nachname
        self.matrikelnummer = matrikelnummer
        self.studienfach = studienfach

    def __str__(self):
        return f'{self.nachname}, {self.vorname} ({self.matrikelnummer})'

student = Student('Ben', 'Schur', 1234567, 'Computerlinguistik')
print(student)
```

```
> python student.py
Schur, Ben (1234567)
```

`__str__` gibt eine Stringrepräsentation der Instanz zurück.

Klassen-Attribute

Neben Instanz-Attributen gibt es auch **Klassen-Attribute**, deren Wert für alle Instanzen derselbe ist.

student.py

```
class Student:
    nächste_nummer = 100000 # Klassen-Attribut

    def __init__(self, vorname, nachname, studienfach):
        self.vorname = vorname # Instanz-Attribut
        self.nachname = nachname
        self.studienfach = studienfach
        self.matrikelnummer = Student.nächste_nummer
        Student.nächste_nummer += 1
```

print-Befehle

```
>>> print(*[1,2,3])
```

print-Befehle

```
>>> print(*[1,2,3])
```

```
1 2 3
```

```
>>> print('Hello', end=' ')
```

```
>>> print('world')
```

print-Befehle

```
>>> print(*[1,2,3])
```

```
1 2 3
```

```
>>> print('Hello', end=' ')
```

```
>>> print('world')
```

```
Hello world
```

```
>>> print(1,2,3, sep='\t')
```

print-Befehle

```
>>> print(*[1,2,3])  
1 2 3  
  
>>> print('Hello', end=' ')  
>>> print('world')  
Hello world  
  
>>> print(1,2,3, sep='\t')  
1    2    3  
  
>>> print('Hello world', file=file)
```

print-Befehle

```
>>> print(*[1,2,3])
```

```
1 2 3
```

```
>>> print('Hello', end=' ')
```

```
>>> print('world')
```

```
Hello world
```

```
>>> print(1,2,3, sep='\t')
```

```
1    2    3
```

```
>>> print('Hello world', file=file)
```

```
Ausgabe in eine Datei
```

```
>>> print('Hello world', file=file, flush=True)
```

print-Befehle

```
>>> print(*[1,2,3])
```

```
1 2 3
```

```
>>> print('Hello', end=' ')
```

```
>>> print('world')
```

```
Hello world
```

```
>>> print(1,2,3, sep='\t')
```

```
1    2    3
```

```
>>> print('Hello world', file=file)
```

```
Ausgabe in eine Datei
```

```
>>> print('Hello world', file=file, flush=True)
```

```
Ausgabe in eine Datei ohne Pufferung
```

- * Dieser Operator ersetzt eine Liste durch die Folge seiner Elemente.

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:
    print(i, end=' ')

for i in range(10):
    print(i, end=' ')

for i in range(0,10,2):
    print(i, end=' ')

for letter in 'Hallo':
    print(letter, end=' ')

for i, letter in enumerate('Hi'):
    print((i, letter))

for zahl, ziffer in zip([1,2], '123'):
    print((i, ziffer), end=' ')
    print(i, ziffer)
```

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:  
    print(i, end=' ')  
  
for i in range(10):  
    print(i, end=' ')  
  
for i in range(0,10,2):  
    print(i, end=' ')  
  
for letter in 'Hallo':  
    print(letter, end=' ')  
  
for i, letter in enumerate('Hi'):  
    print((i, letter))  
  
for zahl, ziffer in zip([1,2], '123'):  
    print((i, ziffer), end=' ')  
    print(i, ziffer)
```

0 1 2 3 4 5 6 7 8 9

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(10):  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(0,10,2):  
    print(i, end=' ')
```

```
for letter in 'Hallo':  
    print(letter, end=' ')
```

```
for i, letter in enumerate('Hi'):  
    print((i, letter))
```

```
for zahl, ziffer in zip([1,2], '123'):  
    print((i, ziffer), end=' ')  
    print(i, ziffer)
```

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(10):  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(0,10,2):  
    print(i, end=' ')
```

0 2 4 6 8

```
for letter in 'Hallo':  
    print(letter, end=' ')
```

```
for i, letter in enumerate('Hi'):  
    print((i, letter))
```

```
for zahl, ziffer in zip([1,2], '123'):  
    print((i, ziffer), end=' ')  
    print(i, ziffer)
```

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(10):  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(0,10,2):  
    print(i, end=' ')
```

0 2 4 6 8

```
for letter in 'Hallo':  
    print(letter, end=' ')
```

H a l l o

```
for i, letter in enumerate('Hi'):  
    print((i, letter))
```

```
for zahl, ziffer in zip([1,2], '123'):  
    print((i, ziffer), end=' ')  
    print(i, ziffer)
```

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(10):  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(0,10,2):  
    print(i, end=' ')
```

0 2 4 6 8

```
for letter in 'Hallo':  
    print(letter, end=' ')
```

H a l l o

```
for i, letter in enumerate('Hi'):  
    print((i, letter))
```

(0, 'H')
(1, 'i')

```
for zahl, ziffer in zip([1,2], '123'):  
    print((i, ziffer), end=' ')  
    print(i, ziffer)
```

for-Schleifen und zip

student.py

```
for i in [0,1,2,3,4,5,6,7,8,9]:  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(10):  
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

```
for i in range(0,10,2):  
    print(i, end=' ')
```

0 2 4 6 8

```
for letter in 'Hallo':  
    print(letter, end=' ')
```

H a l l o

```
for i, letter in enumerate('Hi'):  
    print((i, letter))
```

(0, 'H')
(1, 'i')

```
for zahl, ziffer in zip([1,2], '123'):  
    print((i, ziffer), end=' ')  
    print(i, ziffer)
```

(1, '1')
(2, '2')

Der del-Befehl

```
>>> a = 10  
>>> del a  
>>> print(a)
```


Der del-Befehl

```
>>> a = 10
>>> del a
>>> print(a)
... name 'a' is not defined

>>> liste = [1,2,3]
>>> del liste[1]
>>> liste
```

Der del-Befehl

```
>>> a = 10
>>> del a
>>> print(a)
... name 'a' is not defined

>>> liste = [1,2,3]
>>> del liste[1]
>>> liste
[1, 3]

>>> d = {'a':1, 'b':2, 'c':3}
>>> del d['b']
>>> d
```

Der del-Befehl

```
>>> a = 10
>>> del a
>>> print(a)
... name 'a' is not defined

>>> liste = [1,2,3]
>>> del liste[1]
>>> liste
[1, 3]

>>> d = {'a':1, 'b':2, 'c':3}
>>> del d['b']
>>> d
{'a': 1, 'c': 3}
```

```
>>> class Cls:
>>>     def __init__(self):
>>>         self.attr = 'wert'
>>> c = Cls()
>>> c.attr
```

Der del-Befehl

```
>>> a = 10
>>> del a
>>> print(a)
... name 'a' is not defined

>>> liste = [1,2,3]
>>> del liste[1]
>>> liste
[1, 3]

>>> d = {'a':1, 'b':2, 'c':3}
>>> del d['b']
>>> d
{'a': 1, 'c': 3}
```

```
>>> class Cls:
>>>     def __init__(self):
>>>         self.attr = 'wert'
>>> c = Cls()
>>> c.attr
'wert'
>>> del c.attr
>>> c.attr
```

Der del-Befehl

```
>>> a = 10
>>> del a
>>> print(a)
... name 'a' is not defined

>>> liste = [1,2,3]
>>> del liste[1]
>>> liste
[1, 3]

>>> d = {'a':1, 'b':2, 'c':3}
>>> del d['b']
>>> d
{'a': 1, 'c': 3}
```

```
>>> class Cls:
>>>     def __init__(self):
>>>         self.attr = 'wert'
>>> c = Cls()
>>> c.attr
'wert'
>>> del c.attr
>>> c.attr
'Cls' object has no attribute 'attr'
```

Sortieren

```
>>> sorted([7,1,5,2,3,5])
```

Sortieren

```
>>> sorted([7,1,5,2,3,5])
```

```
[1, 2, 3, 5, 5, 7]
```

```
>>> sorted([7,1,5,2,3,5], reverse=True)
```

Sortieren

```
>>> sorted([7,1,5,2,3,5])
```

```
[1, 2, 3, 5, 5, 7]
```

```
>>> sorted([7,1,5,2,3,5], reverse=True)
```

```
sorted([7,1,5,2,3,5], reverse=True)
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'])
```


Sortieren

```
>>> sorted([7,1,5,2,3,5])
```

```
[1, 2, 3, 5, 5, 7]
```

```
>>> sorted([7,1,5,2,3,5], reverse=True)
```

```
sorted([7,1,5,2,3,5], reverse=True)
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'])
```

```
['12', 'LMU', 'PAGS', 'ab', 'o', 'x']
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'], key=len)
```

Sortieren

```
>>> sorted([7,1,5,2,3,5])
```

```
[1, 2, 3, 5, 5, 7]
```

```
>>> sorted([7,1,5,2,3,5], reverse=True)
```

```
sorted([7,1,5,2,3,5], reverse=True)
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'])
```

```
['12', 'LMU', 'PAGS', 'ab', 'o', 'x']
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'], key=len)
```

```
['x', 'o', 'ab', '12', 'LMU', 'PAGS']
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', 12, 'PAGS'])
```

Sortieren

```
>>> sorted([7,1,5,2,3,5])
```

```
[1, 2, 3, 5, 5, 7]
```

```
>>> sorted([7,1,5,2,3,5], reverse=True)
```

```
sorted([7,1,5,2,3,5], reverse=True)
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'])
```

```
['12', 'LMU', 'PAGS', 'ab', 'o', 'x']
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', '12', 'PAGS'], key=len)
```

```
['x', 'o', 'ab', '12', 'LMU', 'PAGS']
```

```
>>> sorted(['x', 'ab', 'o', 'LMU', 12, 'PAGS'])
```

```
... '<' not supported between instances of 'int' and 'str'
```

Sortieren mit namenlosen Funktionen (Lambda-Funktionen)

```
>>> d = {'c':75, 'b':21, 'd':33, 'a':113}  
>>> sorted(d.items())
```

Sortieren mit namenlosen Funktionen (Lambda-Funktionen)

```
>>> d = {'c':75, 'b':21, 'd':33, 'a':113}
>>> sorted(d.items())
[('a', 113), ('b', 21), ('c', 75), ('d', 33)]

>>> def get_key(x):
...     return x[1]
...
>>> sorted(d.items(), key=get_key)
```

Sortieren mit namenlosen Funktionen (Lambda-Funktionen)

```
>>> d = {'c':75, 'b':21, 'd':33, 'a':113}
>>> sorted(d.items())
[('a', 113), ('b', 21), ('c', 75), ('d', 33)]

>>> def get_key(x):
...     return x[1]
...
>>> sorted(d.items(), key=get_key)
[('b', 21), ('d', 33), ('c', 75), ('a', 113)]

>>> sorted(d.items(), key=lambda x: -x[1])
```

Sortieren mit namenlosen Funktionen (Lambda-Funktionen)

```
>>> d = {'c':75, 'b':21, 'd':33, 'a':113}
>>> sorted(d.items())
[('a', 113), ('b', 21), ('c', 75), ('d', 33)]

>>> def get_key(x):
...     return x[1]
...
>>> sorted(d.items(), key=get_key)
[('b', 21), ('d', 33), ('c', 75), ('a', 113)]

>>> sorted(d.items(), key=lambda x: -x[1])
[('a', 113), ('c', 75), ('d', 33), ('b', 21)]

>>> max(d.items(), key=lambda x: x[1])
```

Sortieren mit namenlosen Funktionen (Lambda-Funktionen)

```
>>> d = {'c':75, 'b':21, 'd':33, 'a':113}
>>> sorted(d.items())
[('a', 113), ('b', 21), ('c', 75), ('d', 33)]

>>> def get_key(x):
...     return x[1]
...
>>> sorted(d.items(), key=get_key)
[('b', 21), ('d', 33), ('c', 75), ('a', 113)]

>>> sorted(d.items(), key=lambda x: -x[1])
[('a', 113), ('c', 75), ('d', 33), ('b', 21)]

>>> max(d.items(), key=lambda x: x[1])
('a', 113)
```


Generatoren und yield

Einlesen einer tsv-Datei (mit Tabulatoren als Trennzeichen)

```
def tsv_reader(file):  
    return [line.rstrip('\n').split('\t') for line in file]
```

Generatoren und yield

Einlesen einer tsv-Datei (mit Tabulatoren als Trennzeichen)

```
def tsv_reader(file):  
    return [line.rstrip('\n').split('\t') for line in file]
```

Nachteil: Bei großen Dateien wird viel Speicherplatz verbraucht.

Alternative: ein Generator

```
def tsv_reader(file):  
    for line in file:  
        yield line.rstrip('\n').split('\t')
```



```
with open('beispiel.tsv', 'rt', encoding='utf-8') as file:  
    for row in tsv_reader(file):  
        print(*row)
```



```
tsv_reader(file)[10] # Fehler, Generatoren sind nicht indexierbar
```

List Comprehension

Berechnung einer Liste von Quadratzahlen

```
quadrat = []  
for i in range(10):  
    quadrat.append(i*i)
```

List Comprehension

Berechnung einer Liste von Quadratzahlen

```
quadrat = []  
for i in range(10):  
    quadrat.append(i*i)
```

Dieselbe Aufgabe kann eleganter mit einer **List Comprehension** gelöst werden:

```
quadrat = [i*i for i in range(10)]
```

List Comprehension

Berechnung einer Liste von Quadratzahlen

```
quadrat = []  
for i in range(10):  
    quadrat.append(i*i)
```

Dieselbe Aufgabe kann eleganter mit einer **List Comprehension** gelöst werden:

```
quadrat = [i*i for i in range(10)]
```

Auch ein **Generator-Ausdruck** ist möglich:

```
quadrat = (i*i for i in range(10))
```

List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]
```

List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]
```

```
[0.0, 0.693, 1.099, 1.386]
```

```
>>> [chr(x) for x in range(70,80)]
```

List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]
```

```
[0.0, 0.693, 1.099, 1.386]
```

```
>>> [chr(x) for x in range(70,80)]
```

```
['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

```
>>> [[0 for _ in range(3)] for _ in range(2)]
```


List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]
```

```
[0.0, 0.693, 1.099, 1.386]
```

```
>>> [chr(x) for x in range(70,80)]
```

```
['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

```
>>> [[0 for _ in range(3)] for _ in range(2)]
```

```
[[0, 0, 0], [0, 0, 0]]
```

```
>>> liste = [[0] * 3] * 2
```

```
>>> liste
```

List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]
```

```
[0.0, 0.693, 1.099, 1.386]
```

```
>>> [chr(x) for x in range(70,80)]
```

```
['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

```
>>> [[0 for _ in range(3)] for _ in range(2)]
```

```
[[0, 0, 0], [0, 0, 0]]
```

```
>>> liste = [[0] * 3] * 2
```

```
>>> liste
```

```
[[0, 0, 0], [0, 0, 0]]
```

```
>>> liste[0][0] = 1
```

```
>>> liste
```

List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]  
[0.0, 0.693, 1.099, 1.386]  
  
>>> [chr(x) for x in range(70,80)]  
['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']  
  
>>> [[0 for _ in range(3)] for _ in range(2)]  
[[0, 0, 0], [0, 0, 0]]  
  
>>> liste = [[0] * 3] * 2  
>>> liste  
[[0, 0, 0], [0, 0, 0]]  
  
>>> liste[0][0] = 1  
>>> liste  
[[1, 0, 0], [1, 0, 0]]  
  
>>> letters = 'abcdefg'  
>>> {letter:i for i, letter in enumerate(letters)}
```

List und Dictionary Comprehensions

```
>>> [round(math.log(x), 3) for x in range(1,5)]
```

```
[0.0, 0.693, 1.099, 1.386]
```

```
>>> [chr(x) for x in range(70,80)]
```

```
['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

```
>>> [[0 for _ in range(3)] for _ in range(2)]
```

```
[[0, 0, 0], [0, 0, 0]]
```

```
>>> liste = [[0] * 3] * 2
```

```
>>> liste
```

```
[[0, 0, 0], [0, 0, 0]]
```

```
>>> liste[0][0] = 1
```

```
>>> liste
```

```
[[1, 0, 0], [1, 0, 0]]
```

```
>>> letters = 'abcdefg'
```

```
>>> {letter:i for i, letter in enumerate(letters)}
```

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6}
```

Konvention: Der Variablenname `_` wird verwendet, wenn die Syntax eine Variable verlangt, die nicht weiter benötigt wird.

chain und join

```
>>> from itertools import chain  
>>> liste1 = [1, 2, 3]  
>>> liste2 = [4, 5, 6]  
>>> chain(liste1, liste2)
```

chain und join

```
>>> from itertools import chain
>>> liste1 = [1, 2, 3]
>>> liste2 = [4, 5, 6]
>>> chain(liste1, liste2)
<itertools.chain ...>
>>> list(chain(liste1, liste2))
```

chain und join

```
>>> from itertools import chain
>>> liste1 = [1, 2, 3]
>>> liste2 = [4, 5, 6]
>>> chain(liste1, liste2)
<itertools.chain ...>

>>> list(chain(liste1, liste2))
[1, 2, 3, 4, 5, 6]

>>> liste = ['Hello', 'World']
>>> ' '.join(liste)
```

chain und join

```
>>> from itertools import chain
>>> liste1 = [1, 2, 3]
>>> liste2 = [4, 5, 6]
>>> chain(liste1, liste2)
<itertools.chain ...>

>>> list(chain(liste1, liste2))
[1, 2, 3, 4, 5, 6]

>>> liste = ['Hello', 'World']
>>> ' '.join(liste)
'Hello World'

>>> ''.join(liste)
```


chain und join

```
>>> from itertools import chain
>>> liste1 = [1, 2, 3]
>>> liste2 = [4, 5, 6]
>>> chain(liste1, liste2)
<itertools.chain ...>

>>> list(chain(liste1, liste2))
[1, 2, 3, 4, 5, 6]

>>> liste = ['Hello', 'World']
>>> ' '.join(liste)
'Hello World'

>>> ''.join(liste)
'HelloWorld'
```

chain: verkettet Iterables

join: konkateniert Strings mit einem Trennzeichen dazwischen

Shebang und Kommandozeilen-Argumente

arguments.py

```
#!/usr/bin/python3
```

```
import sys
```

```
print(' '.join(sys.argv))
```

```
> chmod +x arguments.py
```

```
> ./arguments.py arg1 arg2
```

Shebang und Kommandozeilen-Argumente

arguments.py

```
#!/usr/bin/python3
```

```
import sys
```

```
print(' '.join(sys.argv))
```

```
> chmod +x arguments.py
```

```
> ./arguments.py arg1 arg2
```

```
arguments.py arg1 arg2
```

Shebang und Kommandozeilen-Argumente

arguments.py

```
#!/usr/bin/python3

import sys

print(' '.join(sys.argv))
```

```
> chmod +x arguments.py
> ./arguments.py arg1 arg2
arguments.py arg1 arg2
```

`sys.argv` Liste mit dem Programmnamen und den Kommandozeilen-Argumenten

Der Shebang `#!/usr/bin/python3` muss in der 1. Zeile stehen. Er sagt dem Betriebssystem, mit welchem Interpreter das Programm ausgeführt werden soll.

Argparse

Optionale Argumente verarbeitet man am besten mit dem Modul Argparse.

myprogram.py

```
import argparse

parser = argparse.ArgumentParser(description='my great program')
# Definition of an obligatory argument (positional argument)
parser.add_argument('file', help='argument file')
# Definition of some optional arguments
parser.add_argument('-n', '--number', type=int, default=1, help='an
    optional numeric argument')
parser.add_argument('-q', '--quiet', action='store_true', default=False,
    help='suppress messages')
args = parser.parse_args()

print(args.file, args.number, args.quiet)
```

```
> python myprogram.py datei.txt
datei.txt 1 False
```

Argparser

```
> python myprogram.py
```

Argparser

```
> python myprogram.py  
usage: myprogram.py [-h] [-n NUMBER] [-q] file  
test.py: error: the following arguments are required: file  
  
> python myprogram.py datei.txt -n 3
```

Argparser

```
> python myprogram.py
usage: myprogram.py [-h] [-n NUMBER] [-q] file
test.py: error: the following arguments are required: file

> python myprogram.py datei.txt -n 3
datei.txt 3 False

> python myprogram.py datei.txt -q
```


Argparser

```
> python myprogram.py
usage: myprogram.py [-h] [-n NUMBER] [-q] file
test.py: error: the following arguments are required: file

> python myprogram.py datei.txt -n 3
datei.txt 3 False

> python myprogram.py datei.txt -q
datei.txt 1 True

> python myprogram.py datei.txt --number=5 -q
```

Argparser

```
> python myprogram.py
usage: myprogram.py [-h] [-n NUMBER] [-q] file
test.py: error: the following arguments are required: file

> python myprogram.py datei.txt -n 3
datei.txt 3 False

> python myprogram.py datei.txt -q
datei.txt 1 True

> python myprogram.py datei.txt --number=5 -q
datei.txt 5 True

> python myprogram.py datei.txt -n=5
```

Argparser

```
> python myprogram.py
usage: myprogram.py [-h] [-n NUMBER] [-q] file
test.py: error: the following arguments are required: file

> python myprogram.py datei.txt -n 3
datei.txt 3 False

> python myprogram.py datei.txt -q
datei.txt 1 True

> python myprogram.py datei.txt --number=5 -q
datei.txt 5 True

> python myprogram.py datei.txt -n=5
datei.txt 5 False
```

stdin, stdout, stderr

Das Modul **sys** stellt 3 spezielle Datei-Objekte bereit, die immer geöffnet sind.

- stdin** zum Einlesen von Tastatureingaben (vgl. `input`)
- stdout** für normale Ausgaben auf den Bildschirm
- stderr** zur Ausgabe von Fehlermeldungen auf den Bildschirm

stdin, stdout, stderr

Das Modul **sys** stellt 3 spezielle Datei-Objekte bereit, die immer geöffnet sind.

- stdin** zum Einlesen von Tastatureingaben (vgl. `input`)
- stdout** für normale Ausgaben auf den Bildschirm
- stderr** zur Ausgabe von Fehlermeldungen auf den Bildschirm

Warum gibt es neben `stdout` auch noch `stderr`?

Wenn die Ausgabe eines Programms mit

```
> python programm.py > datei.txt
```

in eine Datei umgeleitet wird, erscheinen die `stderr`-Ausgaben trotzdem auf dem Bildschirm!

stdin, stdout, stderr

test.py

```
import sys

print('Dein Name:', end=' ', file=sys.stdout, flush=True)
name = sys.stdin.readline().strip()
if name == '':
    sys.stderr.write('\nFehler: Leere Eingabe!\n')
else:
    sys.stdout.write(f'\nHallo {name}!\n')
```

```
> python test.py
```

```
Dein Name: Ben
```

```
Hallo Ben!
```

```
> python test.py
```

```
Dein Name:
```

```
Fehler: Leere Eingabe!
```

```
> echo Ben | python test.py
```

```
Dein Name:
```

```
Hallo Ben!
```

```
> echo Ben | python test.py > aus.txt
```

```
> echo '' | python test.py > /dev/null
```

```
Fehler: Leere Eingabe!
```

fileinput

ermöglicht Eingaben von Tastatur, Pipes oder Argumentdateien.

test.py

```
import fileinput

for line in fileinput.input():
    print(line.strip())
```

> python test.py

Eingabe von der Tastatur

fileinput

ermöglicht Eingaben von Tastatur, Pipes oder Argumentdateien.

test.py

```
import fileinput

for line in fileinput.input():
    print(line.strip())
```

```
> python test.py                                # Eingabe von der Tastatur
Hello!
Hello!

> echo 'Hello!' | python test.py                 # Eingabe von einer Pipe
```


fileinput

ermöglicht Eingaben von Tastatur, Pipes oder Argumentdateien.

test.py

```
import fileinput

for line in fileinput.input():
    print(line.strip())
```

```
> python test.py                                # Eingabe von der Tastatur
Hello!
Hello!

> echo 'Hello!' | python test.py                 # Eingabe von einer Pipe
Hello!

> echo Hello > hello.txt
> echo 'world!' > world.txt
> python test.py hello.txt world.txt            # Eingabe aus einer Datei
```

fileinput

ermöglicht Eingaben von Tastatur, Pipes oder Argumentdateien.

test.py

```
import fileinput

for line in fileinput.input():
    print(line.strip())
```

```
> python test.py                                # Eingabe von der Tastatur
Hello!
Hello!

> echo 'Hello!' | python test.py                 # Eingabe von einer Pipe
Hello!

> echo Hello > hello.txt
> echo 'world!' > world.txt
> python test.py hello.txt world.txt            # Eingabe aus einer Datei
Hello
world!
```

translate, maketrans und replace

In Unix gibt es den Befehl **tr** für Zeichenersetzungen:

```
> echo 'Hallo!' | tr 'aol' 'oak'  
Hokka!
```

Dasselbe geht auch in Python, ist aber etwas umständlicher:

```
>>> tabelle = str.maketrans('aol','oak'))  
>>> 'Hallo!'.translate(tabelle)  
'Hokka!'
```

Mit der `replace`-Methode kann ein Teilstring durch einen anderen Teilstring ersetzt werden:

```
>>> 'Hallo!'.replace('allo','ow')  
'How!'
```

Reguläre Ausdrücke

Mit regulären Ausdrücken kann nicht nur nach einer bestimmten Zeichenfolge wie `schmid@cis.lmu.de` gesucht werden sondern nach **Mustern**:

`[a-zA-Z0-9]+@[A-Za-z0-9]+\.[A-Za-z]+`

<code>[a-zA-Z0-9]+</code>	Benutzername
<code>[A-Za-z]+</code>	Top-Level-Domain am Ende
<code>([A-Za-z0-9]+\.)+</code>	eine oder mehr Unterdomänen

`\.` Der Punkt soll hier nicht für ein beliebiges Zeichen stehen (Quotierung).

Reguläre Ausdrücke

In Python gibt es für reguläre Ausdrücke das Paket **re**:

check.py

```
import sys
import re

alpha = '[a-zA-Z]'
alnum = '[a-zA-Z0-9]'

topdomain = rf'{alpha}+'
domain = rf'({alnum}+\\-)*{alnum}+\\. '
username = rf'({alnum}+\\._)*{alnum}+'
# $ markiert das Stringende
regex = re.compile(rf'{username}@({domain})+{topdomain}$')

emailadresse = sys.argv[1]
if regex.match(emailadresse):
    print('OK')
else:
    print('Fehler')
```

Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis..de
```

Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis..de
```

Fehler

```
> python check.py Helmut.Schmid@CIS.LMU.DE
```


Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis..de
```

Fehler

```
> python check.py Helmut.Schmid@CIS.LMU.DE
```

OK

```
> python check.py schmid@cis.lmu.de
```

Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis..de
```

Fehler

```
> python check.py Helmut.Schmid@CIS.LMU.DE
```

OK

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis-test.lmu.de
```

Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis..de
```

Fehler

```
> python check.py Helmut.Schmid@CIS.LMU.DE
```

OK

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis-test.lmu.de
```

OK

```
> python check.py schmid@cis-.lmu.de
```

Reguläre Ausdrücke – Programmausführung

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis..de
```

Fehler

```
> python check.py Helmut.Schmid@CIS.LMU.DE
```

OK

```
> python check.py schmid@cis.lmu.de
```

OK

```
> python check.py schmid@cis-test.lmu.de
```

OK

```
> python check.py schmid@cis-.lmu.de
```

Fehler

Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
```

Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
<re.Match object; span=(0, 17), match='schmid@cis.lmu.de'>
>>> match.groups()
```

Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
<re.Match object; span=(0, 17), match='schmid@cis.lmu.de'>
>>> match.groups()
('schmid', 'lmu.', 'de')
>>> match.group(1)
```

Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
<re.Match object; span=(0, 17), match='schmid@cis.lmu.de'>
>>> match.groups()
('schmid', 'lmu.', 'de')
>>> match.group(1)
'schmid'
>>> match.group(2,3)
```


Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
<re.Match object; span=(0, 17), match='schmid@cis.lmu.de'>
>>> match.groups()
('schmid', 'lmu.', 'de')
>>> match.group(1)
'schmid'
>>> match.group(2,3)
('lmu.', 'de')
>>> match.group(0)
```

Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
<re.Match object; span=(0, 17), match='schmid@cis.lmu.de'>
>>> match.groups()
('schmid', 'lmu.', 'de')
>>> match.group(1)
'schmid'
>>> match.group(2,3)
('lmu.', 'de')
>>> match.group(0)
'schmid@cis.lmu.de'
>>> match = regex.match('schmid@cis.lmu.de-uk')
>>> match.group(0,1,2,3)
```

Reguläre Ausdrücke – Klammern

```
>>> import re
>>> regex = re.compile(r'([a-z]+)@([a-z]+\.)+([a-z]+)')
>>> match = regex.match('schmid@cis.lmu.de')
>>> match
<re.Match object; span=(0, 17), match='schmid@cis.lmu.de'>
>>> match.groups()
('schmid', 'lmu.', 'de')
>>> match.group(1)
'schmid'
>>> match.group(2,3)
('lmu.', 'de')
>>> match.group(0)
'schmid@cis.lmu.de'
>>> match = regex.match('schmid@cis.lmu.de-uk')
>>> match.group(0,1,2,3)
('schmid@cis.lmu.de', 'schmid', 'lmu.', 'de')
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')  
'ein -Text'
```

```
>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')  
'ein -Text'
```

```
>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')  
'ein Beispiel-Text'
```

```
>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')  
>>> match.group(0, 1)
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
'ein -Text'

>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
'ein Beispiel-Text'

>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')
>>> match.group(0, 1)
('<b>', '<b>')

>>> match.span()
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
'ein -Text'

>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
'ein Beispiel-Text'

>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')
>>> match.group(0, 1)
('<b>', '<b>')

>>> match.span()
(4, 7)

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n')
```


Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
'ein -Text'

>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
'ein Beispiel-Text'

>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')
>>> match.group(0, 1)
('<b>', '<b>')

>>> match.span()
(4, 7)

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n')
['CIS', 'LMU', 'DE']

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n', flags=re.S)
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
'ein -Text'

>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
'ein Beispiel-Text'

>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')
>>> match.group(0, 1)
('<b>', '<b>')

>>> match.span()
(4, 7)

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n')
['CIS', 'LMU', 'DE']

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n', flags=re.S)
['CIS\nLMU\nDE\n']

>>> re.findall(r'^[a-z]', 'CIS\nLMU\nDE\n', flags=re.I)
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
'ein -Text'

>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
'ein Beispiel-Text'

>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')
>>> match.group(0, 1)
('<b>', '<b>')

>>> match.span()
(4, 7)

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n')
['CIS', 'LMU', 'DE']

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n', flags=re.S)
['CIS\nLMU\nDE\n']

>>> re.findall(r'^[a-z]', 'CIS\nLMU\nDE\n', flags=re.I)
['C']

>>> re.findall(r'^[a-z]', 'CIS\nLMU\nDE\n', flags=re.I|re.M)
```

Reguläre Ausdrücke – sub, search, findall

```
>>> re.sub(r'<.*>', '', 'ein <b>Beispiel</b>-Text')
'ein -Text'

>>> re.sub(r'<.*?>', '', 'ein <b>Beispiel</b>-Text')
'ein Beispiel-Text'

>>> match = re.search(r'(<.*?>)', 'ein <b>Beispiel</b>-Text')
>>> match.group(0, 1)
('<b>', '<b>')

>>> match.span()
(4, 7)

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n')
['CIS', 'LMU', 'DE']

>>> re.findall(r'.+', 'CIS\nLMU\nDE\n', flags=re.S)
['CIS\nLMU\nDE\n']

>>> re.findall(r'^[a-z]', 'CIS\nLMU\nDE\n', flags=re.I)
['C']

>>> re.findall(r'^[a-z]', 'CIS\nLMU\nDE\n', flags=re.I|re.M)
['C', 'L', 'D']
```

Reguläre Ausdrücke – spezielle Klammerausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()  
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()  
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()  
('def',)
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
('def',)
```

Lookbehind

```
>>> re.match(r'(.+)(?<=bc)(.+)', 'abcdef').group(0,1,2)
```


Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
```

```
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
```

```
('def',)
```

Lookbehind

```
>>> re.match(r'(.+)(?<=bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abc', 'def')
```

```
>>> re.match(r'(...+?)(?<!bc)(.+)', 'abcdef').group(0,1,2)
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
```

```
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
```

```
('def',)
```

Lookbehind

```
>>> re.match(r'(.+)(?<=bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abc', 'def')
```

```
>>> re.match(r'(...+?)(?<!bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abcd', 'ef')
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
('abc', 'def')
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
('def',)
```

Lookbehind

```
>>> re.match(r'(.+)(?<=bc)(.+)', 'abcdef').group(0,1,2)
('abcdef', 'abc', 'def')
>>> re.match(r'(...+?)(?<!bc)(.+)', 'abcdef').group(0,1,2)
('abcdef', 'abcd', 'ef')
```

Lookahead

```
>>> re.match(r'(.+?)(?=de)(.+)', 'abcdef').group(0,1,2)
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
```

```
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
```

```
('def',)
```

Lookbehind

```
>>> re.match(r'(.+)(?<=bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abc', 'def')
```

```
>>> re.match(r'(...+?)(?<!bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abcd', 'ef')
```

Lookahead

```
>>> re.match(r'(.+?)(?=de)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abc', 'def')
```

```
>>> re.match(r'(.+?)(?!de)(...+)', 'abcdef').group(0,1,2)
```

Reguläre Ausdrücke – spezielle Klammersausdrücke

Non-capturing Groups

```
>>> re.match(r'(.+)(def)', 'abcdef').groups()
```

```
('abc', 'def')
```

```
>>> re.match(r'(?!.+)(def)', 'abcdef').groups()
```

```
('def',)
```

Lookbehind

```
>>> re.match(r'(.+)(?<=bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abc', 'def')
```

```
>>> re.match(r'(...+?)(?<!bc)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abcd', 'ef')
```

Lookahead

```
>>> re.match(r'(.+?)(?=de)(.+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'abc', 'def')
```

```
>>> re.match(r'(.+?)(?!de)(...+)', 'abcdef').group(0,1,2)
```

```
('abcdef', 'a', 'bcdef')
```

Exceptions

Wenn in Python ein Fehler auftritt bspw. Division durch 0, so wird eine **Exception** ausgelöst. Python bricht dann ab und gibt eine Fehlermeldung aus, die anzeigt, wo der Fehler aufgetreten ist.

division.py

```
import sys

divident = float(sys.argv[1])
divisor = float(sys.argv[2])

print(divident/divisor)
```

```
> python division.py 10 2
Traceback (most recent call last):
  File "/home/schmid/tmp/division.py", line 6, in <module>
    print(divident/divisor)
          ^
ZeroDivisionError: float division by zero
```

Exceptions

Exceptions können aber auch im Programm abgefangen werden:

division.py

```
import sys

try:
    dividend = float(sys.argv[1])
    divisor = float(sys.argv[2])
except ValueError:
    print('Die Argumente müssen zwei Zahlen in Ziffern sein!')
    sys.exit(1)

try:
    print(dividend/divisor)
except ZeroDivisionError as e:
    print(e)
```

Programmtest

```
> python division.py 10 5
```


Programmtest

```
> python division.py 10 5
```

```
2.0
```

```
> python division.py 10 fünf
```

Programmtest

```
> python division.py 10 5
```

```
2.0
```

```
> python division.py 10 fünf
```

```
Die Argumente müssen zwei Zahlen in Ziffern sein!
```

```
> python division.py 10 0
```

Programmtest

```
> python division.py 10 5
```

```
2.0
```

```
> python division.py 10 fünf
```

```
Die Argumente müssen zwei Zahlen in Ziffern sein!
```

```
> python division.py 10 0
```

```
Durch 0 kann nicht dividiert werden!
```

match ... case

http-error.py

```
import sys

error_code = sys.argv[1]
match error_code:
    case '400':
        print("Bad request")
    case '401' | '403':
        print("Forbidden")
    case '404':
        print("Not found")
    case '418':
        print("I'm a teapot")
    case _:
        print("Etwas anderes ist schiefgelaufen")
```

```
> python http-error.py 404
Not found
```

match ... case

Die einzelnen Fälle können auch komplexere Ausdrücke mit Variablen sein:

```
import sys

point = [float(v) for v in sys.argv[1:]]
match point:
    case (0, 0):
        print('origin')
    case (x, 0):
        print(f'X={x}')
    case (0, y):
        print(f'Y={y}')
    case (x, y):
        print(f'X={x}, Y={y}')
    case _:
        print('Etwas anderes ist schiefgelaufen')
```

for ... else

```
import sys

passwort = 'hillbilly'
for _ in range(3):
    eingabe = input('Passwort: ')
    if eingabe == passwort:
        print('Zugriff gewährt')
        break
    print('falsch')
else:
    # wird nur ausgeführt, wenn die Schleife bis zum Ende durchlief
    print('Zugriff gesperrt! Zu viele Fehlversuche!')
    sys.exit(1) # Programmabbruch
```

Der while-Befehl kann ebenso mit einer else-Klausel kombiniert werden.

Vererbung

- Klassen können Attribute und Methoden von anderen Klassen erben.
- Dadurch ist es möglich, Unterklassen von allgemeineren Klassen zu definieren, welche Funktionalität von der Elternklasse erben.

Vererbung

```
class Produkt: # Elternklasse
    def __init__(self, nummer, preis):
        self.nummer = nummer
        self.preis = preis
        self.stückzahl = 0

    def einkauf(self, stückzahl):
        self.stückzahl += stückzahl

    def verkauf(self, stückzahl):
        if self.stückzahl < stückzahl:
            raise Exception('zuwenig Produkte vorrätig')
        self.stückzahl -= stückzahl

class Brett(Produkt): # Unterklasse
    def __init__(self, nummer, preis, material, länge, breite, tiefe):
        super().__init__(nummer, preis)
        self.material = material
        self.länge = länge
        self.breite = breite
        self.tiefe = tiefe
```


Vererbung

produkte.py

```
...  
class Schrauben(Produkt):  
    def __init__(self, nummer, preis, packungsgröße, material, länge,  
        dicke, kopftyp):  
        super().__init__(nummer, preis)  
        self.material = material  
        self.packungsgröße = packungsgröße  
        self.länge = länge  
        self.dicke = dicke  
        self.kopftyp = kopftyp  
  
schrauben = Schrauben(12345, 2.5, 20, 'verzinkt', 50, 4.5, 'Torx')  
schrauben.einkauf(5)  
print(f'noch vorhanden: {schrauben.stückzahl} Stück')  
schrauben.verkauf(7)
```

```
> python produkte.py  
noch vorhanden: 5 Stück
```

```
...
```

Exception: zuwenig Produkte vorrätig

Dynamische Typisierung

Python ist eine **dynamisch typisierte** Sprache.

Variablen können Werte beliebigen Typs annehmen.

Das ist komfortabel, hat aber zwei Nachteile:

- Man sieht nicht, Argumente welchen Typs eine Funktion erwartet.
- Argumente falschen Typs werden erst zur Laufzeit erkannt.

Lösung: (optionale) Type Hints

Type Hints

```
def addiere(x: float, y:float) -> float:  
    return x + y
```

```
print(addiere(3, 5))  
print(addiere(2, -4.8))  
print(addiere(2, 'drei'))
```

```
> python addiere.py
```

Type Hints

```
def addiere(x: float, y:float) -> float:  
    return x + y  
  
print(addiere(3, 5))  
print(addiere(2, -4.8))  
print(addiere(2, 'drei'))
```

> python addiere.py

8

-2.8

... unsupported operand type(s) for +: 'int' and 'str'

⇒ Der Python-Interpreter ignoriert die Type Hints!

Aber: Viele Entwicklungsumgebungen (auch VS Code) nutzen die Type Hints, um solche Fehler zu erkennen.

Decorator, * und **

Ein **Decorator** ändert das Verhalten einer Funktion, ohne den Code zu verändern.

test.py

```
def changecase(func):  
    ''' schreibt den Ausgabertext von func in Großbuchstaben '''  
    def newfunc(*args, **kwargs):  
        result = func(*args, **kwargs)  
        return result.upper()  
    return newfunc  
  
@changecase  
def myfunction():  
    return 'Hello World!'  
  
print(myfunction())
```

```
> python test.py
```

```
HELLO WORLD!
```

Decorator dataclass

In Python 3.10 und höher gibt es den Decorator `dataclass`.

Er fügt automatisch die Methoden `__init__`, `__repr__` und `__eq__` zur Klasse hinzu.

mietwagen.py

```
from dataclasses import dataclass

@dataclass
class Mietwagen:
    hersteller: str
    model: str
    baujahr: int
    typ: str='PKW' # Attribute mit Defaultwert müssen am Ende kommen

print(Mietwagen('Volvo', 'FH 16 Aero', 2025, 'LKW'))
print(Mietwagen('Peugeot', 308, '2020')) # 2 Typfehler
```

Die Type Hints sind obligatorisch, obwohl Python nicht prüft, ob sie eingehalten werden.

Decorator dataclass

`slots=True`: Python speichert die Attribute effizient in einer Liste statt in einem Dictionary. Neue Attribute können aber nicht hinzugefügt werden.

mietwagen.py

```
from dataclasses import dataclass

@dataclass(slots=True, frozen=True) # frozen=True => immutable
class Mietwagen:
    hersteller: str
    model: str
    baujahr: int
    typ: str='PKW'

print(Mietwagen('Volvo', 'FH 16 Aero', 2025, 'LKW'))
print(Mietwagen('Peugeot', 308, '2020')) # 2 Typfehler
```

```
> python mietwagen.py
```

```
Mietwagen(hersteller='Volvo', model='FH 16 Aero', baujahr=2025, typ='LKW')
Mietwagen(hersteller='Peugeot', model=308, baujahr='2020', typ='PKW')
```

Decorator staticmethod

Methoden, die nicht auf die Attribute zugreifen, können als statisch deklariert werden.

```
import math
from dataclasses import dataclass

@dataclass(slots=True, frozen=True)
class Quader:
    breite: float
    höhe: float
    tiefe: float

    @staticmethod
    def _fläche(x, y): # hier kein self-Argument!
        return x * y

    def oberfläche(self):
        return (2 * self._fläche(self.breite, self.höhe)
                + 2 * self._fläche(self.höhe, self.tiefe)
                + 2 * self._fläche(self.breite, self.tiefe))

print(Quader(5,2,3).oberfläche())
```


Geschützte Methoden

- `_methode` geschützte Methode, die nicht extern verwendet werden sollte
- `__methode` wird automatisch in `__klasse__methode` umbenannt.
- `__methode__` Magische Methoden wie `__init__` und `__str__`

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.
- Imports kommen zuerst in der Datei.

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.
- Imports kommen zuerst in der Datei.
- separate Zeile für jeden Import

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.
- Imports kommen zuerst in der Datei.
- separate Zeile für jeden Import
- erst Standard-Bibliotheken, dann Drittanbieter-Module, dann lokale Module

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.
- Imports kommen zuerst in der Datei.
- separate Zeile für jeden Import
- erst Standard-Bibliotheken, dann Drittanbieter-Module, dann lokale Module
- Zwei Leerzeilen zwischen Klassen und Funktionen auf der obersten Ebene

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.
- Imports kommen zuerst in der Datei.
- separate Zeile für jeden Import
- erst Standard-Bibliotheken, dann Drittanbieter-Module, dann lokale Module
- Zwei Leerzeilen zwischen Klassen und Funktionen auf der obersten Ebene
- Eine Leerzeile vor und nach jeder Klassen-Methode

Der PEP-8 Style-Guide für Python-Code

Empfehlungen für gutes Programmieren

- 4 Leerzeichen einrücken
- maximal 79 Zeichen pro Zeile
- Variablen klein schreiben (Snake Case) `nummer raum_nummer`
- Klassen mit Camel Case schreiben `Nummer RaumNummer`
- Konstanten komplett groß schreiben `NUMMER RAUM_NUMMER`
- Keine unnötigen Leerzeichen in Klammern oder vor Kommas und Doppelpunkten.
- Binäre Operatoren (wie `=`, `+`, `*`) werden von einfachen Leerzeichen umschlossen.
- Imports kommen zuerst in der Datei.
- separate Zeile für jeden Import
- erst Standard-Bibliotheken, dann Drittanbieter-Module, dann lokale Module
- Zwei Leerzeilen zwischen Klassen und Funktionen auf der obersten Ebene
- Eine Leerzeile vor und nach jeder Klassen-Methode
- Leerzeilen sonst sparsam zur Gliederung einsetzen

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`
- statt mit `except:` alle Exceptions abzufangen besser nur spezifische Exceptions

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`
- statt mit `except:` alle Exceptions abzufangen besser nur spezifische Exceptions
- `string.startswith('pref')` statt `string[:4] == 'pref'` (String-Slicing) verwenden

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`
- statt mit `except:` alle Exceptions abzufangen besser nur spezifische Exceptions
- `string.startswith('pref')` statt `string[:4] == 'pref'` (String-Slicing) verwenden
- Type Hints verwenden, außer der Code wird zu unübersichtlich durch sie

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`
- statt mit `except:` alle Exceptions abzufangen besser nur spezifische Exceptions
- `string.startswith('pref')` statt `string[:4] == 'pref'` (String-Slicing) verwenden
- Type Hints verwenden, außer der Code wird zu unübersichtlich durch sie
- Dateien immer mit `with open...` öffnen

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`
- statt mit `except:` alle Exceptions abzufangen besser nur spezifische Exceptions
- `string.startswith('pref')` statt `string[:4] == 'pref'` (String-Slicing) verwenden
- Type Hints verwenden, außer der Code wird zu unübersichtlich durch sie
- Dateien immer mit `with open...` öffnen
- Strings einheitlich entweder mit `'` oder mit `"` umschließen

Der PEP-8 Style-Guide für Python-Code

- Kommentare hinter Codezeilen sparsam verwenden.
- Veraltete Kommentare sind schlechter als keine Kommentare.
- Docstrings für alle öffentlich zugänglichen Klassen, Funktionen und Methoden
- statt `var == None` und `var != None` besser `var is None` und `var is not None`
- statt `if condition == True` und `if condition == False` besser `if condition` und `if not condition`
- statt mit `except:` alle Exceptions abzufangen besser nur spezifische Exceptions
- `string.startswith('pref')` statt `string[:4] == 'pref'` (String-Slicing) verwenden
- Type Hints verwenden, außer der Code wird zu unübersichtlich durch sie
- Dateien immer mit `with open...` öffnen
- Strings einheitlich entweder mit `'` oder mit `"` umschließen
- Den Backslash `\` für Zeilenumbrüche meiden und lieber Klammern hinzufügen.

Umbruch von überlangen Codezeilen

```
def funktion(argument1, argument2, argument3,  
             argument5=None, argument6=' '):  
    pass
```

Umbruch von überlangen Codezeilen

```
def funktion(argument1, argument2, argument3,  
             argument5=None, argument6=''):  
    pass  
  
tokenisierter_text = [  
    [line.split() for line in paragraph.split('\n')]  
    for paragraph in text.split('\n\n')  
]
```

Umbruch von überlangen Codezeilen

```
def funktion(argument1, argument2, argument3,
             argument5=None, argument6=''):
    pass

tokenisierter_text = [
    [line.split() for line in paragraph.split('\n')]
    for paragraph in text.split('\n\n')
]

summe = (
    2 * quaderbreite * quaderhöhe
    + 2 * quaderbreite * quadertiefe
    + 2 * quaderhöhe * quadertiefe
)
```

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

`exit('Fehler: ..')` bricht das Programm mit einer Fehlermeldung ab.

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

`exit('Fehler: ..')` bricht das Programm mit einer Fehlermeldung ab.

`path` gibt die Liste der Import-Pfade aus.

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

`exit('Fehler: ..')` bricht das Programm mit einer Fehlermeldung ab.

`path` gibt die Liste der Import-Pfade aus.

`getsizeof(obj)` liefert die Speichergröße des Objektes.

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

`exit('Fehler: ..')` bricht das Programm mit einer Fehlermeldung ab.

`path` gibt die Liste der Import-Pfade aus.

`getsizeof(obj)` liefert die Speichergröße des Objektes.

```
>>> sys.getsizeof(1)
```

```
28
```

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

`exit('Fehler: ..')` bricht das Programm mit einer Fehlermeldung ab.

`path` gibt die Liste der Import-Pfade aus.

`getsizeof(obj)` liefert die Speichergröße des Objektes.

```
>>> sys.getsizeof(1)
```

```
28
```

```
>>> sys.getsizeof(1111111111)
```

```
32
```

Python-Bibliothek sys

`argv` Liste mit Programmname und Kommandozeilen-Argumenten

`version` gibt die Python-Version aus

`platform` Name des Betriebssystems (linux, win32, darwin)

`stdin stdout stderr` Standard-Ein- und -Ausgabedateien

`exit('Fehler: ..')` bricht das Programm mit einer Fehlermeldung ab.

`path` gibt die Liste der Import-Pfade aus.

`getsizeof(obj)` liefert die Speichergröße des Objektes.

```
>>> sys.getsizeof(1)
```

```
28
```

```
>>> sys.getsizeof(1111111111)
```

```
32
```

```
>>> sys.getsizeof('1')
```

```
44
```

Python-Bibliothek math

Funktionen: `sqrt pow exp log sin cos tan`

Python-Bibliothek math

Funktionen: `sqrt pow exp log sin cos tan`

Konstanten: `pi e inf nan`

Python-Bibliothek math

Funktionen: `sqrt pow exp log sin cos tan`

Konstanten: `pi e inf nan`

Rundungen: `ceil floor trunc`

Python-Bibliothek math

Funktionen: `sqrt pow exp log sin cos tan`

Konstanten: `pi e inf nan`

Rundungen: `ceil floor trunc`

Produkt: `prod(zahlen)` (ähnlich der sum-Operation)

Python-Bibliothek math

Funktionen: `sqrt pow exp log sin cos tan`

Konstanten: `pi e inf nan`

Rundungen: `ceil floor trunc`

Produkt: `prod(zahlen)` (ähnlich der sum-Operation)

```
>>> math.prod([1, 2, 3, 4, 5])  
120
```

Python-Bibliothek collections

`defaultdict` Dictionary mit Defaultwert für unbekannte Keys

Python-Bibliothek collections

`defaultdict` Dictionary mit Defaultwert für unbekannte Keys

`Counter` spezielles Dictionary zur Berechnung von Häufigkeiten

Python-Bibliothek collections

`defaultdict` Dictionary mit Defaultwert für unbekannte Keys

`Counter` spezielles Dictionary zur Berechnung von Häufigkeiten

`namedtuple` Tupel, dessen Elemente mit Namen adressiert werden können

Python-Bibliothek collections

`defaultdict` Dictionary mit Defaultwert für unbekannte Keys

`Counter` spezielles Dictionary zur Berechnung von Häufigkeiten

`namedtuple` Tupel, dessen Elemente mit Namen adressiert werden können

```
>>> Punkt = collections.namedtuple('Punkt', ['x', 'y'])
>>> punkt = Punkt(3, 4)
>>> print(f'x={punkt.x} y={punkt.y}')
x=3 y=4
```

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

`choice(liste)` wählt ein zufälliges Element

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

`choice(liste)` wählt ein zufälliges Element

`choices(liste, k)` wählt zufällig k Elemente mit Zurücklegen

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

`choice(liste)` wählt ein zufälliges Element

`choices(liste, k)` wählt zufällig k Elemente mit Zurücklegen

`sample(liste, k)` wählt zufällig k Elemente ohne Zurücklegen

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

`choice(liste)` wählt ein zufälliges Element

`choices(liste, k)` wählt zufällig k Elemente mit Zurücklegen

`sample(liste, k)` wählt zufällig k Elemente ohne Zurücklegen

```
>>> random.choices([1,2,3], 4  
[3, 3, 2, 2]
```

Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

`choice(liste)` wählt ein zufälliges Element

`choices(liste, k)` wählt zufällig k Elemente mit Zurücklegen

`sample(liste, k)` wählt zufällig k Elemente ohne Zurücklegen

```
>>> random.choices([1,2,3], 4
```

```
[3, 3, 2, 2]
```

```
>>> random.sample([1,2,3], 3
```

```
[1, 3, 2]
```


Python-Bibliothek random

`shuffle(liste)` ordnet die Elemente zufällig um (in-place)

`random()` gibt eine Zufallszahl zwischen 0 und 1 zurück

`randomint(a, b)` liefert eine ganze Zahl zwischen a und b (inklusive)

`uniform(a, b)` liefert eine Gleitkommazahl zwischen a und b

`choice(liste)` wählt ein zufälliges Element

`choices(liste, k)` wählt zufällig k Elemente mit Zurücklegen

`sample(liste, k)` wählt zufällig k Elemente ohne Zurücklegen

```
>>> random.choices([1,2,3], 4
```

```
[3, 3, 2, 2]
```

```
>>> random.sample([1,2,3], 3
```

```
[1, 3, 2]
```

```
>>> random.sample([1,2,3], 4
```

```
...Sample larger than population...
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
>>> path.stem
'datei'
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
>>> path.stem
'datei'
>>> path.suffix
'.txt'
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
>>> path.stem
'datei'
>>> path.suffix
'.txt'
>>> path.parent
PosixPath('dir/subdir')
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
>>> path.stem
'datei'
>>> path.suffix
'.txt'
>>> path.parent
PosixPath('dir/subdir')
>>> path.parts
('dir', 'subdir', 'datei.txt')
```

Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
>>> path.stem
'datei'
>>> path.suffix
'.txt'
>>> path.parent
PosixPath('dir/subdir')
>>> path.parts
('dir', 'subdir', 'datei.txt')
>>> Path.home()
PosixPath('/home/schmid')
```


Python-Bibliothek pathlib

```
>>> from pathlib import Path
>>> path = Path('dir/subdir/datei.txt')
>>> path
PosixPath('dir/subdir/datei.txt')
>>> path.name
'datei.txt'
>>> path.stem
'datei'
>>> path.suffix
'.txt'
>>> path.parent
PosixPath('dir/subdir')
>>> path.parts
('dir', 'subdir', 'datei.txt')
>>> Path.home()
PosixPath('/home/schmid')
>>> Path.cwd()
PosixPath('/home/schmid/tmp')
```

Python-Bibliothek numpy

- Bibliothek für Berechnungen mit mehrdimensionalen Arrays
- schnell da weitgehend in C implementiert
- geringer Speicherplatzbedarf
- Array-Operationen
- viele integrierte Funktionen (statistische Tests, Fourier-Transformation, ...)

Python-Bibliothek numpy

- Bibliothek für Berechnungen mit mehrdimensionalen Arrays
- schnell da weitgehend in C implementiert
- geringer Speicherplatzbedarf
- Array-Operationen
- viele integrierte Funktionen (statistische Tests, Fourier-Transformation, ...)

```
>>> m = np.random.rand(3,4).round()
```

```
>>> m
```

```
array([[0.96, 0.49, 0.69, 0.72],  
       [0.59, 0.98, 0.27, 0.06],  
       [0.31, 0.96, 0.97, 0.18]])
```

Python-Bibliothek numpy

- Bibliothek für Berechnungen mit mehrdimensionalen Arrays
- schnell da weitgehend in C implementiert
- geringer Speicherplatzbedarf
- Array-Operationen
- viele integrierte Funktionen (statistische Tests, Fourier-Transformation, ...)

```
>>> m = np.random.rand(3,4).round()
```

```
>>> m
```

```
array([[0.96, 0.49, 0.69, 0.72],  
       [0.59, 0.98, 0.27, 0.06],  
       [0.31, 0.96, 0.97, 0.18]])
```

```
>>> m+5
```

```
array([[5.96, 5.49, 5.69, 5.72],  
       [5.59, 5.98, 5.27, 5.06],  
       [5.31, 5.96, 5.97, 5.18]])
```

Python-Bibliothek numpy

- Bibliothek für Berechnungen mit mehrdimensionalen Arrays
- schnell da weitgehend in C implementiert
- geringer Speicherplatzbedarf
- Array-Operationen
- viele integrierte Funktionen (statistische Tests, Fourier-Transformation, ...)

```
>>> m = np.random.rand(3,4).round() >>> m.T
>>> m                                array([[0.08, 0.39, 0.73],
array([[0.96, 0.49, 0.69, 0.72],      [0.01, 0.33, 0.6 ],
      [0.59, 0.98, 0.27, 0.06],      [0.98, 0.72, 0.15],
      [0.31, 0.96, 0.97, 0.18]])     [0.45, 0.6 , 0.38]])

>>> m+5
array([[5.96, 5.49, 5.69, 5.72],
      [5.59, 5.98, 5.27, 5.06],
      [5.31, 5.96, 5.97, 5.18]])
```

Python-Bibliothek numpy

- Bibliothek für Berechnungen mit mehrdimensionalen Arrays
- schnell da weitgehend in C implementiert
- geringer Speicherplatzbedarf
- Array-Operationen
- viele integrierte Funktionen (statistische Tests, Fourier-Transformation, ...)

```
>>> m = np.random.rand(3,4).round() >>> m.T
>>> m                                array([[0.08, 0.39, 0.73],
array([[0.96, 0.49, 0.69, 0.72],      [0.01, 0.33, 0.6 ],
      [0.59, 0.98, 0.27, 0.06],      [0.98, 0.72, 0.15],
      [0.31, 0.96, 0.97, 0.18]])     [0.45, 0.6 , 0.38]])

>>> m+5                               >>> m.dot(m.T) # Matrixprodukt
array([[5.96, 5.49, 5.69, 5.72],    array([[1.1694, 1.0101, 0.3824],
      [5.59, 5.98, 5.27, 5.06],      [1.0101, 1.1394, 0.8187],
      [5.31, 5.96, 5.97, 5.18]])     [0.3824, 0.8187, 1.0598]])
```

Python-Bibliothek pandas

- implementiert Datenbank-ähnliche Operationen
- importiert CSV-Dateien, Excel-Tabellen, SQL-Tabellen
- basiert auf Numpy

Python-Bibliothek pandas

- implementiert Datenbank-ähnliche Operationen
- importiert CSV-Dateien, Excel-Tabellen, SQL-Tabellen
- basiert auf Numpy

```
>>> import pandas as pd
>>> datentypen = {'Alter': int, 'Gehalt':float}
>>> df = pd.read_csv('mitarbeiter.csv', dtype=datentypen)
>>> df
```

	Name	Abteilung	Alter	Gehalt
0	Stefan Müller	IT	32	85000.0
1	Gretel Maier	Marketing	25	45000.0

Python-Bibliothek pandas

- implementiert Datenbank-ähnliche Operationen
- importiert CSV-Dateien, Excel-Tabellen, SQL-Tabellen
- basiert auf Numpy

```
>>> import pandas as pd
>>> datentypen = {'Alter': int, 'Gehalt':float}
>>> df = pd.read_csv('mitarbeiter.csv', dtype=datentypen)
>>> df
```

	Name	Abteilung	Alter	Gehalt
0	Stefan Müller	IT	32	85000.0
1	Gretel Maier	Marketing	25	45000.0

```
>>> df[df['Alter'] < 30] # jünger als 30
```

	Name	Abteilung	Alter	Gehalt
1	Gretel Maier	Marketing	25	45000.0

Python-Bibliothek pandas

- implementiert Datenbank-ähnliche Operationen
- importiert CSV-Dateien, Excel-Tabellen, SQL-Tabellen
- basiert auf Numpy

```
>>> import pandas as pd
>>> datentypen = {'Alter': int, 'Gehalt':float}
>>> df = pd.read_csv('mitarbeiter.csv', dtype=datentypen)
>>> df
```

	Name	Abteilung	Alter	Gehalt
0	Stefan Müller	IT	32	85000.0
1	Gretel Maier	Marketing	25	45000.0

```
>>> df[df['Alter'] < 30] # jünger als 30
```

	Name	Abteilung	Alter	Gehalt
1	Gretel Maier	Marketing	25	45000.0

```
>>> df[(df['Abteilung'] == 'IT') & (df['Gehalt'] > 50000)]
```

	Name	Abteilung	Alter	Gehalt
0	Stefan Müller	IT	32	85000.0

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

`inv` berechnet die Inverse einer Matrix

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

`inv` berechnet die Inverse einer Matrix

`det` berechnet die Determinante einer Matrix

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

`inv` berechnet die Inverse einer Matrix

`det` berechnet die Determinante einer Matrix

`eig` berechnet Eigenwerte

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

`inv` berechnet die Inverse einer Matrix

`det` berechnet die Determinante einer Matrix

`eig` berechnet Eigenwerte

`svd` Singulärwertzerlegung

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

`inv` berechnet die Inverse einer Matrix

`det` berechnet die Determinante einer Matrix

`eig` berechnet Eigenwerte

`svd` Singulärwertzerlegung

`norm`, `binom`, ... Verteilungen

Python-Bibliothek scipy

- große Sammlung von Algorithmen für Wissenschaft und Datenanalyse
- basiert auf Numpy

`quad` integriert Funktionen über ein Intervall

`minimize` minimiert eine Funktion

`curve_fit` passt eine benutzerdefinierte Funktion an Messdaten an

`root` sucht Nullstellen von nichtlinearen Gleichungssystemen

`linprog` optimiert Zielfunktionen mit Nebenbedingungen

`inv` berechnet die Inverse einer Matrix

`det` berechnet die Determinante einer Matrix

`eig` berechnet Eigenwerte

`svd` Singulärwertzerlegung

`norm`, `binom`, ... Verteilungen

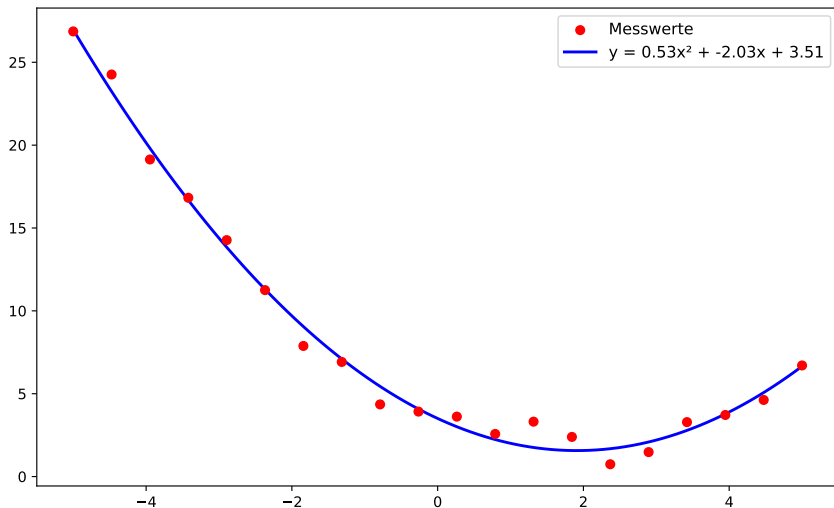
`spectrogram` Spektrogramm eines Signals

Visualisierung mit matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

def funktion(x, a, b, c):
    return a * x**2 + b * x + c
# Messwerte mit Rauschen erzeugen
x = np.linspace(-5, 5, 20)
y = funktion(x, 0.5, -2, 5) + np.random.normal(-1, 1, size=20)
# optimale Werte für a, b und c aus Messwerten ableiten
popt, pcov = curve_fit(funktion, x, y)
a_opt, b_opt, c_opt = popt # berechnete Parameter
# X-Werte für eine glatte Kurve generieren
x_kurve = np.linspace(-5, 5, 100)
y_kurve = funktion(x_kurve, a_opt, b_opt, c_opt)
# Visualisierung mit Matplotlib
plt.figure(figsize=(10, 6))
plt.scatter(x, y, color="red", label="Messwerte", zorder=3)
plt.plot(x_kurve, y_kurve, color="blue", linewidth=2,
        label=f"y = {a_opt:.2f}x2 + {b_opt:.2f}x + {c_opt:.2f}")
plt.legend(fontsize=11)
plt.show()
```

Visualisierung mit matplotlib



Paket-Installation

Fehlende Python-Pakete können mit pip installiert werden:

```
pip install pandas
```

Neuere Linux-Distributionen verweigern allerdings diese Art der Installation, weil die Gefahr besteht, dass wichtige Python-Programme durch die Installation beeinträchtigt werden.

Stattdessen wird die Verwendung einer separaten **virtuellen Umgebung** für jedes einzelne Python-Projekt empfohlen!

Virtuelle Umgebungen

Sie haben ein Programm **myprogram.py** geschrieben, welches **pandas** benötigt.

Erstellen Sie im aktuellen Projektverzeichnis eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Virtuelle Umgebungen

Sie haben ein Programm **myprogram.py** geschrieben, welches **pandas** benötigt.

Erstellen Sie im aktuellen Projektverzeichnis eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Virtuelle Umgebungen

Sie haben ein Programm **myprogram.py** geschrieben, welches **pandas** benötigt.

Erstellen Sie im aktuellen Projektverzeichnis eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install pandas
```

Virtuelle Umgebungen

Sie haben ein Programm **myprogram.py** geschrieben, welches **pandas** benötigt.

Erstellen Sie im aktuellen Projektverzeichnis eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install pandas
```

Erstellen Sie die Liste der installierten Python-Pakete:

```
pip freeze > requirements.txt
```

Virtuelle Umgebungen

Sie haben ein Programm **myprogram.py** geschrieben, welches **pandas** benötigt.

Erstellen Sie im aktuellen Projektverzeichnis eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install pandas
```

Erstellen Sie die Liste der installierten Python-Pakete:

```
pip freeze > requirements.txt
```

Testen Sie Ihr Programm:

```
python myprogram.py
```

Virtuelle Umgebungen

Sie haben ein Programm **myprogram.py** geschrieben, welches **pandas** benötigt.

Erstellen Sie im aktuellen Projektverzeichnis eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install pandas
```

Erstellen Sie die Liste der installierten Python-Pakete:

```
pip freeze > requirements.txt
```

Testen Sie Ihr Programm:

```
python myprogram.py
```

Verlassen Sie zum Schluss die virtuelle Umgebung:

```
deactivate
```

Virtuelle Umgebungen

Erneutes Aktivieren einer virtuellen Umgebung:

```
source .venv/bin/activate
```

Ausführen des Programms:

```
python myprogram.py
```

Verlassen der virtuellen Umgebung:

```
deactivate
```

Virtuelle Umgebungen

Aufgabe: Installation von **myprogram.py** auf einem neuen Rechner

Kopieren Sie **myprogram.py** und **requirements.txt** auf den anderen Rechner und wechseln Sie in das Verzeichnis, in dem sich die beiden Dateien befinden.

Virtuelle Umgebungen

Aufgabe: Installation von **myprogram.py** auf einem neuen Rechner

Kopieren Sie **myprogram.py** und **requirements.txt** auf den anderen Rechner und wechseln Sie in das Verzeichnis, in dem sich die beiden Dateien befinden.

Erstellen Sie eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Virtuelle Umgebungen

Aufgabe: Installation von **myprogram.py** auf einem neuen Rechner

Kopieren Sie **myprogram.py** und **requirements.txt** auf den anderen Rechner und wechseln Sie in das Verzeichnis, in dem sich die beiden Dateien befinden.

Erstellen Sie eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Virtuelle Umgebungen

Aufgabe: Installation von **myprogram.py** auf einem neuen Rechner

Kopieren Sie **myprogram.py** und **requirements.txt** auf den anderen Rechner und wechseln Sie in das Verzeichnis, in dem sich die beiden Dateien befinden.

Erstellen Sie eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install -r requirements
```

Virtuelle Umgebungen

Aufgabe: Installation von **myprogram.py** auf einem neuen Rechner

Kopieren Sie **myprogram.py** und **requirements.txt** auf den anderen Rechner und wechseln Sie in das Verzeichnis, in dem sich die beiden Dateien befinden.

Erstellen Sie eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install -r requirements
```

Testen Sie das Programm:

```
python myprogram.py
```

Virtuelle Umgebungen

Aufgabe: Installation von **myprogram.py** auf einem neuen Rechner

Kopieren Sie **myprogram.py** und **requirements.txt** auf den anderen Rechner und wechseln Sie in das Verzeichnis, in dem sich die beiden Dateien befinden.

Erstellen Sie eine virtuelle Umgebung:

```
python3 -m venv .venv
```

Aktivieren Sie die virtuelle Umgebung:

```
source .venv/bin/activate
```

Installieren Sie die benötigten Python-Pakete:

```
pip install -r requirements
```

Testen Sie das Programm:

```
python myprogram.py
```

Verlassen Sie die virtuelle Umgebung mit

```
deactivate
```