# Kapitel 3: Grundlagen der Programmierung

## 3.4. Übungsaufgaben

### Aufgabe 1

Nachfolgend wird an mehreren Beispielen die Deklaration, Initialisierung, sowie Neu- und Wiederbelegung einer Variable gezeigt.

**Eure Aufgabe ist es, den Wert und den Datentyp der benannten Variable am Ende des kleinen Programms zu bestimmen.**

Versucht zunächst, dies **ohne** Ausführung des Codes zu lösen, also nur durch zeilenweises Nachvollziehen des Programms.

Notiert eure Lösungen immer da, wo steht: `... deine Antwort hier ...`. Ändern könnt ihr die Texte durch doppeltes Hineinklicken.

In [1]:
```
number = 1
number = number + 5
number = number * 2
number = number % 12
print(number, type(number))
```

```
0 <class 'int'>
```

**Welchen Wert und Datentyp hat die Variable `number`?**

- Wert: `0`
- Datentyp: `int`

---

In [2]:
```
number = '1.23'
number = float(number)
text = int(number)
neuer_text = text * 5
neuer_text = str(neuer_text)

print(neuer_text, type(neuer_text))
```

```
5 <class 'str'>
```

**Welchen Wert und Datentyp hat die Variable `neuerText`?**

- Wert: `5`
- Datentyp: `str`

---

In [3]:
```
irgendwas = 'Hallo Schüler(in)'
wie_gehts = ' - wie geht es dir heute?'
text = wie_gehts + irgendwas
text = text * 2
text = 2
text = irgendwas * text

print(text, type(text))
```

```
Hallo Schüler(in)Hallo Schüler(in) <class 'str'>
```

**Welchen Wert und Datentyp hat die Variable `text`?**

- Wert: `Hallo Schüler(in)Hallo Schüler(in)`
- Datentyp: `str`

---

In [4]:
```
nutzereingabe = 2
nutzereingabe = nutzereingabe**2
nutzereingabe = nutzereingabe // 3
nutzereingabe = nutzereingabe * 10
ausgabe = 'Das Ergebnis ist: ' + str(nutzereingabe)

print(ausgabe, type(ausgabe))
```

```
Das Ergebnis ist: 10 <class 'str'>
```

**Welchen Wert und Datentyp hat die Variable `ausgabe`?**

- Wert: `Das Ergebnis ist: 10`
- Datentyp: `str`

## Aufgabe 2

In der untenstehenden Tabelle sind diverse Werte vorgegeben.

Anhand dieser Werte sollt ihr ausprobieren, wie sich verschiedene Datentypisierungen auswirken. Dazu könnt ihr die folgende Programmstruktur als Hilfsmittel nutzen.

In [18]:
```
wert_vorher = None  # <-- Spalte 1 der Tabelle
print(wert_vorher, type(wert_vorher))

print('---')

wert_nachher = str(wert_vorher) # <-- an der Stelle muss immer die Typisierungsfunk
print(wert_nachher, type(wert_nachher))
```

```
None <class 'NoneType'>
---
None <class 'str'>
```

**Zur Verfahrensweise:**

- Ihr beginnt in der ersten Zeile der Tabelle. Dort steht in der Spalte `Wert (vorher)` der Wert `1.2`. Im oberen Programm weist ihr der Variable `wert_vorher` genau diesen Wert zu.
- Weiterhin beinhaltet die Tabelle die gewünschte Typisierungsfunktion. Diese wird in der vierten Codezeile im obigen Codeblock verwendet (der Kommentar gibt einen Hinweis).
- Führt ihr das Programm aus, so erhaltet ihr den Wert und Datentyp **vor der Änderung**, im Anschluss den Wert und Datentyp **nach der Typisierung**.
- Tragt eure Ergebnisse in die Tabelle ein. Sollte es zu Besonderheiten kommen, wie direkt in der ersten Zeile der Tabelle, ist dies in der letzten Spalte einzutragen.
- Veränderungen in der Tabelle könnt ihr wieder vornehmen, indem ihr doppelt auf die Tabelle klickt.

| Wert (vorher) | Datentyp (vorher) | Typisierungsfunktion | Wert (danach) | Datentyp (danach) | Besonderheit(en) |
|---|---|---|---|---|---|
| 1.9 | float | int() | 1 | int | Nachkommastellen wurden abgeschnitten. Es wird nicht gerundet. |
| 42 | int | int() | 42 | int | - |
| 42 | int | float() | 42.0 | float | Eine Nachkommastelle wird hinzugefügt. |
| 1.2 | float | float() | 1.2 | float | - |
| 42 | int | str() | '42' | str | - |
| 1.2 | float | str() | '1.2' | str | - |
| '12.34' | str | int() | X | X | Direkte Konvertierung einer Kommazahl in einer Zeichenkette in eine ganze Zahl ist **nicht** möglich. |
| '12.34' | str | float() | 12.34 | float | - |
| '112' | str | int() | 112 | int | - |
| '112' | str | float() | 112.0 | float | Eine Nachkommastelle wird hinzugefügt. |
| '1' | str | bool() | True | bool | - |
| '0' | str | bool() | True | bool | - |
| True | bool | int() | 1 | int | - |
| False | bool | int() | 0 | int | - |
| True | bool | float() | 1.0 | float | - |
| False | bool | float() | 0.0 | float | - |
| None | none | str() | 'None' | str | ? |
| None | none | bool() | False | bool | ? |
| None | none | int() | X | X | Führt zu einem Fehler. "Nichts" entspricht **keiner** ganzen Zahl. |

## Aufgabe 3

**Implementiere ein Programm, was die folgenden Funktionalitäten erfüllt:**

- Die Nutzerin oder der Nutzer soll aufgefordert werden, eine Zahl einzugeben.
- Diese Eingabe soll zunächst in einer Variable gespeichert werden.
- Aus der Eingabe soll deren Quadratzahl berechnet werden.
- Ausgegeben werden soll in etwa sowas wie: 'Die Quadratzahl deiner eingegebenen Zahl `ZAHL` ist `ERGEBNIS`.' Die Platzhalter sollen durch die tatsächlich eingegebenen und berechneten Werte ersetzt werden.

In [14]:
```
eingabe = input('Gib eine ganze Zahl ein: ')
zahl = float(eingabe)
quadrat = zahl**2
print(f'Die Quadratzahl deiner eingegebenen Zahl {zahl} ist {quadrat}.')
```

```
Die Quadratzahl deiner eingegebenen Zahl 5.0 ist 25.0.
```

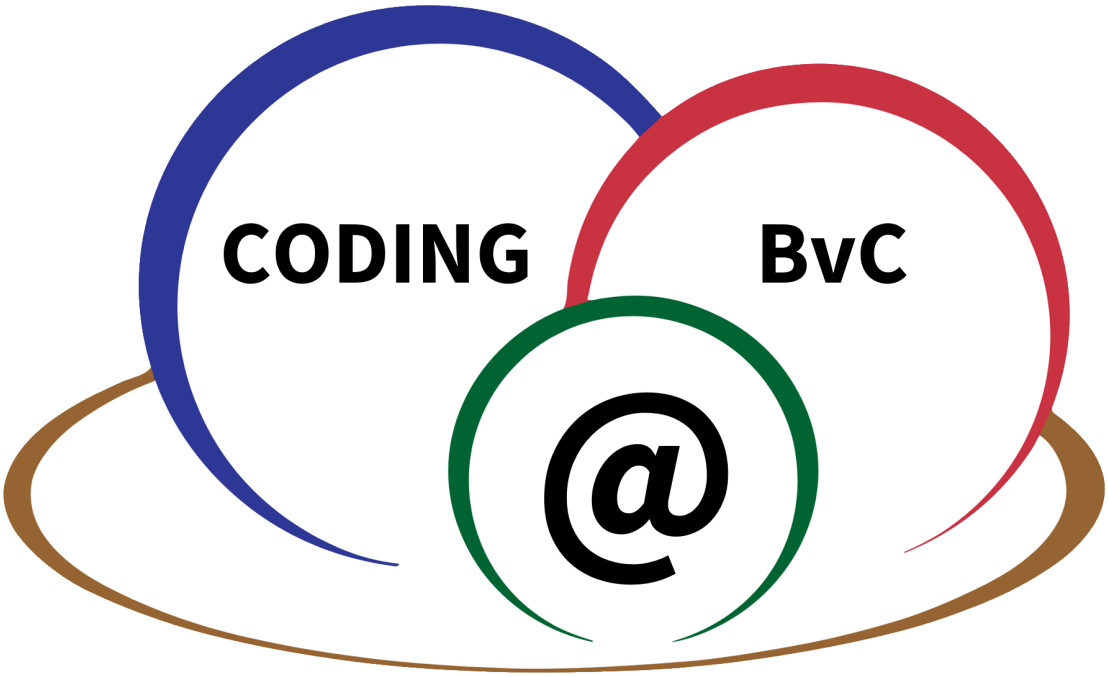
## Aufgabe 4

**Implementiere einen kleinen Taschenrechner für die Grundrechenarten:** Dazu soll der Nutzer zwei Zahlen $a$ und $b$ eingeben. Das Programm gibt dann die Summe $a + b$, die Differenz $a - b$, das Produkt $a \cdot b$ und den Quotienten $a : b$ aus.

In [17]:
```
a = input('Gib die erste Zahl a ein: ')
b = input('Gib die erste Zahl b ein: ')

a = float(a)
b = float(b)

sum = a + b
diff = a - b
prod = a * b
quot = a / b

print(f'{a} + {b} = {sum}')
print(f'{a} - {b} = {diff}')
print(f'{a} * {b} = {prod}')
print(f'{a} / {b} = {quot}')

# ODER: alles in allem, wie nachfolgend dargestellt
# dabei erzeugt \t einen Tabulator und \n einen Zeilenumbruch (nur Zusatz)

print(f'\n{a}\t+\t{b}\t=\t{sum}\n{a}\t-\t{b}\t=\t{diff}\n{a}\t*\t{b}\t=\t{prod}\n{a
```

```
5.0 + 6.0 = 11.0
5.0 - 6.0 = -1.0
5.0 * 6.0 = 30.0
5.0 / 6.0 = 0.8333333333333334

5.0     +       6.0     =       11.0
5.0     -       6.0     =       -1.0
5.0     *       6.0     =       30.0
5.0     /       6.0     =       0.8333333333333334
```


---


CODING
BvC
@