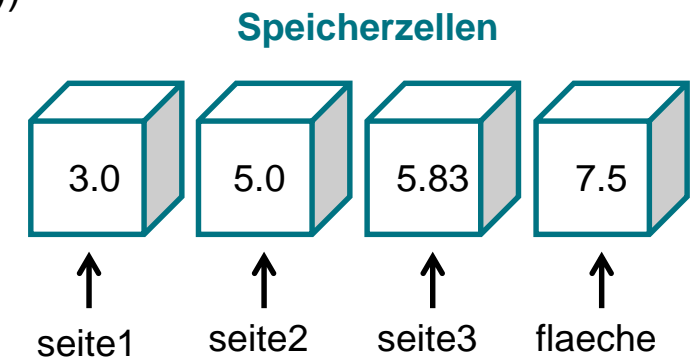


- Zahlen und Zeichenketten (bzw. allgemein: Daten) werden im Speicher des Rechners abgespeichert. **Variablen** kann man sich vereinfacht als Namen der verwendeten Speicherzellen vorstellen.

- Beispiel:

```
import math
seite1 = float(input("Eine Dreiecksseite: "))
seite2 = float(input("Die zweite Dreiecksseite: "))
seite3 = math.sqrt(seite1*seite1 + seite2*seite2)
flaeche = seite1 * seite2 / 2
print("Laenge der Hypothenuse: " + str(seite3) + "\nFlaeche: " + str(flaeche))
```

Es werden vier Variablen verwendet (in der Abb. mit Beispielwerten).



- Im Beispiel wird in der Zeile

```
flaeche = seite1 * seite2 / 2
```

die Fläche berechnet und durch den = Operator der Variablen flaeche zugewiesen. Der berechnete Wert wird dabei im Speicher abgespeichert. Diese Operation heißt **Zuweisung**. Man spricht auch von einem **schreibenden Zugriff** auf die Variable.

- Mit dem Namen der Variable kann man auf den gespeicherten Wert zugreifen (**lesender Zugriff**).
Im Beispiel werden in der Zeile

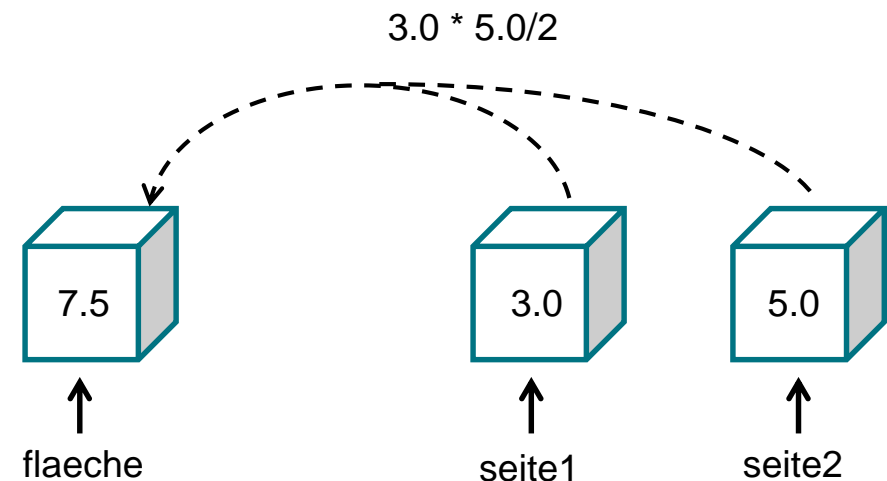
```
print("Laenge der Hypothenuse: " + str(seite3) + "\nFlaeche: " + str(flaeche))
```

auf flaeche lesend zugegriffen und der Wert ausgegeben.

- Die Sprechweise ist, dass eine Variable auf einen Wert im Speicher zeigt oder dass eine Variable eine **Referenz** darauf ist.

- Wir veranschaulichen Variablen durch einen Container für Werte mit einer Referenz darauf (dargestellt als Pfeil).
- Steht eine Variable auf der linken Seite einer Zuweisung (=), dann bedeutet das einen schreibenden Zugriff. Steht eine Variable auf der rechten Seite, so wird lesend auf die Variable zugegriffen. Beispiel:

$$\text{flaeche} = \text{seite1} * \text{seite2} / 2$$



- Verwenden Sie eine Variable nie lesend, bevor die Variable einen Wert erhalten hat!
- Beachten Sie, dass der Operator = keine mathematische Gleichheit bedeutet. Beispiel:

$$x = x + 1$$

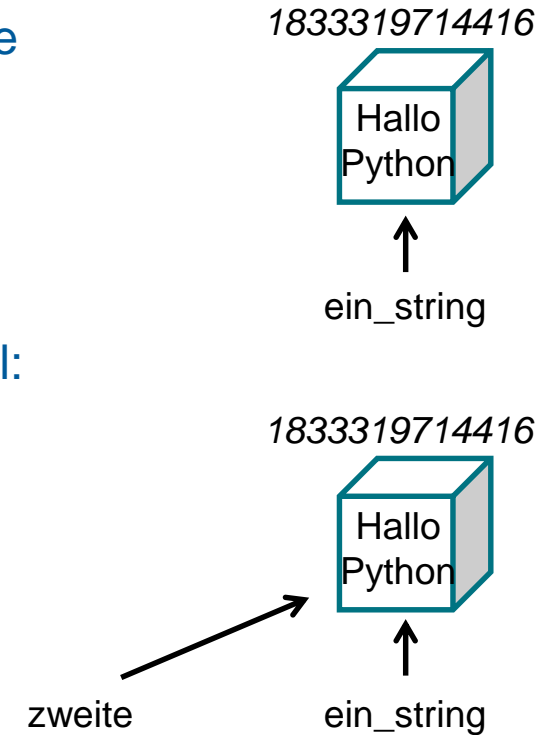
Der Wert $x + 1$ wird der Variablen x zugewiesen. D.h. der Wert von x wird um 1 erhöht.

- Mit der Funktion `id` kann die Identität der Speicherzelle, auf die eine Variable zeigt, herausgefunden werden. Beispiel:

```
ein_string = "Hallo Python"  
print(id(ein_string))
```

- Auf einen Wert kann es auch mehrere Referenzen geben. Beispiel:

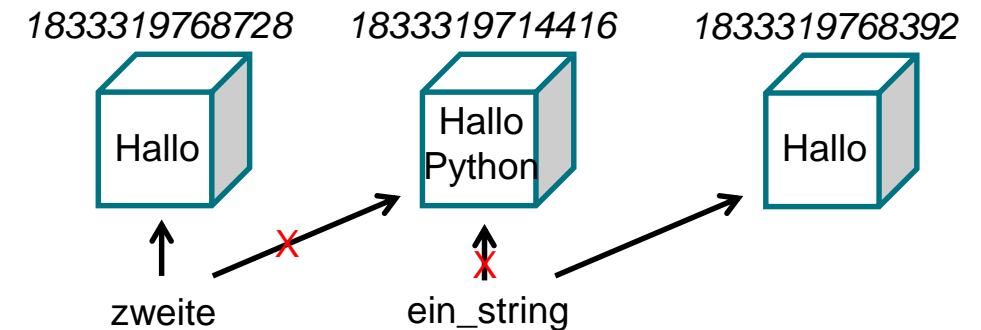
```
zweite = ein_string  
print(str(id(ein_string)) + ", " + str(id(zweite)))
```



- Referenzen können auch „wiederverwendet“ werden, um auf einen anderen Wert zu zeigen. Beispiel:

```
ein_string = ein_string[0:4]
print(str(id(ein_string)) + ", " + str(id(zweite)))
zweite = "Hallo"
print(str(id(ein_string)) + ", " + str(id(zweite)))
```

- Beachten Sie, dass bei der Zuweisung `ein_string = ein_string[0:4]` der alte Wert "Hallo Python" nicht überschrieben wird! Er bleibt im Speicher und könnte mit einer zweiten Referenz weiterverwendet werden.



- Jetzt könnte man annehmen, dass immer mehr Speicher verbraucht wird, wenn Variablen nicht überschrieben werden. Ein intelligentes Speichermanagement verhindert dies aber.
- In Python werden die Werte von ganzen Zahlen, Fließkommazahlen und Zeichenketten bei der Zuweisung nicht überschrieben. Das kann in anderen Programmiersprachen anders sein.

- Nicht nur der Wert, auf die eine Variable zeigt, kann sich ändern sondern auch der Datentyp. Beispiel:

```
x = 15  
print(x)  
x = "text"  
print(x)
```

- Man spricht von **dynamischer Typisierung**.
 - Jede Variable hat zu jedem Zeitpunkt einen eindeutigen Datentyp. Der Datentyp kann sich aber im Laufe der Zeit ändern.
 - In vielen anderen Programmiersprachen ist dies nicht erlaubt (statische Typisierung)!
 - Die dynamische Typisierung ist flexibler, aber leider auch fehleranfälliger.

- Um den aktuellen Datentyp einer Variablen herauszufinden, kann die Funktion `type` verwendet werden.
Beispiel:


```
x = "text"
print(type(x))
```

Ausgabe: <class 'str'>

- Man auch abfragen, ob eine Variable von einem bestimmten Typ ist. Dazu gibt es die Funktion `isinstance`.
Beispiel:

```
zahl = 1
print(isinstance(zahl, int))
```

Ausgabe: True

- Regeln für Variablennamen:
 - Beginnen mit einem Buchstaben oder einem Unterstrich
 - Für die folgenden Zeichen sind Buchstaben, Ziffern und Unterstrich erlaubt
 - Groß- und Kleinschreibung wird unterschieden
 - Müssen verschieden von reservierten Worten, z.B. if, while, return, import, ..., sein
 - Buchstaben sollen klein geschrieben werden (Konvention)
 - Bestandteile sollen mit Unterstrich getrennt werden (Konvention) z.B. anzahl_teilnehmer, button_inc_velocity
 - Sollen die Bedeutung der Variable erkennen lassen (guter Programmierstil)
 - Sollen einheitlich in deutsch oder englisch oder ... sein (guter Programmierstil)
-  Kapitel 4.1 – 4.2, 4.4 – 4.6 in (Klein 2018)



Erstellen Sie Skripte für die folgenden Aufgabenstellungen. Die Eingabe soll wieder mit input und die Ausgabe mit print erfolgen.

1. Eingabe: Zeichenkette wort bestehend aus drei Zeichen

Ausgabe: Folgende Konsolausgabe am Beispiel von wort = thi

```
thi
tthii
itthiit
titthiiti
ititthiitit
```

2. Eingabe: Ganze Zahl w

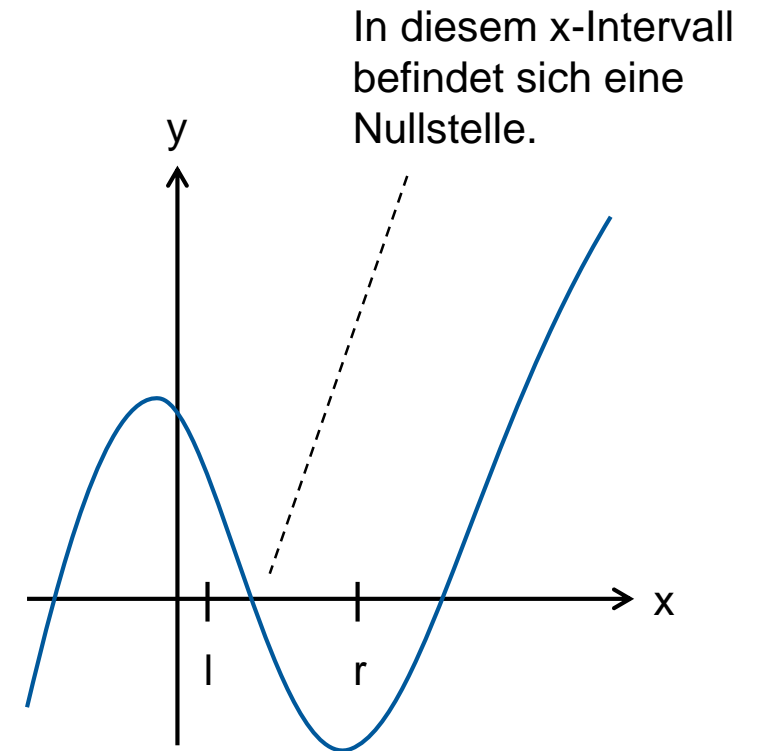
Ausgabe:

- Geben Sie den Wert, die Identität (id) und den Datentyp (type) von w aus.
- Wandeln Sie w in ein Fließkommazahl um: `v = float(w)`. Geben Sie Wert, Identität und Typ von v aus.
- Wandeln Sie w in ein Fließkommazahl um: `v = str(w)`. Geben Sie Wert, Identität und Typ von v aus.

Beispiel Nullstellenbestimmung (1)



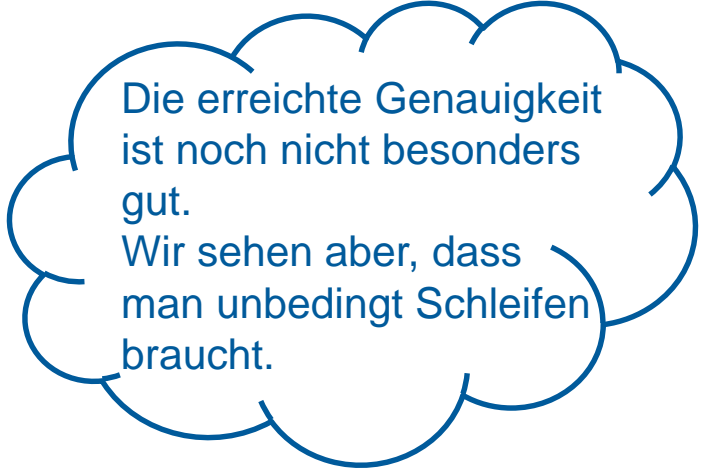
- Jetzt haben wir genügend Python-Kenntnisse, um ein Anwendungsproblem zu lösen. Es soll eine Nullstelle eines Polynoms berechnet werden. In diesem Kapitel führen wir die Nullstellenberechnung in der Python-Shell durch. Später erstellen wir ein Skript dafür.
- Wir verwendet ein Verfahren, das das Intervall, in der sich die Nullstelle befindet, solange halbiert, bis die gewünschte Genauigkeit erreicht ist.
- Als Beispielpolynom nehmen wir $x^3 - 1.8x^2 - 1.2x + 1.6$. Das Startintervall ist $[0.0 ; 1.5]$.



■ In der Shell:

```
>>> l = 0
>>> r = 1.5
>>> x = l
>>> x ** 3 - 1.8 * x ** 2 - 1.2 * x + 1.6
1.6
>>> x = r
>>> x ** 3 - 1.8 * x ** 2 - 1.2 * x + 1.6
-0.87500000000000004
>>> x = (l + r) / 2
>>> x ** 3 - 1.8 * x ** 2 - 1.2 * x + 1.6
0.109375
>>> l = x
```

```
>>> x = (l + r) / 2
>>> x ** 3 - 1.8 * x ** 2 - 1.2 * x + 1.6
-0.60429687499999997
>>> r = x
>>> x = (l + r) / 2
>>> x ** 3 - 1.8 * x ** 2 - 1.2 * x + 1.6
-0.2830566406249999
>>> r = x
>>> x = (l + r) / 2
>>> x ** 3 - 1.8 * x ** 2 - 1.2 * x + 1.6
-0.09326782226562491
>>> x
0.84375
```



Die erreichte Genauigkeit ist noch nicht besonders gut. Wir sehen aber, dass man unbedingt Schleifen braucht.