

# Python

N. Kaelin

10. März 2019

## Inhaltsverzeichnis

<b>I</b>	<b>Lektion 1: Variablen und Datentypen</b>	<b>3</b>
<b>1</b>	<b>Datentypen</b>	<b>3</b>
1.1	Numerische Datentypen . . . . .	3
1.1.1	Arithmetische Operationen . . . . .	4
1.1.2	Vergleichende Operatoren . . . . .	4
1.1.3	Bitweise Operatoren für den Datentyp <code>int</code> . . . . .	4
1.1.4	Methoden nur für den Datentyp <code>complex</code> . . . . .	5
1.2	Sequentielle Datentypen . . . . .	5
1.3	Assoziative Datentypen . . . . .	5
1.4	Mengen . . . . .	6
<b>II</b>	<b>Lektion 2: Verzweigungen, Schleifen und Funktionen</b>	<b>7</b>
<b>2</b>	<b>Verzweigungen</b>	<b>7</b>
2.1	<code>if</code> . . . . .	7
2.1.1	<code>if</code> -Anweisung mit <code>else</code> -Zweig . . . . .	7
2.1.2	<code>elif</code> -Zweige . . . . .	7
<b>3</b>	<b>Schleifen</b>	<b>8</b>
3.1	<code>while</code> . . . . .	8
3.1.1	Durchlauf beenden und zurück nach oben . . . . .	8
3.1.2	<code>while</code> -Schleife abbrechen . . . . .	8
3.1.3	<code>else</code> -Teil . . . . .	8
3.2	<code>for</code> . . . . .	9
3.2.1	<code>else</code> -Teil . . . . .	9
<b>4</b>	<b>Funktionen</b>	<b>9</b>
4.1	Funktionsdefinition . . . . .	9
4.2	Aufruf . . . . .	10
4.3	Weiteres . . . . .	10
4.3.1	Standardwert für Parameter . . . . .	10
4.3.2	Mehrere Rückgabewerte . . . . .	10
4.3.3	Variable Anzahl von Argumenten . . . . .	10
4.3.4	Argumente entpacken . . . . .	11
4.3.5	Beliebige Schlüsselwort-Parameter . . . . .	11
4.3.6	Schlüsselwortparameter entpacken . . . . .	11
4.3.7	Globale Variablen . . . . .	11
4.3.8	Docstring - Funktion dokumentieren . . . . .	12
4.3.9	Call-by-object-reference . . . . .	12

<b>5</b>	<b>Exceptions</b>	<b>13</b>
5.1	Unspezifische Exceptions abfangen . . . . .	13
5.2	Abfangen mehrerer Exceptions . . . . .	13
5.3	else-Teil . . . . .	14
5.4	finally-Teil . . . . .	14
5.5	Exceptions generieren . . . . .	14
<b>6</b>	<b>Dateien</b>	<b>14</b>
6.1	Datei öffnen . . . . .	14
6.2	Dateien lesen und schreiben . . . . .	14
6.2.1	Datei lesen . . . . .	15
6.2.2	Datei schreiben . . . . .	15
6.2.3	glob . . . . .	16
6.2.4	os.path . . . . .	16
<b>7</b>	<b>Strings</b>	<b>16</b>
7.1	Stringformatierung . . . . .	16
7.1.1	im C-Stil (à la printf) . . . . .	16
7.1.2	mit format() . . . . .	17
7.1.3	mit Stringlitterale . . . . .	17
7.1.4	mit string-Methoden . . . . .	18
7.2	Alles über Strings . . . . .	18
7.2.1	Strings aufspalten . . . . .	18
7.2.2	Strings kombinieren . . . . .	19
7.2.3	Suchen von Teilstrings . . . . .	19
7.2.4	Ersetzen von Teilstrings . . . . .	19
7.2.5	Strings bereinigen . . . . .	19
7.2.6	Klein- und Grossbuchstaben . . . . .	20
7.2.7	Strings testen . . . . .	20

## Teil I

# Lektion 1: Variablen und Datentypen

## 1 Datentypen

- Variablen bezeichnen keinen bestimmten Typ.
- Dynamische Typdeklaration
  - **Automatische Zuweisung** des Datentyps bei Deklaration
  - Datentyp ist während dem Programmablauf **veränderbar**
  - Wert- und Typänderung erlaubt!

Datentyp	Beschreibung	False-Wert
NoneType	Indikator für nichts, keinen Wert	None
<b>Numerische Datentypen</b>		
int	Ganze Zahlen	0
float	Gleitkommazahlen	0.0
bool	Boolesche Werte	False
complex	Komplexe Zahlen	0 + 0j
<b>Sequenzielle Datentypen</b>		
str	Zeichenketten oder Strings	"
list	Listen (veränderlich)	[]
tuple	Tupel (unveränderlich)	()
bytes	Sequenz von Bytes (unveränderlich)	b"
bytearray	Sequenz von Bytes (veränderlich)	bytearray(b")
<b>Assoziative Datentypen</b>		
dict	Dictionary (Schlüssel-Wert-Paare)	
<b>Mengen</b>		
set	Menge mit einmalig vorkommenden Objekten	set()
frozenset	Wie set jedoch unveränderlich	frozenset()

- Python erkennt den Datentyp automatisch
- Python ordnet jeder Variablen den Datentyp zu
- Datentypen prüfen:
 

```
type(object)
isinstance(object, ct)
```
- Python achtet auf Typverletzungen
- Python kennt keine implizite Typumwandlung

### 1.1 Numerische Datentypen

- bool
- int
- float
- complex

### 1.1.1 Arithmetische Operationen

Operator	Beschreibung
$x + y$	Summe von $x$ und $y$
$x - y$	Differenz von $x$ und $y$
$x * y$	Produkt von $x$ und $y$
$x / y$	Quotient von $x$ und $y$
$x // y$	Ganzzahliger Quotient <sup>1</sup> von $x$ und $y$
$x \% y$	Rest der Division <sup>1</sup> von $x$ durch $y$
$+x$	Positives Vorzeichen
$-x$	Negatives Vorzeichen
$\text{abs}(x)$	Betrag von $x$
$x ** y$	Potenzieren, $x^y$

<sup>1</sup>Nicht definiert für den Datentyp `complex`

---

**Achtung:** `x++` und `x-` gibt es **nicht**, aber `x += 1`, `x -= 1`, `x *= 2`, ...

---

### 1.1.2 Vergleichende Operatoren

Operator	Beschreibung
<code>==</code>	wahr, wenn $x$ und $y$ gleich sind
<code>!=</code>	wahr, wenn $x$ und $y$ verschieden sind
<code>&lt;</code>	wahr, wenn $x$ kleiner als $y$ ist <sup>2</sup>
<code>&lt;=</code>	wahr, wenn $x$ kleiner oder gleich $y$ ist <sup>2</sup>
<code>&gt;</code>	wahr, wenn $x$ grösser als $y$ ist <sup>2</sup>
<code>&gt;=</code>	wahr, wenn $x$ grösser oder gleich $y$ ist <sup>2</sup>

<sup>2</sup>Nicht definiert für den Datentyp `complex`

### 1.1.3 Bitweise Operatoren für den Datentypen `int`

Operator	Beschreibung
$x \& y$	bitweises UND von $x$ und $y$
$x   y$	bitweises ODER von $x$ und $y$
$x \wedge y$	bitweises EXOR von $x$ und $y$
$\sim x$	bitweises Komplement von $x$
$x \ll n$	Bit-Verschiebung um $n$ Stellen nach links
$x \gg n$	Bit-Verschiebung um $n$ Stellen nach rechts

### 1.1.4 Methoden nur für den Datentyp `complex`

Methode	Beschreibung
<code>x.real</code>	Realteil von <code>x</code> als Gleitkommazahl
<code>x.imag</code>	Imaginärteil von <code>x</code> als Gleitkommazahl
<code>x.conjugate()</code>	Liefert die zu <code>x</code> konjugiert komplexe Zahl

## 1.2 Sequentielle Datentypen

- `str`
- `list`
- `tuple`
- `bytes`
- `bytearray`

Die folgenden Operatoren sind für **alle** sequentiellen Datentypen definiert:

Operator	Beschreibung
<code>x in s</code>	Prüft, ob <code>x</code> in <code>s</code> enthalten ist.
<code>x not in s</code>	Prüft, ob <code>x</code> nicht in <code>s</code> enthalten ist.
<code>s + t</code>	Verkettung der beiden Sequenzen <code>s</code> und <code>t</code> .
<code>s * n</code>	Verkettung von <code>n</code> Kopien der Sequenz <code>s</code> .
<code>s[i]</code>	Liefert das <code>i</code> -te Element von <code>s</code> .
<code>s[i:j]</code>	Liefert den Ausschnitt aus <code>s</code> von <code>i</code> bis <code>j</code> .
<code>s[i:j:k]</code>	Liefert jedes <code>k</code> -te Element im Ausschnitt von <code>s</code> zwischen <code>i</code> und <code>j</code> .
<code>len(s)</code>	Liefert die Anzahl Elemente in der Sequenz <code>s</code> .
<code>max(s)</code>	Liefert das grösste Element in <code>s</code> (sofern eine Ordnung definiert ist).
<code>min(s)</code>	Liefert das kleinste Element in <code>s</code> (sofern eine Ordnung definiert ist).
<code>s.index(x)</code>	Liefert den Index des ersten Vorkommens von <code>x</code> in <code>s</code> .
<code>s.count(x)</code>	Zählt, wie oft <code>x</code> in <code>s</code> vorkommt.

## 1.3 Assoziative Datentypen

- `dict`

Operator	Beschreibung
<code>len(d)</code>	Liefert die Anzahl Schlüssel-Wert-Paare in <code>d</code>
<code>d[k]</code>	Zugriff auf den Wert mit dem Schlüssel <code>k</code>
<code>k in d</code>	Liefert <code>True</code> , wenn der Schlüssel <code>k</code> in <code>d</code> ist.
<code>k not in d</code>	Liefert <code>True</code> , wenn der Schlüssel <code>k</code> nicht in <code>d</code> ist.

Operator	Beschreibung
<code>d.clear()</code>	Löscht alle Elemente aus dem Dictionary.
<code>d.copy()</code>	Erstellt eine Kopie des Dictionaries.
<code>d.get([k, [x]])</code>	Gibt den Wert des Schlüssels <code>k</code> zurück, ansonsten den Wert <code>[x]</code> .
<code>d.items()</code>	Gibt eine Liste der Schlüssel-Wert-Paare als Tuple zurück.
<code>d.keys()</code>	Gibt eine Liste aller Schlüsselwerte zurück.
<code>d.update(d2)</code>	Fügt ein Dictionary <code>d2</code> zu <code>d</code> hinzu.
<code>d.pop(k)</code>	Entfernt das Element mit Schlüssel <code>k</code> .
<code>d.popitem()</code>	Entfernt das zuletzt eingefügte Schlüssel-Wert-Paar.
<code>d.setdefault(k, [x])</code>	Setzt den Wert <code>[x]</code> für den Schlüssel <code>k</code> .

## 1.4 Mengen

- `set`
- `frozenset`

## Teil II

# Lektion 2: Verzweigungen, Schleifen und Funktionen

## 2 Verzweigungen

### 2.1 if

listings/v2\_if1.py

```
if Bedingung:
    Anweisung1
    Anweisung2
```

Anweisungen 1 & 2 nur ausführen, wenn die Bedingung **wahr** ist.

---

**Achtung:** Alle Anweisungen im gleichen Codeblock müssen gleich eingerückt sein, z.B. mit vier Leerzeichen, sonst wird ein Fehler ausgegeben.<

---

#### 2.1.1 if-Anweisung mit else-Zweig

listings/v2\_if2.py

```
if Bedingung:
    Anweisung1
    Anweisung2
else:
    Anweisung3
    Anweisung4
```

- Anweisungen 1 & 2, falls Bedingung **wahr**
- Anweisungen 3 & 4, falls Bedingung **unwahr**

Für jeden Datentyp gibt es einen Wert, der als **unwahr** gilt:

Datentyp	False-Wert
NoneType	None
int	0
float	0.0
bool	False
complex	0 + 0j
str	"
list	[]
tuple	()
bytes	b"
bytearray	bytearray(b")
dict	
set	set()
frozenset	frozenset()

#### 2.1.2 elif-Zweige

listings/v2\_if3.py

```
if Bedingung1:
    Anweisung1
elif Bedingung2:
    Anweisung2
elif Bedingung3:
    Anweisung3
else:
    Anweisung4
```

elif = else if

---

**Achtung:** Python kennt keine switch-case-Anweisung.

---

## 3 Schleifen

### 3.1 while

listings/v2\_while1.py

```
while Bedingung:
    Anweisung1
```

- Anweisung1 wird wiederholt, solange die Bedingung **wahr** ist
- Einrücken des Codeblocks

#### 3.1.1 Durchlauf beenden und zurück nach oben

---

**Achtung:** Python kennt keine do-while-Schleife.

---

listings/v2\_while2.py

```
while Bedingung:
    Anweisung1
    if Ausnahme:
        continue
    Anweisung2
```

continue beendet den aktuellen Durchlauf und springt nach oben.

#### 3.1.2 while-Schleife abbrechen

listings/v2\_while3.py

```
while Bedingung:
    Anweisung1
    if Fehler:
        break
```

break bricht die while-Schleife vorzeitig ab

#### 3.1.3 else-Teil

listings/v2\_while4.py



```
while Bedingung:
    Anweisung1
    if Fehler:
        break
else:
    Anweisung2
```

else-Teil: wenn die Schleife **nicht** durch break abgebrochen wurde

### 3.2 for

listings/v2\_for1.py

```
for Variable in Sequenz:
    Anweisung1
```

- dient zur Iteration einer Sequenz
- Sequenz muss ein iterierbares Objekt sein:  
list, tuple, dict, str, bytes, bytearray, set, frozenset

#### 3.2.1 else-Teil

listings/v2\_for2.py

```
for Variable in Sequenz:
    Anweisung1
else:
    Anweisung2
```

else-Teil wie bei der while-Schleife

## 4 Funktionen

Python besitzt eine grosse Standard-Bibliothek, z.B.:

listings/v2\_func1.py

```
import time # time.time(), time.sleep()
import math # math.pi, math.cos(), math.log10()
import zipfile # ZIP-Dateien manipulieren
import socket # UDP-/TCP-Kommunikation
```

<https://docs.python.org/3/library/>

und eingebaute Datentypen:

<https://docs.python.org/3/library/stdtypes.html>

und eingebaute Funktionen:

<https://docs.python.org/3/library/functions.html>

### 4.1 Funktionsdefinition

einfache Funktionsdefinition:

listings/v2\_func2.py

```
def Funktionsname(Parameterliste):
    Anweisungen
```

Beispiel:

listings/v2\_func3.py

```
def begruessung(vorname, nachname):  
    print('Hallo', vorname, nachname)
```

- Der Funktionsname kann frei gewählt werden
- Parameternamen durch Kommas trennen
- Codeblock gleichmässig einrücken

Der Rückgabewert der Funktion ist None, falls nichts angegeben wird.

listings/v2\_func4.py

```
def gruss(name):  
    print('Hallo', name)
```

return-Anweisung beendet den Funktionsaufruf mit Rückgabewert:

listings/v2\_func5.py

```
def summe(a, b):  
    return a + b
```

- leere return-Anweisung liefert None zurück
- mehrere return-Anweisungen sind erlaubt, wie in C/C++

## 4.2 Aufruf

listings/v2\_func6.py

```
resultat1 = summe(2, 3)  
resultat2 = summe(a=10, b=2)    # Schluesselwortparameter  
resultat3 = summe(b=2, a=10)    # Reihenfolge ist egal  
resultat4 = summe(20, b=4)      # zuerst die namelosen
```

## 4.3 Weiteres

### 4.3.1 Standardwert für Parameter

listings/v2\_func7.py

```
def rosen(farbe='rot'):  
    print('Rosen_sind_' + farbe + '.')
```

```
rosen() # Aufruf 1  
rosen('gelb') # Aufruf 2
```

### 4.3.2 Mehrere Rückgabewerte

listings/v2\_func8.py

```
def summe_und_differenz(a, b):  
    return (a + b, a - b)    # Tupel
```

```
summe, differenz = summe_und_differenz(5, 3)    # Tupel entpacken
```

### 4.3.3 Variable Anzahl von Argumenten

listings/v2\_func9.py

```
def mittelwert(a, *args):    # a ist zwingend
    print('a_=', 1)
    print('args_=', args)    # die restlichen Argumente sind im Tupel args
    a += sum(args)
    return a/len(args) + 1

mittelwert(2, 3, 7)
```

#### 4.3.4 Argumente entpacken

listings/v2\_func10.py

```
def distanz(x, y, z):
    print('x_=', x)
    print('y_=', y)
    print('z_=', z)
    return (x**2 + y**2 + z**2)**0.5

position = (2, 3, 6)
distanz(*position)    # Tupel entpacken
```

#### 4.3.5 Beliebige Schlüsselwort-Parameter

listings/v2\_func11.py

```
def einfache_funktion(x, **kwargs):
    print('x_=', x)
    print('kwargs_=', kwargs)    # die restlichen Argumente sind im Dictionary
    kwargs

einfache_funktion(x='Hallo', farbe='rot', durchmesser=10)
```

#### 4.3.6 Schlüsselwortparameter entpacken

listings/v2\_func12.py

```
punkt = {'x':1, 'y':2, 'z':2}
distanz(**punkt)    # Dictionary entpacken
```

#### 4.3.7 Globale Variablen

listings/v2\_func13.py

```
modul = 'Python'    # globale Variable

def anmeldung():
    print(modul)    # Variable existiert bereits ausserhalb der Funktion

anmeldung()    # Ausgabe: Python

def wechseln():
    modul = 'C++'    # erstellt eine neue lokale Variable
    print('lokal:', modul)

wechseln()    # Ausgabe: lokal: C++
```

```

print('global:', modul) # Ausgabe: global: Python

def wirklich_wechseln():
    global modul    #referenzieren auf die globale Variable
    print('lokal:', modul)

wechseln() # Ausgabe: lokal: C++
print(global:), modul # Ausgabe: global: C++

```

### 4.3.8 Docstring - Funktion dokumentieren

PEP 257 - Docstring Conventions <https://www.python.org/dev/peps/pep-0257>

listings/v2\_func14.py

```

def meine_funktion(a, b):
    '''Gibt die Argumente a und b in umgekehrter Reihenfolge als Tupel zurueck.'''
    return(b, a)

meine_funktion.__doc__ # Ausgabe: 'Gibt die Arguemnte ...'
help(meine_funktion)

```

### 4.3.9 Call-by-object-reference

mit veränderlichen Objekten:

listings/v2\_func15.py

```

x = [1, 2, 3]
y = [7, 8, 9]

def foo(a, b):
    a.append(4)          # Objekt veraendern
    b = [10, 11, 12]     # lokale Variable b referenziert neues Objekt

foo(x, y)
print('x_=', x)
print('y_=', y)

```

mit unveränderlichen Objekten:

listings/v2\_func16.py

```

x = (1, 2, 3)
y = (7, 8, 9)

def foo(a, b):
    # a.append(4)          # Objekt veraendern ist nicht erlaubt
    b = (10, 11, 12)      #lokale Variable b referenziert neues Objekt

foo(x, y)
print('x_=', x)
print('y_=', y)

```

## Teil III

## Lektion 3: Exceptions, Dateien und Strings

## 5 Exceptions

- Fehler (<https://docs.python.org/3/tutorial/errors.html>) können auftreten, z.B.:

listings/v3\_exception1.py

```
int('bla')    => ValueError
5/0           => ZeroDivisionError
a[1000]       => IndexError
10 + 'Fr.'    => TypeError
```

- und führen zu einem Abbruch des Programms
- Fehler können abgefangen werden:

listings/v3\_exception2.py

```
try:
    x = int(input('Zahl eingeben: '))
except:
    print('Falsche Eingabe!')
```

## 5.1 Unspezifische Exceptions abfangen

Nicht empfohlen, da auch Exceptions geschluckt werden, die weitergegeben werden sollten, z.B. KeyboardInterrupt.

listings/v3\_exception3.py

```
eingabe = '10Fr.'
try:
    x = int(eingabe)
except:
    print('Oops! Irgendein Fehler ist aufgetreten.')
```

## 5.2 Abfangen mehrerer Exceptions

listings/v3\_exception4.py

```
eingabe = '10Fr.'
try:
    x = int(eingabe)
    y = 1/x
except ValueError as e:
    print('Oops!' + str(e))
except ZeroDivisionError as e:
    print('Oops!' + str(e))
```

mehrfache Ausnahmen gruppieren:

listings/v3\_exception5.py

```
eingabe = '10Fr.'
try:
    x = int(eingabe)
    y = 1/x
```

```
except (ValueError, ZeroDivisionError):  
    print('Oops! Bitte wiederholen.')
```

### 5.3 else-Teil

listings/v3\_exception6.py

```
try:  
    f = open('datei.txt')  
except IOError:  
    print('Kann Datei nicht oeffnen.')
```

else:

```
    print('Datei schliessen.')
```

```
    f.close()  
print('Ende')
```

### 5.4 finally-Teil

listings/v3\_exception7.py

```
try:  
    welt_rennen()  
finally:  
    print('Dinge, die so oder so gemacht werden muessen.')
```

### 5.5 Exceptions generieren

listings/v3\_exception8.py

```
raise ValueError('Falscher Wert.')
```

## 6 Dateien

### 6.1 Datei öffnen

- Datei mit der open()-Funktion öffnen:

listings/v3\_datei1.py

```
f = open('dokument.txt')           # lesen  
f = open('dokument.txt', 'r')      # lesen  
f = open('dokument.txt', 'w')      # schreiben  
f = open('dokument.txt', 'a')      # anhaengen  
f = open('dokument.txt', 'rb')     # binaer  
f = open('dokument.txt', 'wb')     # binaer
```

- Weitere Parameter findet man in der Hilfe (<https://docs.python.org/3/library/functions.html#open>):

listings/v3\_datei2.py

```
open(file, mode='r', buffering=, encoding=None,  
      errors=None, newline=None, closefd=True,  
      opener=None)
```

### 6.2 Dateien lesen und schreiben

- Datei lesen:

listings/v3\_datei3.py

```
inhalt = f.read()      # gesamte Datei lesen
inhalt = f.read(n)     # n Zeichen lesen
zeilen = f.readlines() # Liste aller Zeilen
```

- Datei schreiben:

listings/v3\_datei4.py

```
f.write('hello')      # String schreiben
f.writelines(['1', '2']) # Liste von Strings
```

- Datei schliessen:

listings/v3\_datei5.py

```
f.close()
```

### 6.2.1 Datei lesen

- mit read()

listings/v3\_datei6.py

```
f = open('mailaenderli.txt')
text = f.read()
f.close()
print(text)
```

- besser mit der with-Anweisung

listings/v3\_datei7.py

```
with open('mailaenderli.txt') as f:
    text = f.read()
print(text)
```

- Variante mit readlines()

listings/v3\_datei8.py

```
with open('mailaenderli.txt') as f:
    zeilen = f.readlines()
print(zeilen)
for zeile in zeilen:
    print(zeile.strip())
```

### 6.2.2 Datei schreiben

listings/v3\_datei9.py

```
personen = ['Alice', 'Bob', 'Charlie']
with open('rangliste.txt', 'w') as f:
    for n, person in enumerate(personen, start=1):
        f.write(str(n) + '. ' + person + '\n')

# Ueberpruefen
with open('rangliste.txt') as f:
    print(f.read())

# Ausgabe: 1. Alice
# Ausgabe: 2. Bob
# Ausgabe: 3. Charlie
```

### 6.2.3 glob

listings/v3\_datei10.py

```
import glob
glob.glob('*.ipynb')
```

### 6.2.4 os.path

listings/v3\_datei11.py

```
import os
full_path = os.path.abspath('mailaenderli.txt')
print(full_path)
# Ausgabe: kompletter Pfad der datei

os.path.isfile(full_path)
# Ausgabe: True

os.path.isdir(full_path)
# Ausgabe: False

os.path.getsize(full_path)
os.path.split(full_path)
os.path.splitext(full_path)
os.path.join('ordner', 'datei.txt')
```

## 7 Strings

### 7.1 Stringformatierung

- Stringformatierung benötigt man um Daten hübsch auszugeben

listings/v3\_strings1.py

Menge	Name	Wert
3	R1	1.50k
7	R2	0.10k
2	R3	22.00k
5	R4	47.00k

- oder systematisch abzuspeichern

listings/v3\_strings2.py

```
Menge , Name , Wert
3 , R1 , 1500
7 , R2 , 100
2 , R3 , 22000
5 , R4 , 47000
```

#### 7.1.1 im C-Stil (à la printf)

listings/v3\_strings3.py

```
spannung = 12.56
strom = 0.5
N = 10
```



```
print('N=_%d, _U=_%f, _I=_%.3f' % (N, spannung, strom))
# Ausgabe: N = 10, U = 12.560000, I = 0.500
print('U=_%g' % spannung) # generelles Format
# Ausgabe: U = 12.56
print('X=_0x%04X, _Y=_0x%04X' % (7, 15)) # hex
# Ausgabe: X = 0x0007, Y = 0x000F
```

### 7.1.2 mit format()

listings/v3\_strings4.py

```
spannung = 12.56
strom = 0.5
'U=_{}, _I=_{}'.format(spannung, strom)
# Ausgabe: 'U = 12.56, I = 0.5'
```

- mit Index:

listings/v3\_strings5.py

```
'U=_[{0}], _I=_[{1}]'.format(spannung, strom)
# Ausgabe: 'U = 12.56, I = 0.5'
```

- mit Index und Format:

listings/v3\_strings6.py

```
'U=_[{0:.2f}], _U=_[{0:.f}]'.format(spannung)
# Ausgabe: 'U = 12.56, U = 12.560000'
```

- links-/rechtsbündig oder zentriert:

listings/v3\_strings7.py

```
'{:>8.2f}'.format(spannung)
# Ausgabe: '    12.56'
'{:<8.2f}'.format(spannung)
# Ausgabe: '12.56   '
'{: ^8.2f}'.format(spannung)
# Ausgabe: ' 12.56  '
```

- mit Schlüsselwortparameter:

listings/v3\_strings8.py

```
'U=_[{u}], _I=_[{i}]'.format(u=spannung, i=strom)
# Ausgabe: 'U = 12.56, I = 0.5'
```

- mit Dictionary:

listings/v3\_strings9.py

```
messung = {'spannung': 24, 'strom': 2.5}
'U=_[{spannung}], _I=_[{strom}]'.format(**messung)
# Ausgabe: 'U = 24, I = 2.5'
```

### 7.1.3 mit Stringlitterale

listings/v3\_strings10.py

```
lokale_variable = 13
f'Wert=_[{lokale_variable:.3f}]'
# Ausgabe: 'Wert = 13.000'
```

### 7.1.4 mit string-Methoden

listings/v3\_strings11.py

```
s = 'Python'
s.center(20, '=')
# Ausgabe: '====Python===='
s.ljust(20, '-')
# Ausgabe: 'Python-----'
s.rjust(20, '*')
# Ausgabe: '*****Python'
'1234'.zfill(20)
# Ausgabe: '00000000000000001234'
```

## 7.2 Alles über Strings

- Unicode-Nummer => Zeichen

listings/v3\_strings12.py

```
chr(65)
# Ausgabe: ('A')
```

- Zeichen => Unicode-Nummer

listings/v3\_strings13.py

```
ord('A')
# Ausgabe: (65)
```

- String => bytes

listings/v3\_strings14.py

```
bin_data = 'A'.encode(utf-8)
print(bin_data)
# Ausgabe: b'A'
bin_data.decode('utf-8')
# Ausgabe: 'A'
```

### 7.2.1 Strings aufspalten

- split()

listings/v3\_strings15.py

```
'Python_ist_eine_Schlange.'.split()
# Ausgabe: ['Python', 'ist', 'eine', 'Schlange.']

csv = '1;2000;30.3;44;505'
csv.split(';')
# Ausgabe: ['1', '2000', '30.3', '44', '505']

csv.split(';', maxsplit=2) # max. zwei Trennungen von links her
# Ausgabe: ['1', '2000', '30.3;44;505']

csv.rsplit(';', maxsplit=2) # max. zwei Trennungen von rechts her
# Ausgabe: ['1;2000;30.3', '44', '505']

'1;2;;;3;4'.split(';')
# Ausgabe: ['1', '2', '', '', '3', '4']
```

- `splitlines()`

listings/v3\_strings16.py

```
csv = '''Dies ist
ein mehrzeiliger
Text.'''
csv.splitlines()
# Ausgabe: ['Dies ist', 'ein mehrzeiliger', 'Text.']
```

### 7.2.2 Strings kombinieren

listings/v3\_strings17.py

```
''.join(['a', 'b', 'c'])
# Ausgabe: 'abc'

','.join(['a', 'b', 'c'])
# Ausgabe: 'a,b,c'
```

### 7.2.3 Suchen von Teilstrings

listings/v3\_strings18.py

```
spruch = '''Wir sollten heute das tun,
von dem wir uns morgen wuenschen
es gestern getan zu haben.'''

'morgen' in spruch
# Ausgabe: True

spruch.find('heute')
# Ausgabe: 12

spruch.count('en')
#Ausgabe: 4
```

### 7.2.4 Ersetzen von Teilstrings

listings/v3\_strings19.py

```
spruch.replace('sollten', 'muessten')
# Ausgabe: 'Wir muessten heute das tun,\nvon dem wir uns morgen wuenschen\nnes
gestern getan zu haben.'
```

### 7.2.5 Strings bereinigen

listings/v3\_strings20.py

```
s = '   Dieser String sollte saubere Enden haben.  \n'
print(s)
# Ausgabe:   Dieser String sollte saubere Enden haben.

s.strip()
# Ausgabe: 'Dieser String sollte saubere Enden haben.'

'Ein Satz ohne Satzzeichen am Schluss?'.rstrip('!?.')
# Ausgabe: 'Ein Satz ohne Satzzeichen am Schluss'
```

### 7.2.6 Klein- und Grossbuchstaben

listings/v3\_strings21.py

```
'Passwort'.lower()
# Ausgabe: 'password'

'Passwort'.upper()
# Ausgabe: 'PASSWORD'
```

### 7.2.7 Strings testen

listings/v3\_strings22.py

```
'255'.isdigit()
# Ausgabe: True

'hallo'.isalpha()
# Ausgabe: True

'Gleis7'.isalnum()
# Ausgabe: True

'klein'.islower()
# Ausgabe: True

'GROSS'.isupper()
# Ausgabe: True

'Haus'.istitle()
# Ausgabe: True
```