In [1]:
```
meine_katze = 2.0
```

In [2]:
```
type(meine_katze)
```

Out[2]:
```
float
```

In [4]:
```
meine_katze=['Polly', 'Luna']
```

In [5]:
```
meine_katze
```

Out[5]:
```
['Polly', 'Luna']
```

In [6]:
```
type(meine_katze)
```

Out[6]:
```
list
```

In [7]:
```
a=5
```

In [8]:
```
a
```

Out[8]:
```
5
```

In [9]:
```
a=10
```

In [10]:
```
a+a
```

Out[10]:
```
20
```

In [11]:
```
a +=10
```

In [12]:
```
a
```

Out[12]:
```
20
```

In [13]:
```
#Variablentyp bestimmen
type(a)
```

Out[13]:
```
int
```

In [14]:
```
a=(1,2)
```

In [15]:
```
type(a)
```

Out[15]:
```
tuple
```

Dynamic Typing

Python verwendet *dynamic typing* (dynamische Typisierung), was bedeutet, dass Sie einer einzigen Variablen verschiedene Datentypen zuweisen können. Dadurch ist Python sehr flexibel in der Handhabung von Datentypen, im Gegensatz zu anderen Sprachen mit *static typing* (statischer Typisierung).

Wer Erfahrungen mit C, C++, Java oder ähnlichen Sprachen hat, hat dort gelernt, dass man einer Variablen einen Typ zuordnen muss, bevor man sie verwenden darf. Der Datentyp muss im Programm festgelegt werden – und zwar, bevor die Variable zum ersten Mal benutzt oder definiert wird.

Dies sieht dann beispielsweise in einem C-Programm so aus:

int i, j; float x; x = i / 3.0 + 5.8;

Während des Programmlaufs können sich dann die Werte für i, j und x ändern, aber ihre Typen sind für die Dauer des Programmlaufs auf int im Falle von i und j und float für die Variable x festgelegt. Dies bezeichnet man als „statische Typdeklaration", da bereits der Compiler den Typ festlegt und während des Programmablaufs keine Änderungen des Typs mehr vorgenommen werden können.

In Python sieht dies anders aus. Zunächst einmal bezeichnen Variablen in Python keinen bestimmten Typ, und deshalb benötigt man in Python keine Typdeklaration.

Benötigt man im Programm beispielsweise eine Variable i mit dem Wert 42 und dem Typ Integer, so erreicht man dies einfach mit der Anweisung „i = 42". Damit hat man automatisch ein Objekt vom Typ Integer angelegt, welches dann von der Variablen i referenziert wird.

Man kann dann im weiteren Verlauf des Programms der Variablen i auch andere Objekte mit anderen Datentypen zuweisen:

In [ ]:
```
i = 42
print (i)

i = "Hallo"
print (i)

i = [3, 9, 17]
print (i)
```

In [ ]:
```
# Datentyp ermitteln
i = 42
```

In [ ]:
```
type(i)
```

In [ ]:
```
# Datentyp ermitteln
i = "Hallo"
```

In [ ]:
```
type(i)
```

In [ ]:
```
# Datentyp ermitteln
i = [3, 9, 17]
```

In [ ]:
```
type(i)
```

Während sich bei statischen Typdeklarationen, also bei Sprachen wie C und C++, nur der Wert, aber nicht der Typ einer Variablen während eines Laufs ändert, kann sich bei dynamischer Typdeklaration, also in Python, sowohl der Wert als auch der Typ einer Variablen ändern.

Wann kann es jetzt zu einem Fehler kommen?

Python achtet auf Typverletzungen zur Laufzeit eines Programms. Man spricht von einer Typverletzung, wenn Datentypen beispielsweise aufgrund fehlender Zuweisungskompatibilität nicht regelgemäß verwendet werden. In Python kann es nur bei Ausdrücken zu Problemen kommen, da man ja einer Variablen einen beliebigen Typ zuordnen kann. Eine Typverletzung liegt beispielsweise vor, wenn man eine Variable vom Typ int zu einer Variablen vom Typ str addieren will. Es wird in diesem Fall ein TypeError generiert.

In [ ]:
```
x = "Ich bin ein String"
y = 42
z = x + y
```

Allerdings könnte man Integer- und FloatWerte – auch wenn es sich um unterschiedliche Datentypen handelt – in einem Ausdruck mischen. Der Wert des Ausdrucks wird dann ein Float.

In [ ]:
```
x = 12
y = 3.5
z = x * y
z
```

In [ ]:
```
print (type(x))
print (type(y))
print (type(z))
```

Typumwandlung

In der Programmierung bezeichnet man die Umwandlung eines Datentyps in einen anderen als Typumwandlung. Man benötigt Typumwandlungen beispielsweise, wenn man Strings und numerische Werte mit dem Stringoperator „+" verketten möchte

In [ ]:
```
Vorname = "Hansi"
Nachname = "Flick"
Alter = 57

# Fehlermeldung
print(Vorname + " " + Nachname + ":  " + Alter)
```

In [ ]:
```
Vorname = "Hansi"
Nachname = "Flick"
Alter = 57

# Fehlermeldung
print(Vorname + " " + Nachname + ":  " + str(Alter))
```

Mit der Funktion type() können wir den Datentyp ermitteln. Als Alternative zur Funktion type() gibt es noch die eingebaute Funktion isinstance(object, ct), die einen Wahrheitswert „True" oder „False" zurückgibt.

object ist das Objekt, welches geprüft werden soll, und ct entspricht der Klasse oder dem Typ, auf den geprüft werden soll. Im folgenden Beispiel prüfen wir, ob es sich bei dem Objekt s um ein String handelt.

In [ ]:
```
s = "Ich bin ein String"
isinstance(s, str)
```

In Programmen kommt es häufig vor, dass wir für ein Objekt wissen wollen, ob es sich um einen von vielen Typen handelt. Also zum Beispiel die Frage, ob eine Variable ein Integer oder ein Float ist. Dies kann man mit der Verknüpfung "or" lösen:

In [ ]:
```
x = 4
isinstance(x, int) or isinstance(x, float)
```

In [ ]:
```
x = 4.8
isinstance(x, int) or isinstance(x, float)
```

Übungsaufgaben

1. Schreiben Sie ein Programm, welches an der Konsole ihr Jahreseinkommen abfragt. Als Ausgabe sollen die zu zahlenden Steuern bei einer Steuerrate von 10% ausgegeben werden.