Zahlen, Zeichenketten und Byte-Strings

Was wie eine Zahl aussieht, muss keine sein. Das zeigt schon ein einfaches Beispiel: Zur Eingabe einer Zahl über die Tastatur der Entwicklungsumgebung benutzen wir die `input`-Funktion:

```
z1 = input('Zahl eingeben:')
```

Den Wert von `z1` lassen wir uns mit Hilfe der `print`-Funktion anzeigen:

```
print(z1)
```

Auf den ersten Blick sieht das Ergebnis gut aus. Versuchen wir aber zu `z1` eine Zahl zu addieren, wird dies mit einer Fehlermeldung quittiert. Dies liegt daran, dass die `input`-Funktion jede Eingabe als Zeichenkette (String, kurz: str) ansieht. Der Inhalt von `z1` ist also vom Typ **str**.

Man kann aber eine Zeichenkette, welche als Zahl gedeutet werden kann, in eine Zahl (Number, kurz: **num**) konvertieren. Bei Zeichenketten, die ganze Zahlen repräsentieren, kann man dazu die `int`-Funktion benutzen:

```
z2 = int(z1)
```

`z2` ist vom Typ num. Üblicherweise fasst man beide Befehle zu einem einzigen zusammen:

```
z = int(input('Zahl eingeben:')
```

Neben den Typen num und str gibt es noch den Typ **bytes**. Variablen dieses Typs werden oft als **Byte-Strings** bezeichet; mit ihnen können Zahlen und Zeichenketten dargestellt werden. Sie stellen eine Folge von Bytes dar. Ein Beispiel ist:

```
bs = b'\x05\x41\x46'
```

Hier besteht **bs** aus den Bytes mit den hexadezimalen Werten 0x05, 0x41 und 0x46. Byte-Strings werden durch `b'...'` gekennzeichnet. Die einzelnen Bytes werden hier jeweils mit einem Backslash (\) angegeben: `\x05, \x41 und \x46.`

Gibt es für das Byte allerdings ein darstellbares ASCII-Zeichen, kann es auch durch dieses ASCII-Zeichen angegeben werden; so kann man statt `b'\x05\x41\x46'` auch `b'\x05AF'` schreiben. Tatsächlich greift Micropython bei der Ausgabe mit der `print`-Funktion auf die zweite Darstellungsweise zurück. Dies ist von Vorteil, wenn Texte mit Hilfe von Byte-Strings übertragen werden, zum Beispiel bei Lese- und Schreibvorgängen über die UART-Schnittstelle.

Häufig ist es erforderlich oder sinnvoll, den Datentyp zu wechseln. Im Folgenden stellen wir Beispiele für entsprechende Umwandlungen vor; dabei stehen Zahlen im Mittelpunkt.

1. Byte-String (bytes-Typ) => Zahl im Zehnersystem (num-Typ)

```
data = b'\x05\x41\x46'
print('data', '=', data)
int_value = int.from_bytes(data, 'big')
print(data, '=>' , int_value)
```

Anzeige im Terminal:

```
data = b'\x05AF'
b'\x05AF' => 344390
```

2. Zahl (num-Typ) => Byte-String (bytes-Typ)

```
data = 270
anzahl_bytes = 2
b = data.to_bytes(anzahl_bytes, 'big')
print(data, '=>', b)
```

Anzeige im Terminal:

```
270 => b'\x01\x0e'
```

3. Zahl im Zehnersystem (num-Typ) => Zahl im Hex-System (str-Typ)

```
int_value = 344390
hex_value = hex(int_value)
print(int_value, '=>', hex_value)
```

Anzeige im Terminal:

```
344390 => 0x54146
```

4. Zahl im Hex-System (str-Typ) => Zahl im Zehnersystem (num-Typ)

```
hex_value = 0x54146
int_value = int(hex_value)
print(hex_value, '=>', int_value)
```

Anzeige im Terminal:

```
0x54146 => 344390
```

5. Zahl als Zeichenkette (str-Typ) => Zahl im Zehnerystem (num-Typ)

```
data = "42"
integer_value = int(data)
print(data, '=>' , integer_value)
```

Anzeige im Terminal:
```
42 => 42
```

6. Zahl im Hex-System (num-Typ) => Zahl im Zehnersystem (num-Typ)

```
data = 0xA2      # 10*16+2 = 162
print(data)
```

Anzeige im Terminal:
```
162
```

7. Binärzahl (num-Typ) => Zahl im Zehnersystem (num-Typ)

```
data = 0b11000011     # 12*16+3 = 195
print(data)
```

Anzeige im Terminal:
```
195
```

8. Binärzahl (str-Typ) => Zahl im Zehnersystem (num-Typ)

```
data = '11000011'     # 12*16+3
print(data, '=>', int(data, 2))
```

Anzeige im Terminal:
```
11000011 => 195
```

9. Zahl im Zehnersystem (num-Typ) => Binärzahl (str-Typ)

```
data = 21
b = bin(data)
print(data, '=>', b)
```

Anzeige im Terminal:
```
21 => 0b10101
```

10. Zahl im Zehnersystem (num-Typ) => Binärzahl als Liste (list-Typ)

```
data = 21
b = bin(data)[2:]     # 0b abschneiden
l = []
for d in b:
    l.append(int(d))
print(data, '=>', l)
```

Anzeige im Terminal:

```
21 => [1, 0, 1, 0, 1]
```

11. Binärzahl als Liste (list-Typ) => Zahl im Zehnersystem (num-Typ)

```
data = [1, 0, 0, 1, 1]     # 19
dec = 0
for digit in data:
    # dec = dec * 2 + int(digit) # oder...
    dec = (dec << 1) + int(digit)
print(data, '=>', dec)
```

Anzeige im Terminal:

```
[1, 0, 0, 1, 1] => 19
```