Datentypen

Übersicht

Python kennt die folgenden Datentypen:

int - ganze Zahlen
float - gebrochene Zahlen
complex - komplexe Zahlen
bool - Wahrheitswerte
none – nichts
str - Zeichenketten

Datentypen sind Klassen

Datentypen sind in Python als Klassen implementiert.

```
>>> type(42)
<class 'int'>

>>> type(3.14)
<class 'float'>

>>> type('Hallo')
<class 'str'>
>>>
```

Die haben deshalb auch Methoden

```
>>> dir(str)
['__class__', '__name__', 'count', 'endswith',
'find', 'format', 'index', 'isalpha',
'isdigit', 'islower', 'isspace', 'isupper',
'join', 'lower', 'lstrip', 'replace', 'rfind',
'rindex', 'rsplit', 'rstrip', 'split',
'startswith', 'strip', 'upper', '__bases__',
'__dict__', 'center', 'encode', 'partition',
'rpartition', 'splitlines']
>>>
```

Zahlendarstellung

Literale – Als Literale bezeichnet man als Text dargestellte Zahlen

Dezimal – Keine führende 0.	123
Binär – 0b... oder 0B…	0b1001
Oktal – 0o... oder 0O…	0o123
Hexadezimal – 0x... oder 0X…	0x1fe

Datentypen - int

`int` ist die Klasse für ganze Zahlen. Sie enthält alle positiven und negativen ganzen Zahlen.

Die Größe eines `int` ist nicht festgelegt. Sie ist praktisch nur durch die Speichergröße begrenzt.

```
>>> type(42)
<class 'int'>
>>> type(-42)
<class 'int'>
>>> type(0)
<class 'int'>
>>>

>>> 10 ** 100
10000000000000000000000000000000000000000000000000000000000000000000000000000000
>>> 10 ** 1000000
MemoryError: memory allocation failed, allocating 34258 bytes
>>>
```

Datentypen - float

Gleitkommazahlen werden grundsätzlich nur als Dezimalzahlen dargestellt.

Die Trennung zwischen dem ganzzahligen und dem gebrochenen Teil erfolgt in Deutschland durch ein Komma.

International wird dazu der Punkt verwendet.

Dementsprechend verwenden auch Programmiersprachen üblicherweise den „.“, denn sie sind international.

Z.B: **3.14** nicht 3,14.

Datentypen - float

Eine Gleitkommazahl kann als Dezimalbruch oder in Exponentialschreibweise dargestellt werden.

Dezimalbruch
Beispiel: 123.456 (mit Punkt!)

Exponential (wissenschaftlich)
Hier erfolgt die Darstellung als Mantisse und Exponent:

Zahl = Mantisse * Exponent (Basis10)

Die **Mantisse** ist eine Gleitkommazahl mit üblicherweise nur einer Vorkommastelle

Der **Exponent** ist immer ein Vielfaches von 10. Er muss **immer ganzzahlig** sein!
Zur Kennzeichnung dieser Schreibweise dient das **e** oder **E**.

Beispiele: 1.0e3 ⇒ 1000.0
3.14E7 ⇒ 3140000.0
7.35E-3 ⇒ 0.00735

Datentypen - float

Die maximale Größe von Mantisse und Exponent sind begrenzt:

CPython werden Gleitkommazahlen in 64-bit gespeichert.

Die minimale bzw. maximale Größe von `float` in CPython beträgt:

```
Python 3.10.4 (/usr/bin/python3)
>>> import sys
>>> print(sys.float_info)
sys.float_info(
    max=1.7976931348623157e+308,
    max_exp=1024,
    max_10_exp=308,
    min=2.2250738585072014e-308,
    min_exp=-1021,
    min_10_exp=-307,
    dig=15,
    mant_dig=53,
    epsilon=2.220446049250313e-16,
    radix=2,
    rounds=1)
>>>
```

Datentypen - float

Bei **Micropython** sieht es anders aus. Hier spielt der Platzbedarf eine große Rolle. Deshalb ist die Größe von Gleitkommazahlen hier eingeschränkt. **sys.float_info** gibt es im M5Stack-MP leider nicht.
Eine konkrete Angabe zur maximalen Größe habe ich im Internet leider auch nicht finden können. Deshalb habe ich es ausprobiert:

Mantisse: 9.999999 - also auf 6 Nachkommastellen bzw. auf insgesamt 7 Stellen.

Exponent: E-40 ... E38 (bei exp10)

Datentypen - float

Durch die begrenzte Größe kommt es zu Rundungsfehlern.

Wenn insgesamt mehr als 7 Stellen vorhanden sind wird die Fließkommazahl gerundet!

Der Übergang zum Aufrunden liegt aber bei 0.0000006:

Intern wird wohl mit einer 12-stelligen Mantisse gerechnet.

Beim Rechnen mit Fließkommazahlen kommt es gelegentlich zu Rundungsfehlern, weil gebrochene Zahlen intern nicht exakt dargestellt werden können.

```
>>> 10.99999
10.99999
>>> 10.999999
11.0

>>> 9.99999959
9.999999
>>> 9.9999996
10.0
>>>

# Noch präzieser:
>>> 9.99999959999
9.999999
>>> 9.999999599991
10.0
>>>
```

Datentypen - bool

Es gibt nur zwei Wahrheitswerte **Wahr** und **Falsch**.

In Python werden diese durch **True** und **False** dargestellt.

Wahrheitswerte entstehen bei Vergleichen.

Die Funktion **bool()** wandelt Daten in Wahrheitswerte um.

```
>>> 3 > 2
True
>>> 2 > 3
False

>>> bool(0)
False
>>> bool(42)
True
>>> bool(3.14)
True
>>> bool(-1)
True
>>> bool('Hallo')
True
>>>
```

Datentypen - None

None bedeutet **Nichts**.

Eine Variable die **`None`** zurückliefert hat keinen Inhalt.

None wird **nicht** bei einer leeren Variablen, z.B leere Liste, zurückgegeben.

Meist tritt dieser Wert auf, wenn eine Funktion keinen Rückgabewert hat.

```
>>> type(None)
<class 'NoneType'>
>>>
```

```
>>> x = print('Hallo')
Hallo
>>> x
>>>

>>> type(x)
<class 'NoneType'>
>>>
```

Datentypen - str

Strings gehören m.E. zu den Datenstrukturen. Der zugrunde liegende Datentyp Charakter existiert in Python aber nicht.

Strings werden in Python durch einfache oder doppelte Anführungsstriche begrenzt.

Micropython speichert Zeichen als UTF8.

```
>>> "Ich bin ein String"

>>> 'Ich bin auch ein String'

>>> 'Deshalb kann man auch "Dieses" machen!'

'Deshalb kann man auch "Dieses" machen!'

>>>
```

Datentypen – str

Micropython speichert Zeichen als UTF8.

Intern werden Zeichen als Zahl gespeichert. Welche Zahl welches Zeichen repräsentiert lässt ich mit ord() heraus bekommen.

chr() wandelt die Zahl wieder in ein Zeichen.

```
>>> ord('A')
65
```

```
>>> chr(65)

'A'

>>>
```

Datentypen - str

chr() wandelt die Zahl wieder in ein Zeichen.

Es gibt in Python auch lange Strings, die über mehrere Zeilen gehen. Dafür verwendet man dreifache Anführungszeichen. Sowohl ‘ ‘ als auch “ “.

Dreifache Anführungszeichen werden auch für umfangreiche Kommentare benutzt.

```
>>> chr(126)
'~'
>>> chr(127)
'\x7f'
>>>
>>> ''' In Micropython wandelt chr() nur die
Zahlen < 127 in Zeichen um. Größere Zahlen
werden in das Unicode Equivalent umgewandelt.'''
' In Micropython wandelt chr() nur die Zahlen <
127 in Zeichen um. Gr\xf6\xdfere Zahlen werden
in das Unicode Equivalent umgewandelt.\n'
>>> print(''' In Micropython wandelt chr() nur
die Zahlen  <127 in Zeichen um. Größere Zahlen
werden in das Unicode Equivalent umgewandelt.
''')
 In Micropython wandelt chr() nur die Zahlen <
127 in Zeichen um. Größere Zahlen werden in das
Unicode Equivalent umgewandelt.
```

Datentypen - casting

Als **casting** bezeichnet man das Umwandeln eines Datentyp in einen anderen.

int() wandelt in einen Integer

float() wandelt in Fließkomma

str() wandelt in einen String

bool() wandelt in ein Boolean

```
>>> int(3.14)
3

>>> float('3.14')
3.14

>>> float(3)
3.0

>>> str(3.14)
'3.14'
>>>
```