Spezielle Operatoren mit Python

Neuere Sprach-Features kennenlernen

Kombinierte Zuweisungsausdrücke

Mit **`Python 3.8`** wurde der Operator **`:=`** für kombinierte Zuweisungsausdrücke eingeführt.

Er führt auch den `Spitznamen` **`walrus-operator`**, aufgrund der Ähnlichkeit mit den Augen und den Zähnen eines Walrosses.

```
1 # 24-walrus-operator-example.py
2
3 import random
4 random.seed()
5 total = 0
6 while (total := total + random.randint(1,8)) < 30:
7         print("Subtotal:", total)
```

Konstanten mit string

Das Modul **`string`** stellt einige nützliche Zeichenketten-Konstanten bereit, zum Beispiel alle Buchstaben, alle Ziffern oder alle Interpunktionszeichen, wie Sie in folgendem Programm sehen:

```
1 # 24-string-module.py
2
3 import string
4 print("Lower:", string.ascii_lowercase)
5 print("Upper:", string.ascii_uppercase)
6 print("Characters:", string.ascii_letters)
7 print("Numbers:", string.digits)
8 print("Punctuation:", string.punctuation)
```

Verzweigungen mit match

Mit Python 3.10 wurde endlich der Befehl **`match`** hinzugefügt. Ein lange gewünschtes Feature, welches dem switch-Statement in anderen Sprachen sehr ähnlich ist.

```
1  # 24-match.py
2
3  x = "Paris"
4  match x:
5    case "Paris":
6      print("Frankreich")
7    case "Rom":
8      print("Italien")
9    case "Madrid":
10     print("Spanien")
11   case _:
12     print("Unbekanntes Land")
```

Verzweigungen mit match

Natürlich geht das auch mit or-Kombinationen (oder Variante).

```python
# 24-match-or.py

import random
x = random.randint(1,6)
print("x =", x)

match x:
    case 1 | 3 | 5:
        print("uneven")
    case 2 | 4 | 6:
        print("even")
    case _:
        print("no value")
```

Verzweigungen mit match

In Kombination mit **`if`** wird es nochmals ein bisschen raffinierter, denn nun können auch **`Bedingungen`** abgedeckt werden:

```
# 24-match-if.py

x = random.randint(1,10)
print("x * 1.5 =", x * 1.5)
match x * 1.5:
  case x if x < 5:
    print("kleiner Wert")
  case x if x > 11:
    print("großer Wert")
  case _:
    print("mittlerer Wert")
```

Typenhinweise (seit 3.6)

Wir haben ja bereits kurz über Typenhinweise gesprochen, jedoch kann man dies noch weiter treiben:

Wo überall hat es

Typen?

Achtung: Typen haben nur provisorischen Charakter (nicht änderbar / nicht bindend)

```python
# 24-types.py

a: int = 42
b: float = 42.5
c: str = "Hallo"
d: bool = True

print("Variables:", a, b, c, d)


def average(x: float, y: float) -> float:
    result = (x + y) / 2
    return result


print("Average:", average(3.4, 9.4))
```

Typenhinweise (seit 3.6)

Wir haben ja bereits kurz über Typenhinweise gesprochen, jedoch kann man dies noch weiter treiben:

Wo überall hat es

Typen?

Achtung: Typen haben nur provisorischen Charakter (nicht änderbar / nicht bindend)

```python
# 24-types.py

a: int = 42
b: float = 42.5
c: str = "Hallo"
d: bool = True

print("Variables:", a, b, c, d)


def average(x: float, y: float) -> float:
    result = (x + y) / 2
    return result


print("Average:", average(3.4, 9.4))
```

Ende

Das war alles für dieses Kapitel