1 Elemente des Programmierens

Ein Programm zu schreiben ist nicht schwieriger,
als einen Aufsatz zu schreiben.

Wir schauen uns die Grundbausteine von Programmen der Programmiersprache Python an und starten mit dem Programmieren.

1. Elemente des Programmierens
1.2 Grundlegende Datentypen
1.3 Verzweigungen und Schleifen
1.4 Arrays
1.5 Ein- und Ausgabe
1.6 Dictionaries und Abschluss-Beispiel Page Rank

Vorlesung 1

1. Elemente des Programmierens
1.2 Grundlegende Datentypen
Das erste Programm
Datentyp `int` – ganze Zahlen
Datentyp `str` – Texte
Datentyp `bool` – Wahrheitswerte
Datentyp `float` – Dezimalbrüche
1.3 Verzweigungen und Schleifen
1.4 Arrays
1.5 Ein- und Ausgabe
1.6 Dictionaries und Abschluss-Beispiel Page Rank

1.2 Grundlegende Datentypen

Programme dienen der Verarbeitung von Daten.

Die elementaren Arten von Daten für die Programmiersprache Python sind

- Zahlen,
- Texte und
- Wahrheitswerte.

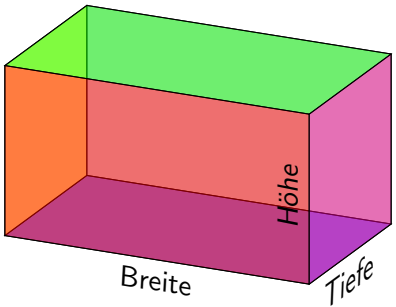
Zu ihrer Verarbeitung sind verschiedene Grund-Operationen verfügbar
(und sie werden intern unterschiedlich behandelt).

1 Der erste Programmier-Auftrag

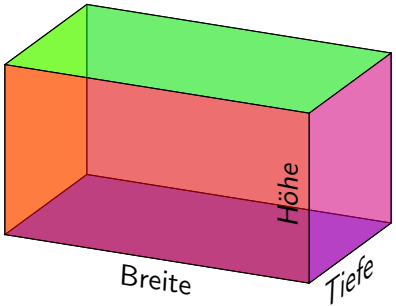
Berechne das Volumen und die Oberfläche eines Quaders

mit Höhe 5, Breite 12 und Tiefe 32 (die Maßeinheiten sind egal).

Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders

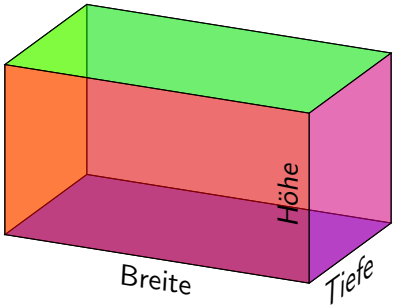


Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders



Volumen = Breite · Höhe · Tiefe (es werden ganze Zahlen benutzt)

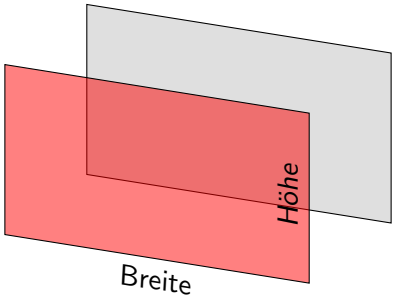
Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders



Volumen = Breite · Höhe · Tiefe (es werden ganze Zahlen benutzt)

Oberfläche =

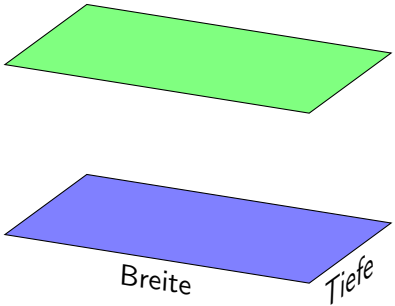
Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders



Volumen = Breite · Höhe · Tiefe (es werden ganze Zahlen benutzt)

Oberfläche = 2 · Breite · Höhe

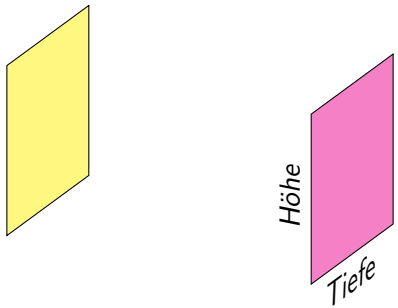
Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders



Volumen = Breite · Höhe · Tiefe (es werden ganze Zahlen benutzt)

Oberfläche = 2 · Breite · Höhe + 2 · Breite · Tiefe

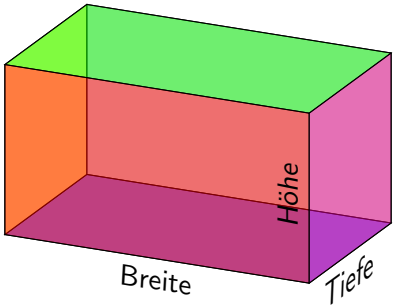
Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders



Volumen = Breite · Höhe · Tiefe (es werden ganze Zahlen benutzt)

Oberfläche = 2 · Breite · Höhe + 2 · Breite · Tiefe + 2 · Höhe · Tiefe

Die Ideen zur Berechnung von Volumen und Oberfläche eines Quaders



Volumen = Breite · Höhe · Tiefe (es werden ganze Zahlen benutzt)

$$\begin{aligned} \text{Oberfläche} &= 2 \cdot \text{Breite} \cdot \text{Höhe} + 2 \cdot \text{Breite} \cdot \text{Tiefe} + 2 \cdot \text{Höhe} \cdot \text{Tiefe} \\ &= 2 \cdot (\ \text{Breite} \cdot \text{Höhe} + \text{Breite} \cdot \text{Tiefe} + \text{Höhe} \cdot \text{Tiefe}\ ) \end{aligned}$$

Der grobe Ablauf des Programms

1. Gib die Maße des Quaders an.

2. Berechne das Volumen und die Oberfläche des Quaders mit den angegebenen Formeln.

3. Gib die berechneten Werte aus.

Der grobe Ablauf des Programms

1. Gib die Maße des Quaders an.
   Höhe = 5, Breite = 12, Tiefe = 32

2. Berechne das Volumen und die Oberfläche des Quaders mit den angegebenen Formeln.

3. Gib die berechneten Werte aus.

Der grobe Ablauf des Programms

1. Gib die Maße des Quaders an.
   Höhe = 5, Breite = 12, Tiefe = 32

2. Berechne das Volumen und die Oberfläche des Quaders mit den angegebenen Formeln.
   Volumen = Breite · Höhe · Tiefe
   Oberfläche = 2 · (Breite · Höhe + Breite · Tiefe + Höhe · Tiefe)

3. Gib die berechneten Werte aus.

Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

Kommentar beginnt mit „#“. Von dort ab wird die Zeile ignoriert (. . .)

Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

---

hoehe, breite, tiefe, oberflaeche und volumen sind Variablen.

Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

`hoehe = ...` weist der Variablen `hoehe` den Wert von `...` zu.

Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

Steht eine Variable nicht links von =, dann wird ihr Wert benutzt.

Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

Das Programm rechnet mit ganzen Zahlen und benutzt die Operatoren + und * .

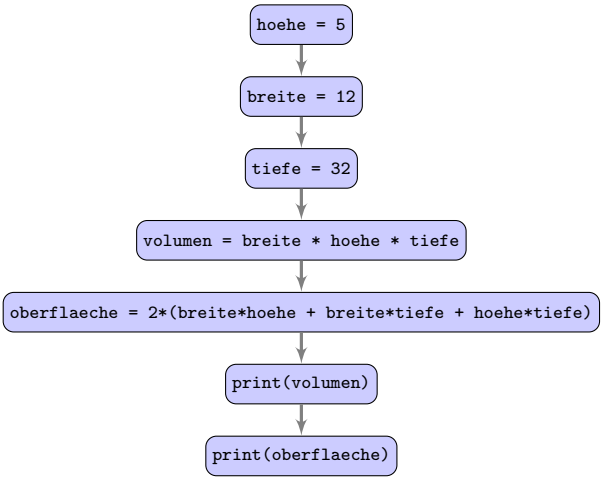
Ein erstes Python-Programm

```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = breite * hoehe * tiefe
oberflaeche = 2 * ( breite*hoehe + breite*tiefe + hoehe*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)
```

Konvention: Variablennamen beginnen mit kleinen Buchstaben.

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

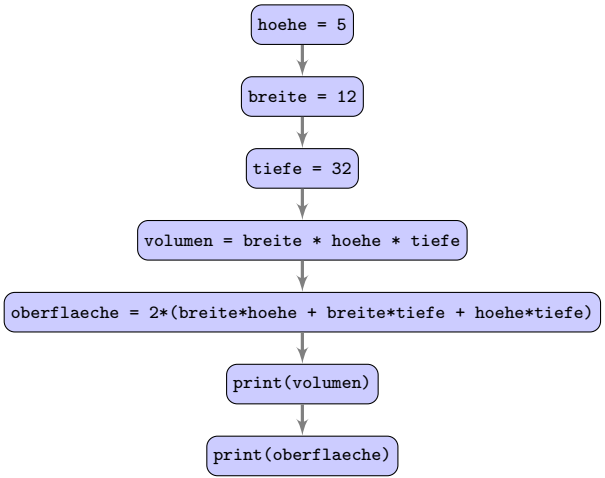
Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



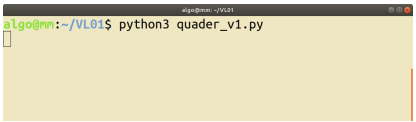
Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

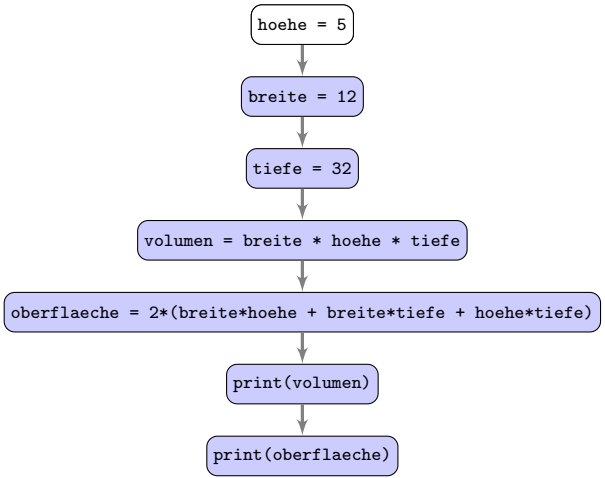


Einfache Vorstellung vom Speicher:

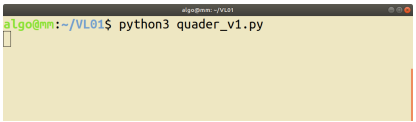
Variable	Wert
`hoehe`	
`breite`	
`tiefe`	
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

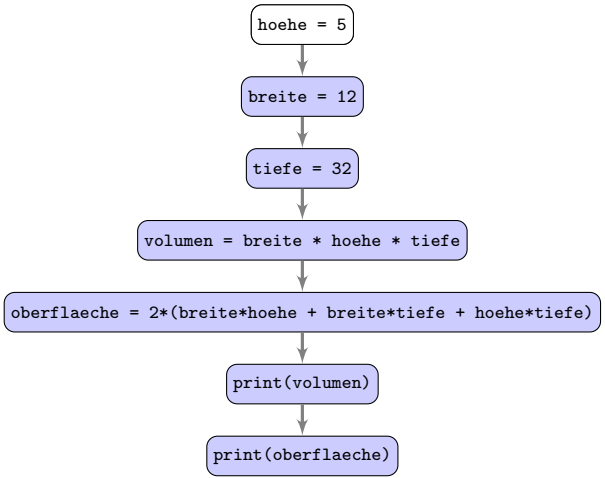


Einfache Vorstellung vom Speicher:

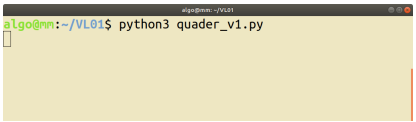
Variable	Wert
`hoehe`	
`breite`	
`tiefe`	
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

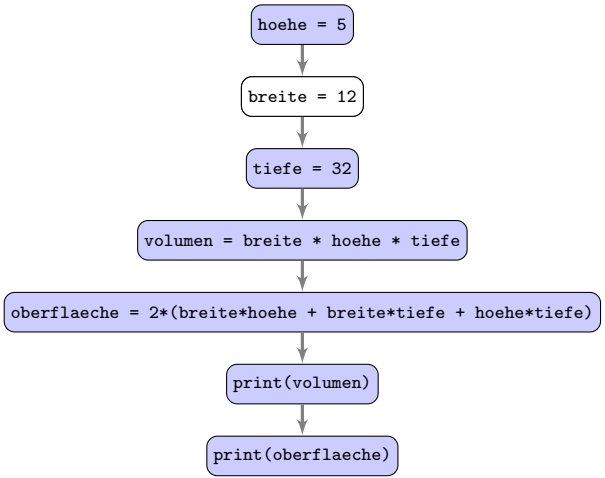


Einfache Vorstellung vom Speicher:

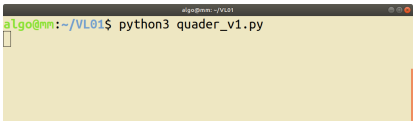
Variable	Wert
`hoehe`	5
`breite`	
`tiefe`	
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

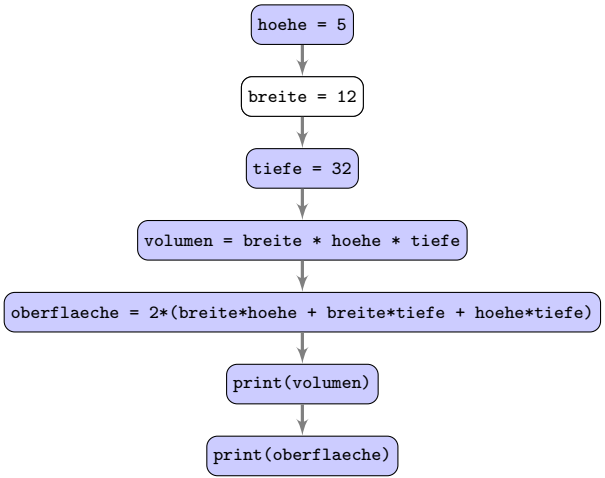


Einfache Vorstellung vom Speicher:

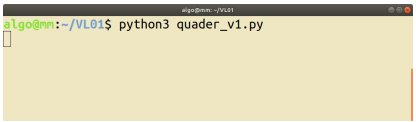
Variable	Wert
`hoehe`	5
`breite`	
`tiefe`	
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

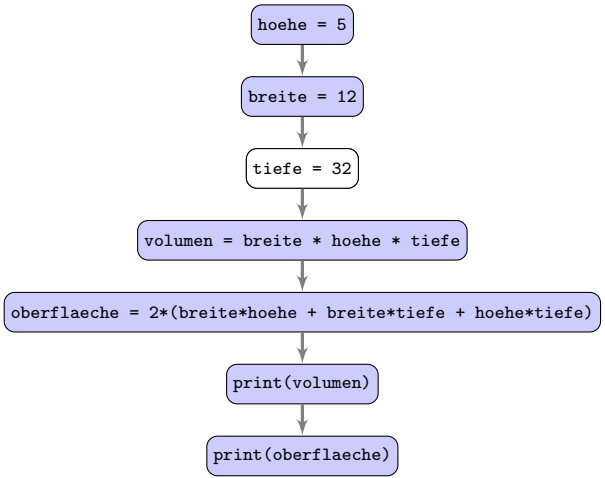


Einfache Vorstellung vom Speicher:

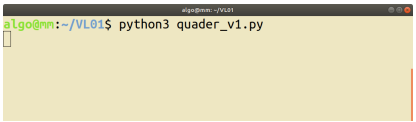
Variable	Wert
`hoehe`	5
`breite`	12
`tiefe`	
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

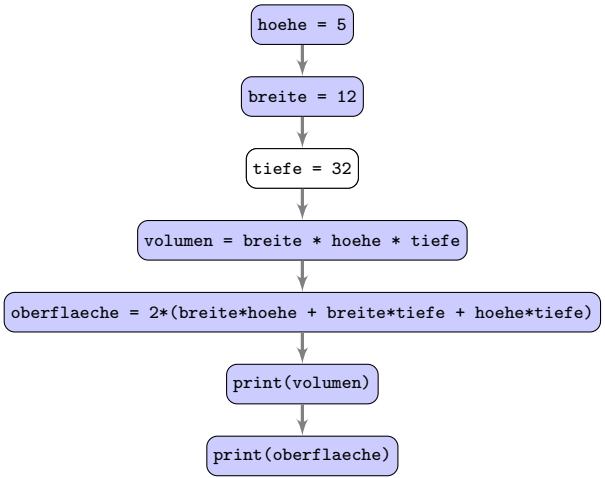


Einfache Vorstellung vom Speicher:

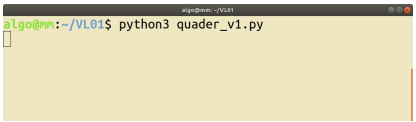
Variable	Wert
`hoehe`	5
`breite`	12
`tiefe`	
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

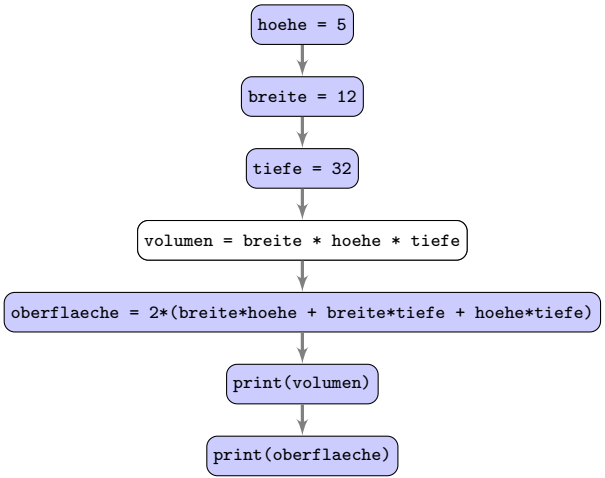


Einfache Vorstellung vom Speicher:

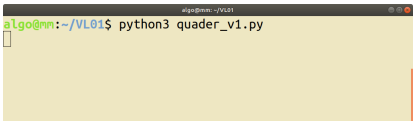
Variable	Wert
`hoehe`	5
`breite`	12
`tiefe`	32
`volumen`	
`oberflaeche`	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

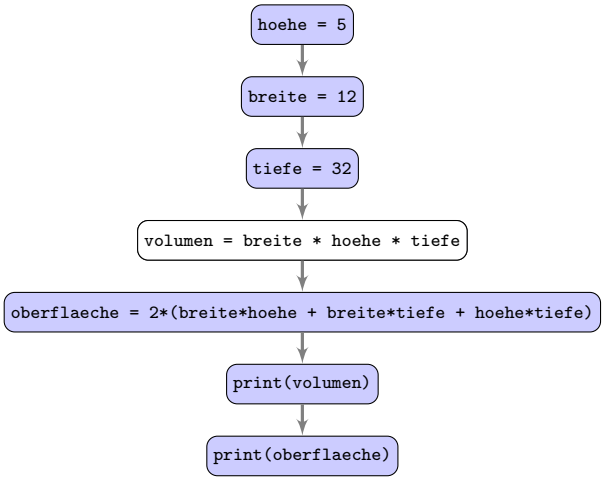


Einfache Vorstellung vom Speicher:

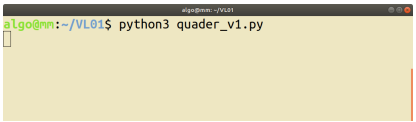
Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	
oberflaeche	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

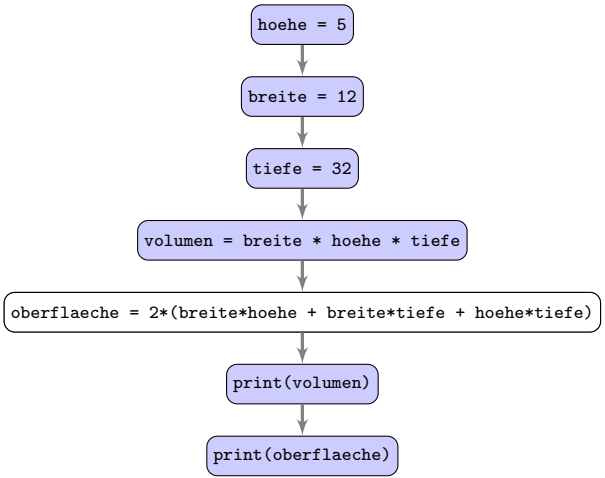


Einfache Vorstellung vom Speicher:

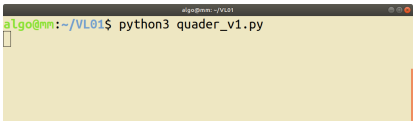
Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	1920
oberflaeche	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

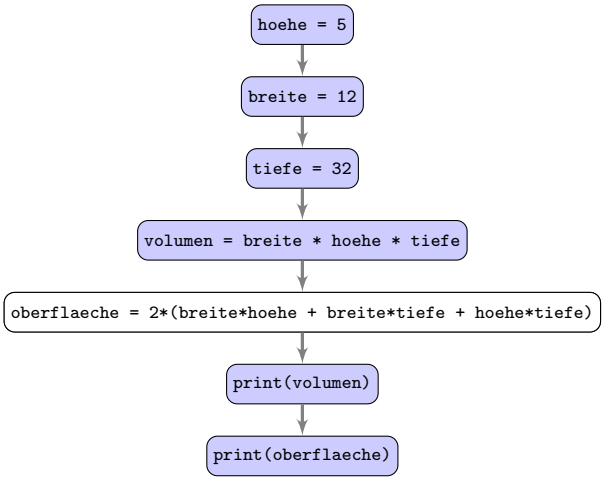


Einfache Vorstellung vom Speicher:

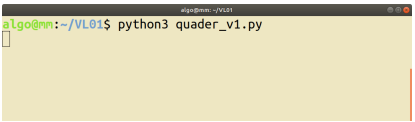
Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	1920
oberflaeche	

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

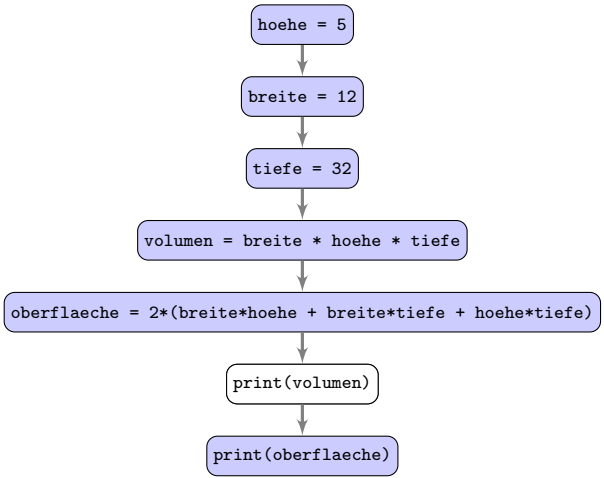


Einfache Vorstellung vom Speicher:

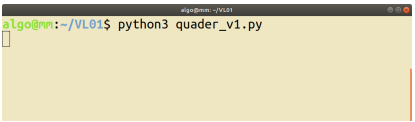
Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	1920
oberflaeche	1208

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

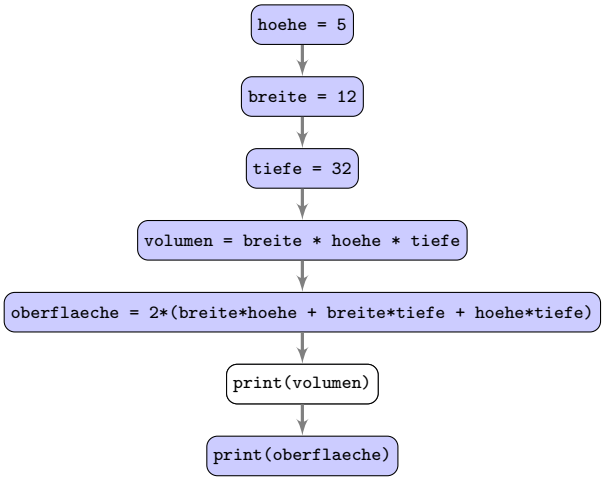


Einfache Vorstellung vom Speicher:

Variable	Wert
`hoehe`	5
`breite`	12
`tiefe`	32
`volumen`	1920
`oberflaeche`	1208

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

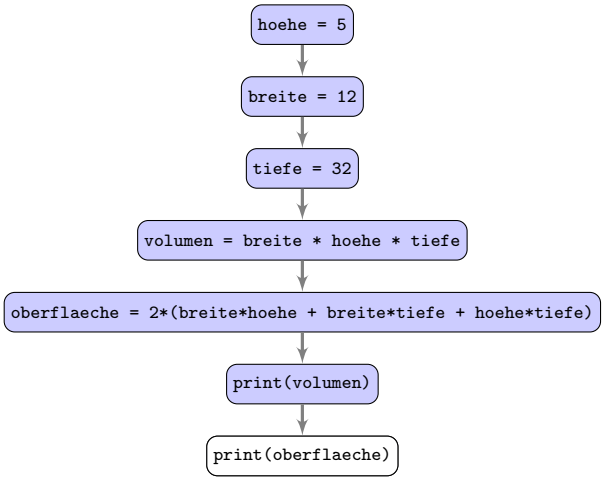
```
algo@mm:~/VL01$ python3 quader_v1.py
1920
```

Einfache Vorstellung vom Speicher:

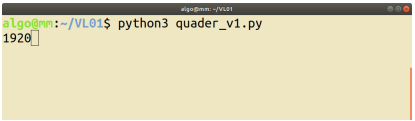
Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	1920
oberflaeche	1208

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt
und der Speicher wird entsprechend verändert.

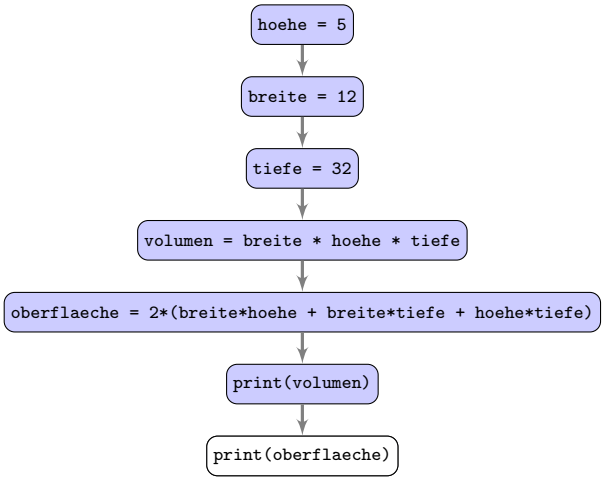


Einfache Vorstellung vom Speicher:

Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	1920
oberflaeche	1208

Was beim Start des Programms passiert ... (sehr grobe Vorstellung)

Das Programm legt eine Folge von Anweisungen fest.
Jede Anweisung liest und/oder schreibt im Speicher des Rechners.



Beim Start des Programms werden die Anweisungen in der vorgegebenen Reihenfolge ausgeführt und der Speicher wird entsprechend verändert.

```
algo@mm:~/VL01$ python3 quader_v1.py
1920
1208
algo@mm:~/VL01$
```

Einfache Vorstellung vom Speicher:

Variable	Wert
hoehe	5
breite	12
tiefe	32
volumen	1920
oberflaeche	1208

2 Der Datentyp `int` für ganze Zahlen

Zum Arbeiten mit ganzen Zahlen benutzt man den Datentyp `int` (*integer*, ganze Zahlen).

Für jeden Datentyp gibt es *Literale*, *Operatoren* und *Funktionen*.

Literale sind Darstellungen von Werten des Datentyps:

```
2020
2346283472947294
-1263516
0
```

Die Anweisung

```
hoehe = 5
```

weist der Variablen `hoehe` den `int`-Wert 5 zu (sehr grob gesagt).

2 Der Datentyp `int` für ganze Zahlen

Zum Arbeiten mit ganzen Zahlen benutzt man den Datentyp `int` (*integer*, ganze Zahlen).

Für jeden Datentyp gibt es *Literale*, *Operatoren* und *Funktionen*.

Literale sind Darstellungen von Werten des Datentyps:

```
2020
2346283472947294
-1263516
0
```

Die Anweisung

```
hoehe = 5
```

weist der Variablen `hoehe` den `int`-Wert 5 zu (sehr grob gesagt).

Operatoren und *Funktionen* dienen zum Rechnen mit Werten des Datentyps.

Operatoren:	`+`	Addition	`2+3`
	`-`	Subtraktion, Vorzeichen	`2-3`, `-2`
	`*`	Multiplikation	`2*3`
	`**`	Potenz	`2**3`
	`//`	ganzzahlige Division	`17//4`
	`%`	Rest bei ganzzahliger Division, modulo	`17%4`
Funktionen:	`abs( )`	Betrag eines `int`-Wertes	`abs(-17)`
	`max( , )`	Maximum zweier `int`-Werte	`max(17,4)`
	`min( , )`	Minimum zweier `int`-Werte	`min(17,4)`

Die Operatoren und Funktionen auf `int`-Werten
haben einen Wert vom Datentyp `int` als Ergebnis.

Ausdrücke vom Datentyp `int`

bestehen aus

- Literalen vom Datentyp `int`
- Variablen, die an *Objekte* vom Datentyp `int` gebunden sind (s.u.)
- den Operatoren `+ - * ** // %`
- Klammern `(` und `)`
- Funktionen vom Typ `int` ...

und werden zu einem Objekt vom Datentyp `int` ausgewertet.

Beispiele für `int`-Ausdrücke ohne Variablen:

```
60 * 60 * 24
(52-6)*40
max(32,17) + min(28, max(2,4))
(5+7)**2
5**2 + 2*5**7 + 7**max(9,12)
```

Operatoren haben unterschiedliche Bindungsstärke („Punktrechnung geht vor Strichrechnung“).
Im Zweifel: Klammern benutzen ...

Operator // (ganzzahlige Division)

a // b ist die *größte ganze Zahl kleiner oder gleich* $\frac{a}{b}$:

17 // 6 ist 2 (da „17 geteilt durch 6 gleich $\underline{2}$ Rest 5“ ist, d.h. $17 = \underline{2} \cdot 6 + 5$),

und -17 // 6 ist −3 (da $-17 = \underline{-3} \cdot 6 + 1$).

Mathematische Schreibweise für die ganzzahlige Division:

a // b entspricht $\lfloor \frac{a}{b} \rfloor$

$\lfloor x \rfloor$ ist die größte ganze Zahl $\leqslant x$ („untere Gaußklammer“)

Operator % (modulo-Operator)

a % b ist der *Rest bei der ganzzahligen Division* von a durch b (für positives b).

17 % 6 ist 5 (da „17 geteilt durch 6 gleich 2 Rest $\underline{5}$“ ist, d.h. $17 = 2 \cdot 6 + \underline{5}$)
und −17 % 6 ist 1 (da $-17 = -3 \cdot 6 + \underline{1}$).

Mathematische Schreibweise für a % b ist $a \bmod b$.

Es gilt: $a = b \cdot \lfloor \frac{a}{b} \rfloor + (a \bmod b)$.

Beispiele für `int`-Ausdrücke ohne Variablen:

```
abs((16*60+15)-(17*60+45))//60
```
die Anzahl der ganzen Stunden im Zeitraum 16:15 – 17:45 Uhr

```
abs((16*60+15)-(17*60+45))%60
```
die Minuten, die zu den ganzen Stunden in dem Zeitraum dazukommen

Weitere Beispiele für `int`-Ausdrücke (mit `int`-Variablen (farbig notiert)):

```
hh * 60 * 60 + mm * 60 + sec
```

```
(52 - 6) * 40 // LP_pro_Jahr
```

```
(1+5*((jahr-1)%4)+4*((jahr-1)%100)+6*((jahr-1)%400))%7
```

Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

Beispiele:

```
jahr = 2020
jahr = jahr + 1
```

Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

Beispiele:

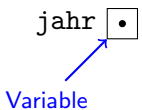
```
jahr = 2020
jahr = jahr + 1
```

Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

`jahr = 2020`

erzeugt ein *Objekt* mit Wert 2020 vom Typ `int`, und bindet `jahr` daran



Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

Beispiele:

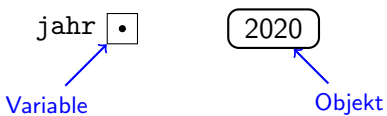
```
jahr = 2020
jahr = jahr + 1
```

Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

```
jahr = 2020
```

erzeugt ein *Objekt* mit Wert 2020 vom Typ `int`, und bindet `jahr` daran



Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

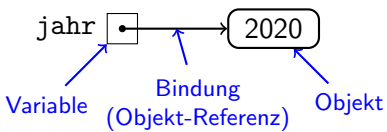
Beispiele:

```
jahr = 2020
jahr = jahr + 1
```

Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

`jahr = 2020` erzeugt ein *Objekt* mit Wert 2020 vom Typ `int`, und bindet `jahr` daran



Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

Beispiele:

```
jahr = 2020
jahr = jahr + 1
```

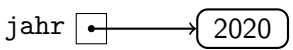
Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

```
jahr = 2020
```

erzeugt ein Objekt mit Wert `jahr+1` vom Typ `int`, und bindet `jahr` daran

```
jahr = jahr + 1
```



Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

Beispiele:

```
jahr = 2020
jahr = jahr + 1
```

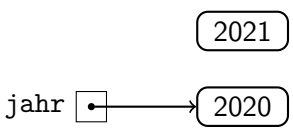
Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

```
jahr = 2020
```

```
jahr = jahr + 1
```

erzeugt ein Objekt mit Wert `jahr+1` vom Typ `int`, und bindet `jahr` daran



Anweisungen

Eine *Anweisung* besteht aus einer Variablen, = und einem Ausdruck.

Beispiele:

```
jahr = 2020
jahr = jahr + 1
```

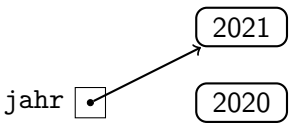
Beim Abarbeiten einer Anweisung, deren Wert einen grundlegenden Datentyp hat, wird

- zuerst der Ausdruck ausgewertet
  und ein Objekt mit Typ und Wert des Ausdrucks erzeugt, und
- danach wird die Variable an das Objekt *gebunden*.

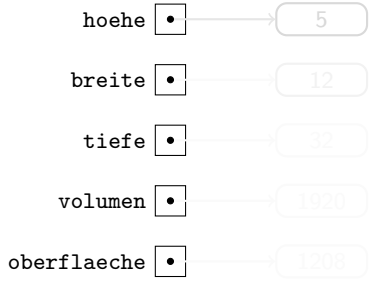
```
jahr = 2020
```

```
jahr = jahr + 1
```

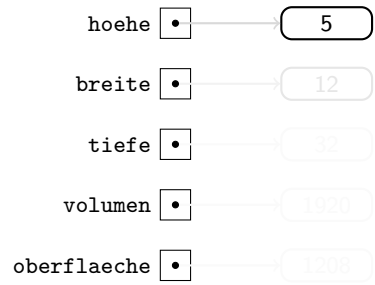
erzeugt ein Objekt mit Wert `jahr+1` vom Typ `int`, und bindet `jahr` daran



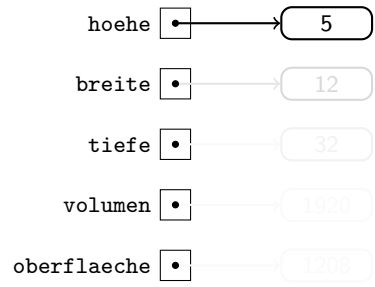
```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



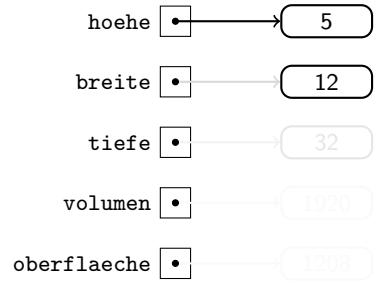
```python
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



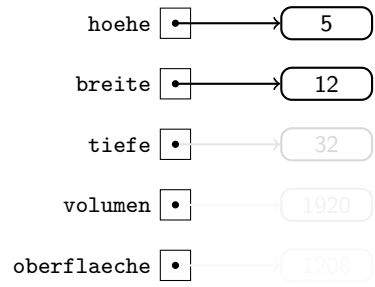
```python
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



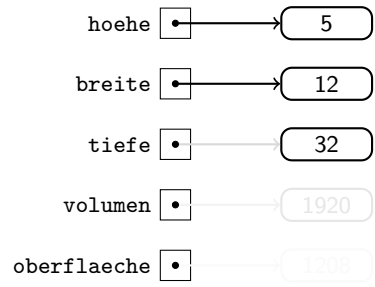
```python
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



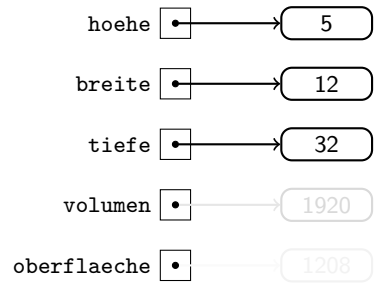
```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



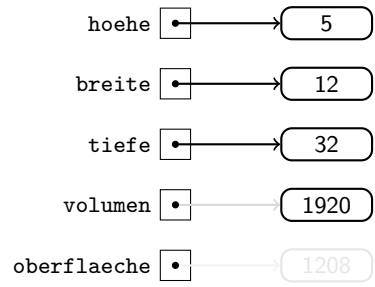
```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



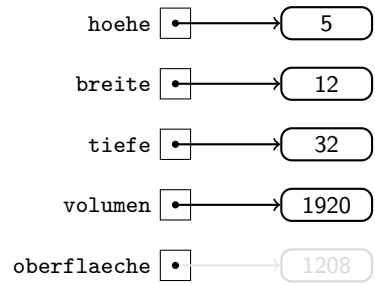
```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



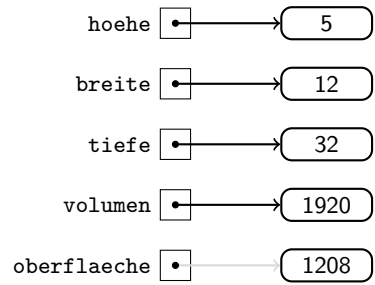
```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



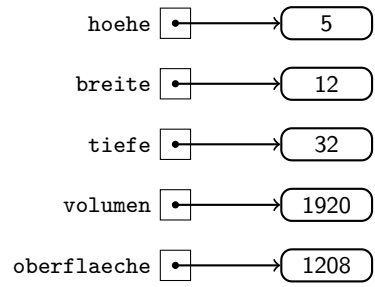
```python
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



```
#-----------------------------------------------------------------
# quader_v1.py
#-----------------------------------------------------------------
# Berechne das Volumen und die Oberfläche eines Quaders.
#-----------------------------------------------------------------

# Zuerst werden die Maße des Quaders angegeben.
hoehe = 5
breite = 12
tiefe = 32

# Daraus können nach den bekannten Formeln
# das Volumen und die Oberfläche berechnet werden.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print(volumen)
print(oberflaeche)

#-----------------------------------------------------------------
# python3 quader_v1.py
# 1920
# 1208
```



Weitere Beispiele für Anweisungen:

```
w = (1+5*((jahr-1)%4)+4*((jahr-1)%100)+6*((jahr-1)%400))%7
```

```
arbeitsstundenProLP = (52 - 6) * 40 // LP_pro_Jahr
```

```
zaehler = zaehler + 1
```

```
zaehler += 1
```

3 Der Datentyp `str` für Texte

Zum Arbeiten mit Texten benutzt man den Datentyp `str` (*string*, Zeichenketten).

Die **Literale** sind Folgen von Zeichen, die man an der Tastatur eingeben kann und die durch `' ... '` oder `" ... "` eingeschlossen sind.

```
'Das ist ein str-Literal.'      '1234'      "a1/2 2/b_3()"
```

Achtung: das Literal `1234` ist vom Typ `int`
das Literal `'1234'` ist vom Typ `str`

3 Der Datentyp `str` für Texte

Zum Arbeiten mit Texten benutzt man den Datentyp `str` (*string*, Zeichenketten).

Die **Literale** sind Folgen von Zeichen, die man an der Tastatur eingeben kann und die durch `' ... '` oder `" ... "` eingeschlossen sind.

```
'Das ist ein str-Literal.'      '1234'      "a1/2 2/b_3()"
```

Achtung: das Literal `1234` ist vom Typ `int`
das Literal `'1234'` ist vom Typ `str`

Operatoren für den Datentyp `str` sind + und *.

Der Operator + schreibt zwei Strings hintereinander (Konkatenation).

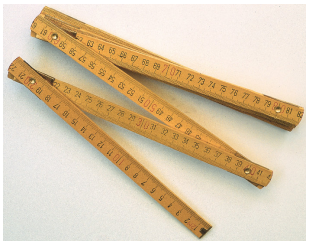
Der Ausdruck `'Das ist ' + 'ein Satz.'`
ergibt den String `'Das ist ein Satz.'` .

Der Operator * schreibt wiederholt einen String hintereinander.

Der Ausdruck `'Ole' * 4` (`'Ole'` ist vom Typ `str` und `4` ist vom Typ `int`)
ergibt den String `'OleOleOleOle'` (4mal `'Ole'` hintereinandergeschrieben).

Die Operatoren haben einen Wert vom Datentyp `str` als Ergebnis.

Programmier-Auftrag:
Gib die Einteilungsmarken auf einem Zollstock aus



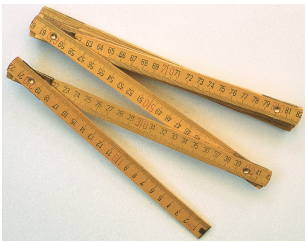
Die Striche sind im Millimeter-Abstand.

Alle 5mm ist der Strich länger, und alle 10mm ist der Strich noch länger.

Beispiel für die ersten 29 Striche:

```
|         |
|   |   |   |   |
||||||||||||||||||||||||||||
```

Programmier-Auftrag:
Gib die Einteilungsmarken auf einem Zollstock aus



Die Striche sind im Millimeter-Abstand.
Alle 5mm ist der Strich länger, und alle 10mm ist der Strich noch länger.

Beispiel für die ersten 29 Striche:

```
|           |
|   |   |   |   |
|||||||||||||||||||||||||||||
```

Die Idee

Die Striche sind im Millimeter-Abstand.
Alle 5mm ist der Strich länger, und alle 10mm ist der Strich noch länger.

Beispiel für die ersten 29 Striche – sie sind auf 3 Zeilen aufgeteilt:

unterste Zeile: schreibe 29 Striche `'|' * 29`

mittlere Zeile: schreibe 4 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden `'␣␣␣␣|' * (29//5)`

oberste Zeile: schreibe 9 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden `(' '*9 + '|') * (29//10)`

Die Idee

Die Striche sind im Millimeter-Abstand.
Alle 5mm ist der Strich länger, und alle 10mm ist der Strich noch länger.

Beispiel für die ersten 29 Striche – sie sind auf 3 Zeilen aufgeteilt:

unterste Zeile:	schreibe 29 Striche	`'|' * 29`
mittlere Zeile:	schreibe 4 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden	`'␣␣␣␣|' * (29//5)`
oberste Zeile:	schreibe 9 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden	`(' '*9 + '|') * (29//10)`

Die Idee

Die Striche sind im Millimeter-Abstand.
Alle 5mm ist der Strich länger, und alle 10mm ist der Strich noch länger.

Beispiel für die ersten 29 Striche – sie sind auf 3 Zeilen aufgeteilt:

unterste Zeile:	schreibe 29 Striche	`'|' * 29`
mittlere Zeile:	schreibe 4 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden	`'␣␣␣␣|' * (29//5)`
oberste Zeile:	schreibe 9 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden	`(' '*9 + '|') * (29//10)`

Die Idee

Die Striche sind im Millimeter-Abstand.
Alle 5mm ist der Strich länger, und alle 10mm ist der Strich noch länger.

Beispiel für die ersten 29 Striche – sie sind auf 3 Zeilen aufgeteilt:

unterste Zeile:	schreibe 29 Striche	`'|' * 29`
mittlere Zeile:	schreibe 4 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden	`'␣␣␣␣|' * (29//5)`
oberste Zeile:	schreibe 9 Leerzeichen und einen Strich, bis 29 Zeichen geschrieben wurden	`(' '*9 + '|') * (29//10)`

Der grobe Ablauf des Programms

1. Gib die Länge des Zollstocks an – er soll 79 Striche lang sein.

   Der Zollstock im Beispiel hat Länge 29 (die Anzahl der Striche in der untersten Zeile).

2. Berechne die unterste, mittlere und obere Zeile der Einteilungsmarken.

```
'|' * 29
'    |' * (29//5)
(' '*9 + '|') * (29//10)
```

3. Gib die drei Zeilen von oben nach unten aus.

Der grobe Ablauf des Programms

1. Gib die Länge des Zollstocks an – er soll 79 Striche lang sein.

   Der Zollstock im Beispiel hat Länge 29 (die Anzahl der Striche in der untersten Zeile).

2. Berechne die unterste, mittlere und obere Zeile der Einteilungsmarken.

```
'|' * 29
'␣␣␣␣|' * (29//5)
(' '*9 + '|') * (29//10)
```

3. Gib die drei Zeilen von oben nach unten aus.

Der grobe Ablauf des Programms

1. Gib die Länge des Zollstocks an – er soll 79 Striche lang sein.

   Der Zollstock im Beispiel hat Länge 29 (die Anzahl der Striche in der untersten Zeile).

2. Berechne die unterste, mittlere und obere Zeile der Einteilungsmarken.

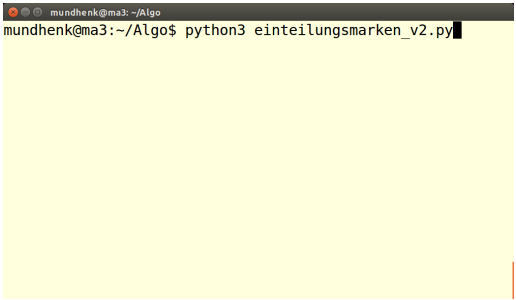
```
'|' * 29
'␣␣␣␣|' * (29//5)
(' '*9 + '|') * (29//10)
```

3. Gib die drei Zeilen von oben nach unten aus.

```
#-------------------------------------------------------------------------------
# einteilungsmarken.py
#-------------------------------------------------------------------------------

# Die Variable laenge gibt die Länge des Zollstocks in Millimeter an.
laenge = 79

# Die Variable untersteZeile besteht aus den Millimeter-Strichen des Zollstocks.
untersteZeile = '|' * laenge

# Die Variable mittlereZeile besteht aus 5-Millimeter-Strichen.
mittlereZeile = '    |' * ( laenge//5 )

# Die Variable obersteZeile besteht aus 10-Millimeter-Strichen.
obersteZeile = ((' '*9) + '|') * (laenge//10)

# Die Einteilungsmarken werden ausgegeben.
print(obersteZeile)
print(mittlereZeile)
print(untersteZeile)


#-------------------------------------------------------------------------------
# python3 einteilungsmarken.py
#         |         |         |         |         |         |         |
#     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



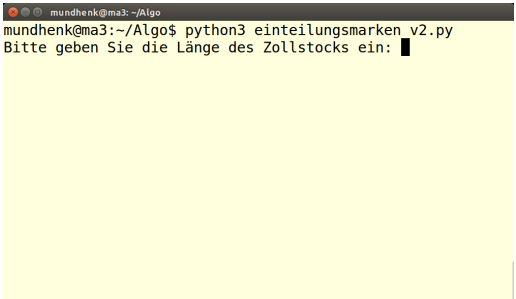
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



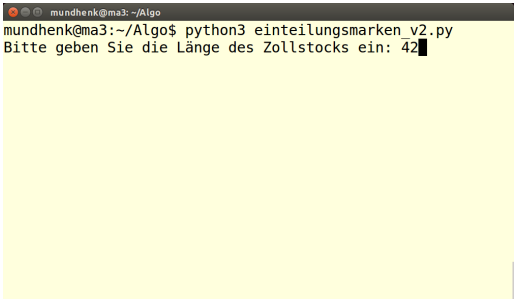
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



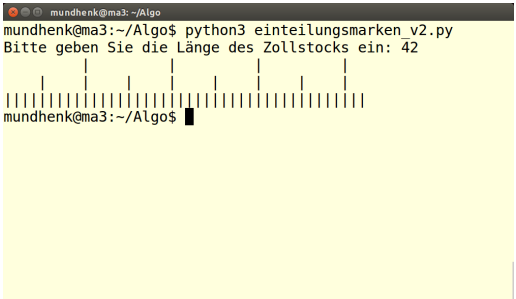
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



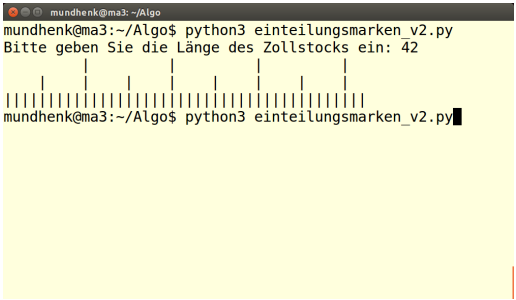
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



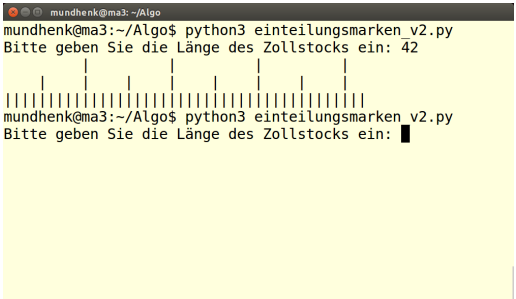
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



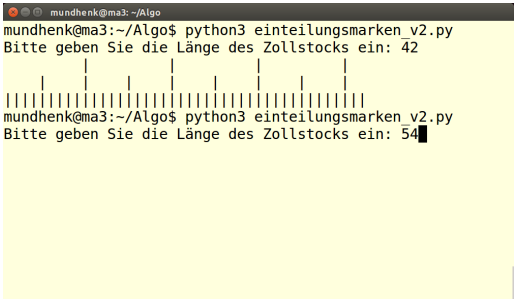
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



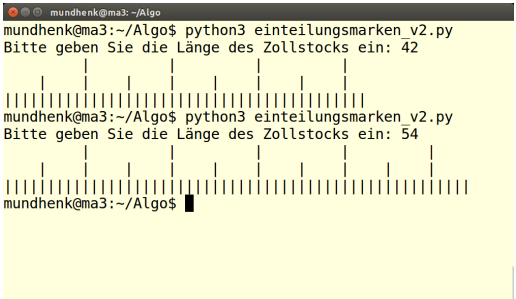
Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

Eingaben von der Tastatur lesen

Wir wollen das Programm nun so ändern,
dass die Länge des Zollstocks über die Tastatur eingegeben wird.
Das geänderte Programm heißt `einteilungsmarken_v2.py` .



Funktionen für den Datentyp `str` sind `input(x)` und `int(x)` .

`input(x)` gibt String `x` auf dem Bildschirm aus und hat als Ergebnis vom Datentyp `str`, was an der Tastatur eingegeben wird – bis die *Enter*-Taste gedrückt wird (*standard input*).

`int(x)` hat als Ergebnis den `int`-Wert, der von String `x` dargestellt wird.

```
#-------------------------------------------------------------------------------
# einteilungsmarken_v2.py
#-------------------------------------------------------------------------------

# Die Variable laenge gibt die Länge des Zollstocks in Millimeter an.
# Ihr Wert wird von der Tastatur (standard input) gelesen.
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)

# Die Variable untersteZeile besteht aus den Millimeter-Strichen des Zollstocks.
untersteZeile = '|' * laenge
# Die Variable mittlereZeile besteht aus 5-Millimeter-Strichen.
mittlereZeile = '    |' * ( laenge//5 )
# Die Variable obersteZeile besteht aus 10-Millimeter-Strichen.
obersteZeile = ((' '*9) + '|') * (laenge//10)

# Die Einteilungsmarken werden ausgegeben.
print(obersteZeile)
print(mittlereZeile)
print(untersteZeile)

#-------------------------------------------------------------------------------
# python3 einteilungsmarken_v2.py
# Bitte geben Sie die Länge des Zollstocks an: 79
#          |         |         |         |         |         |         |
#     |    |    |    |    |    |    |    |    |    |    |    |    |    |
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

```python
#-------------------------------------------------------------------------------
# einteilungsmarken_v2.py
#-------------------------------------------------------------------------------

# Die Variable laenge gibt die Länge des Zollstocks in Millimeter an.
# Ihr Wert wird von der Tastatur (standard input) gelesen.
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)

# Die Variable untersteZeile besteht aus den Millimeter-Strichen des Zollstocks.
untersteZeile = '|' * laenge
# Die Variable mittlereZeile besteht aus 5-Millimeter-Strichen.
mittlereZeile = '    |' * ( laenge//5 )
# Die Variable obersteZeile besteht aus 10-Millimeter-Strichen.
obersteZeile = ((' '*9) + '|') * (laenge//10)

# Die Einteilungsmarken werden ausgegeben.
print(obersteZeile)
print(mittlereZeile)
print(untersteZeile)

#-------------------------------------------------------------------------------
# python3 einteilungsmarken_v2.py
# Bitte geben Sie die Länge des Zollstocks an: 52
#          |         |         |         |         |
#     |    |    |    |    |    |    |    |    |    |
# ||||||||||||||||||||||||||||||||||||||||||||||||||
```

Die Bestimmung des Wertes der Variable `laenge` am Anfang des Programms wurde verändert.

```
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)
```

Der Funktionsaufruf `input('Bitte geben Sie die Länge des Zollstocks ein: ')`
gibt den String `'Bitte ...'` auf dem Bildschirm aus und
wartet, dass etwas über die Tastatur eingegeben wird – *Enter* beendet die Eingabe.
Diese Eingabe ist ein String und ist das Ergebnis des Funktionsaufrufs.

Die Anweisung `eingabe = input('Bitte geben Sie die Länge des Zollstocks ein: ')`
speichert das Ergebnis des Funktionsaufrufs in der Variable `eingabe` (Typ `str`).

Wir brauchen aber statt des Strings (z.B. `'42'`) seinen `int`-Wert (z.B. `42`),
da der Rest des Programmes davon ausgeht, dass `laenge` den Typ `int` hat.

Der Funktionsaufruf `int(eingabe)`
hat als Ergebnis den `int`-Wert von `eingabe` (falls der String eine ganze Zahl darstellt).

Die Anweisung `laenge = int(eingabe)` speichert diesen `int`-Wert in der Variable `laenge`.

Nun kann der Rest des Programms zum Schreiben der Striche so ablaufen wie zuvor.

Die Bestimmung des Wertes der Variable `laenge` am Anfang des Programms wurde verändert.

```
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)
```

Der Funktionsaufruf `input('Bitte geben Sie die Länge des Zollstocks ein: ')`
gibt den String `'Bitte ...'` auf dem Bildschirm aus und
wartet, dass etwas über die Tastatur eingegeben wird – *Enter* beendet die Eingabe.
Diese Eingabe ist ein String und ist das Ergebnis des Funktionsaufrufs.

Die Anweisung `eingabe = input('Bitte geben Sie die Länge des Zollstocks ein: ')`
speichert das Ergebnis des Funktionsaufrufs in der Variable `eingabe` (Typ `str`).

Wir brauchen aber statt des Strings (z.B. `'42'`) seinen `int`-Wert (z.B. `42`),
da der Rest des Programmes davon ausgeht, dass `laenge` den Typ `int` hat.

Der Funktionsaufruf `int(eingabe)`
hat als Ergebnis den `int`-Wert von `eingabe` (falls der String eine ganze Zahl darstellt).

Die Anweisung `laenge = int(eingabe)` speichert diesen `int`-Wert in der Variable `laenge`.

Nun kann der Rest des Programms zum Schreiben der Striche so ablaufen wie zuvor.

Die Bestimmung des Wertes der Variable `laenge` am Anfang des Programms wurde verändert.

```
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)
```

Der Funktionsaufruf `input('Bitte geben Sie die Länge des Zollstocks ein: ')`
gibt den String `'Bitte ...'` auf dem Bildschirm aus und
wartet, dass etwas über die Tastatur eingegeben wird – *Enter* beendet die Eingabe.
Diese Eingabe ist ein String und ist das Ergebnis des Funktionsaufrufs.

Die Anweisung `eingabe = input('Bitte geben Sie die Länge des Zollstocks ein: ')`
speichert das Ergebnis des Funktionsaufrufs in der Variable `eingabe` (Typ `str`).

Wir brauchen aber statt des Strings (z.B. `'42'`) seinen `int`-Wert (z.B. `42`),
da der Rest des Programmes davon ausgeht, dass `laenge` den Typ `int` hat.

Der Funktionsaufruf `int(eingabe)`
hat als Ergebnis den `int`-Wert von `eingabe` (falls der String eine ganze Zahl darstellt).

Die Anweisung `laenge = int(eingabe)` speichert diesen `int`-Wert in der Variable `laenge`.

Nun kann der Rest des Programms zum Schreiben der Striche so ablaufen wie zuvor.

Die Bestimmung des Wertes der Variable `laenge` am Anfang des Programms wurde verändert.

```
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)
```

Der Funktionsaufruf `input('Bitte geben Sie die Länge des Zollstocks ein: ')`
gibt den String `'Bitte ...'` auf dem Bildschirm aus und
wartet, dass etwas über die Tastatur eingegeben wird – *Enter* beendet die Eingabe.
Diese Eingabe ist ein String und ist das Ergebnis des Funktionsaufrufs.

Die Anweisung `eingabe = input('Bitte geben Sie die Länge des Zollstocks ein: ')`
speichert das Ergebnis des Funktionsaufrufs in der Variable `eingabe` (Typ `str`).

Wir brauchen aber statt des Strings (z.B. `'42'`) seinen `int`-Wert (z.B. 42),
da der Rest des Programmes davon ausgeht, dass `laenge` den Typ `int` hat.

Der Funktionsaufruf `int(eingabe)`
hat als Ergebnis den `int`-Wert von `eingabe` (falls der String eine ganze Zahl darstellt).

Die Anweisung `laenge = int(eingabe)` speichert diesen `int`-Wert in der Variable `laenge`.

Nun kann der Rest des Programms zum Schreiben der Striche so ablaufen wie zuvor.

Die Bestimmung des Wertes der Variable `laenge` am Anfang des Programms wurde verändert.

```
eingabe = input('Bitte geben Sie die Länge des Zollstocks an: ')
laenge = int(eingabe)
```

Der Funktionsaufruf `input('Bitte geben Sie die Länge des Zollstocks ein: ')`
gibt den String `'Bitte ...'` auf dem Bildschirm aus und
wartet, dass etwas über die Tastatur eingegeben wird – *Enter* beendet die Eingabe.
Diese Eingabe ist ein String und ist das Ergebnis des Funktionsaufrufs.

Die Anweisung `eingabe = input('Bitte geben Sie die Länge des Zollstocks ein: ')`
speichert das Ergebnis des Funktionsaufrufs in der Variable `eingabe` (Typ `str`).

Wir brauchen aber statt des Strings (z.B. `'42'`) seinen `int`-Wert (z.B. 42),
da der Rest des Programmes davon ausgeht, dass `laenge` den Typ `int` hat.

Der Funktionsaufruf `int(eingabe)`
hat als Ergebnis den `int`-Wert von `eingabe` (falls der String eine ganze Zahl darstellt).

Die Anweisung `laenge = int(eingabe)` speichert diesen `int`-Wert in der Variable `laenge`.

Nun kann der Rest des Programms zum Schreiben der Striche so ablaufen wie zuvor.

int-Werte zu str-Werten umwandeln

Um den Wert eines Ausdrucks a mit `print(a)` ausgeben zu können,
muss a vom Typ `str` sein oder in ein Objekt vom Typ `str` umgewandelt werden können.

Die Funktion `str(x)` gibt einen String zurück,
der den Wert von `x` als String darstellt.

Beispiele:

`str(123)` gibt den String `'123'` zurück (123 ist ein `int`-Literal)

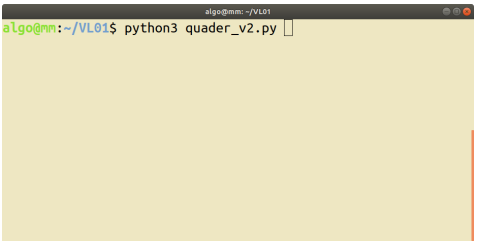
```
v = 123 + 456                                  (Hier ist + die Operation auf int.)
s = 'Das Volumen ist ' + str(v) + '.'          (Hier ist + die Operation auf str.)
```

(s ist der String `'Das Volumen ist 579.'`)

Nun können wir das Programm zur Berechnung von Volumen und Oberfläche eines Quaders so ändern, dass

- die Maße des Quaders über die Tastatur eingegeben und
- die Ergebnisse schöner ausgegeben werden.

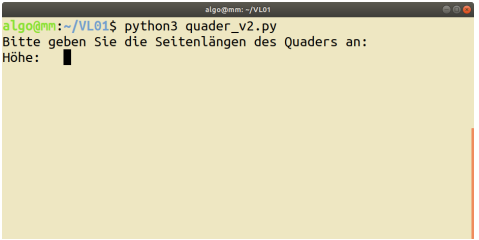
```
#---------------------------------------------------------------------
# quader_v2.py
#---------------------------------------------------------------------

# Zuerst werden die Maße des Quaders eingelesen.
print('Bitte geben Sie die Seitenlängen des Quaders an:')
hoehe  = int(input('Höhe:   '))
breite = int(input('Breite: '))
tiefe  = int(input('Tiefe:  '))

# Das Volumen und die Oberfläche des Quaders werden berechnet.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print('Das Volumen des Quaders ist ' + str(volumen) + '.')
print('Die Oberfläche des Quaders ist ' + str(oberflaeche) + '.')
```



Nun können wir das Programm zur Berechnung von Volumen und Oberfläche eines Quaders so ändern, dass

- die Maße des Quaders über die Tastatur eingegeben und
- die Ergebnisse schöner ausgegeben werden.

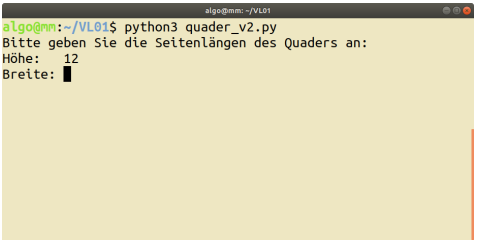
```
#-----------------------------------------------------------------------
# quader_v2.py
#-----------------------------------------------------------------------

# Zuerst werden die Maße des Quaders eingelesen.
print('Bitte geben Sie die Seitenlängen des Quaders an:')
hoehe  = int(input('Höhe:   '))
breite = int(input('Breite: '))
tiefe  = int(input('Tiefe:  '))

# Das Volumen und die Oberfläche des Quaders werden berechnet.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print('Das Volumen des Quaders ist ' + str(volumen) + '.')
print('Die Oberfläche des Quaders ist ' + str(oberflaeche) + '.')
```



Nun können wir das Programm zur Berechnung von Volumen und Oberfläche eines Quaders so ändern, dass

- die Maße des Quaders über die Tastatur eingegeben und
- die Ergebnisse schöner ausgegeben werden.

```
#-----------------------------------------------------------------------
# quader_v2.py
#-----------------------------------------------------------------------

# Zuerst werden die Maße des Quaders eingelesen.
print('Bitte geben Sie die Seitenlängen des Quaders an:')
hoehe  = int(input('Höhe:   '))
breite = int(input('Breite: '))
tiefe  = int(input('Tiefe:  '))

# Das Volumen und die Oberfläche des Quaders werden berechnet.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print('Das Volumen des Quaders ist ' + str(volumen) + '.')
print('Die Oberfläche des Quaders ist ' + str(oberflaeche) + '.')
```



Nun können wir das Programm zur Berechnung von Volumen und Oberfläche eines Quaders so ändern, dass

- die Maße des Quaders über die Tastatur eingegeben und
- die Ergebnisse schöner ausgegeben werden.

```
#-----------------------------------------------------------------------
# quader_v2.py
#-----------------------------------------------------------------------

# Zuerst werden die Maße des Quaders eingelesen.
print('Bitte geben Sie die Seitenlängen des Quaders an:')
hoehe  = int(input('Höhe:   '))
breite = int(input('Breite: '))
tiefe  = int(input('Tiefe:  '))

# Das Volumen und die Oberfläche des Quaders werden berechnet.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print('Das Volumen des Quaders ist ' + str(volumen) + '.')
print('Die Oberfläche des Quaders ist ' + str(oberflaeche) + '.')
```

```
algo@mm: ~/VL01
algo@mm:~/VL01$ python3 quader_v2.py
Bitte geben Sie die Seitenlängen des Quaders an:
Höhe:   12
Breite: 11
Tiefe:
```

Nun können wir das Programm zur Berechnung von Volumen und Oberfläche eines Quaders so ändern, dass

- die Maße des Quaders über die Tastatur eingegeben und
- die Ergebnisse schöner ausgegeben werden.

```
#-----------------------------------------------------------------------
# quader_v2.py
#-----------------------------------------------------------------------

# Zuerst werden die Maße des Quaders eingelesen.
print('Bitte geben Sie die Seitenlängen des Quaders an:')
hoehe  = int(input('Höhe:   '))
breite = int(input('Breite: '))
tiefe  = int(input('Tiefe:  '))

# Das Volumen und die Oberfläche des Quaders werden berechnet.
volumen = hoehe * breite * tiefe
oberflaeche = 2 * ( hoehe*breite + hoehe*tiefe + breite*tiefe )

# Die berechneten Werte werden ausgegeben.
print('Das Volumen des Quaders ist ' + str(volumen) + '.')
print('Die Oberfläche des Quaders ist ' + str(oberflaeche) + '.')
```

```
algo@mm: ~/VL01
algo@mm:~/VL01$ python3 quader_v2.py
Bitte geben Sie die Seitenlängen des Quaders an:
Höhe:   12
Breite: 11
Tiefe:  10
Das Volumen des Quaders ist 1320.
Die Oberfläche des Quaders ist 724.
algo@mm:~/VL01$
```

Es gibt „spezielle“ Zeichen wie z.B. den Zeilenumbruch `\n`.

```
print('Zeile 1\nZeile 2')
```

liefert die Ausgabe

```
Zeile1
Zeile2
```

`print(x)` gibt auf dem Bildschirm String `x` (bzw. `str(x)`) gefolgt von `\n` aus.

Anstelle des Zeilenumbruchs `\n` kann man auch andere Zeichen wählen.
`print(x, end='')` gibt nur String `x` aus.
`print(x, end='\n\n')` gibt String `x` gefolgt von zwei Zeilenumbrüchen aus.

Es gibt „spezielle“ Zeichen wie z.B. den Zeilenumbruch `\n`.

```
print('Zeile 1\nZeile 2')
```

liefert die Ausgabe

```
Zeile1
Zeile2
```

`print(x)` gibt auf dem Bildschirm String `x` (bzw. `str(x)`) gefolgt von `\n` aus.

Anstelle des Zeilenumbruchs `\n` kann man auch andere Zeichen wählen.
`print(x, end='')` gibt nur String `x` aus.
`print(x, end='\n\n')` gibt String `x` gefolgt von zwei Zeilenumbrüchen aus.

4 Der Datentyp `bool` für Wahrheitswerte

Zum Arbeiten mit Wahrheitswerten benutzt man den Datentyp `bool` (George Boole (1815-1864)).

Die **Literale** sind `True` (für den Wahrheitswert *wahr*) und `False` (für den Wahrheitswert *falsch*).

Ausdrücke, die Wahrheitswerte als Ergebnis haben, kann man z.B. auch mit `int`-Ausdrücken und Vergleichsoperatoren bzw. mit `str`-Ausdrücken und Vergleichsoperatoren bilden.

Operator	Bedeutung	Ausdrücke und ihre Wahrheitswerte			
		`True`	`False`	`True`	`False`
`==`	gleich	`3==3`	`2==3`	`'abc'=="abc"`	`'abc'=='abc '`
`!=`	ungleich	`2!=3`	`3!=3`	`'abc'!='Abc'`	`'a2c'!='a2c'`
`<`	kleiner als	`2<3`	`3<3`	`'ab'=='abc'`	`'a'=='A'`
`<=`	kleiner oder gleich	`3<=3`	`4<=3`	`'abc'<='abc'`	`'abc'<='ab'`
`>`	größer als	`4>3`	`3>4`	`'00'>'0'`	`'A'>'a'`
`>=`	größer oder gleich	`4>=4`	`3>=4`	`'a'>='B'`	`'0011'>='011'`

4 Der Datentyp `bool` für Wahrheitswerte

Zum Arbeiten mit Wahrheitswerten benutzt man den Datentyp `bool` (George Boole (1815-1864)).

Die **Literale** sind `True` (für den Wahrheitswert *wahr*) und `False` (für den Wahrheitswert *falsch*).

Ausdrücke, die Wahrheitswerte als Ergebnis haben, kann man z.B. auch mit `int`-Ausdrücken und Vergleichsoperatoren bzw. mit `str`-Ausdrücken und Vergleichsoperatoren bilden.

Operator	Bedeutung	Ausdrücke und ihre Wahrheitswerte			
		`True`	`False`	`True`	`False`
`==`	gleich	`3==3`	`2==3`	`'abc'=="abc"`	`'abc'=='abc '`
`!=`	ungleich	`2!=3`	`3!=3`	`'abc'!='Abc'`	`'a2c'!='a2c'`
`<`	kleiner als	`2<3`	`3<3`	`'ab'=='abc'`	`'a'=='A'`
`<=`	kleiner oder gleich	`3<=3`	`4<=3`	`'abc'<='abc'`	`'abc'<='ab'`
`>`	größer als	`4>3`	`3>4`	`'00'>'0'`	`'A'>'a'`
`>=`	größer oder gleich	`4>=4`	`3>=4`	`'a'>='B'`	`'0011'>='011'`

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

```
True and (False or True)
```

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`True and (False or True)` hat den gleichen Wert wie

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`True and (False or True)` hat den gleichen Wert wie

`True and True`

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`True and (False or True)` hat den gleichen Wert wie

`True and True` hat den Wert `True`.

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

```
(False or not (True and False)) and False
```

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

`(False or not False) and False`

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

`(False or not False) and False` hat den gleichen Wert wie

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

`(False or not False) and False` hat den gleichen Wert wie

`(False or True) and False`

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

`(False or not False) and False` hat den gleichen Wert wie

`(False or True) and False` hat den gleichen Wert wie

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

`(False or not False) and False` hat den gleichen Wert wie

`(False or True) and False` hat den gleichen Wert wie

`True and False`

Operatoren sind `and` und `or` (beide 2stellig), sowie `not` (1stellig).
Das Ergebnis der Operatoren ist vom Typ `bool`.
Die folgenden Tabellen beschreiben die Operatoren.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Beispiele für Ausdrücke und deren Werte:

`(False or not (True and False)) and False` hat den gleichen Wert wie

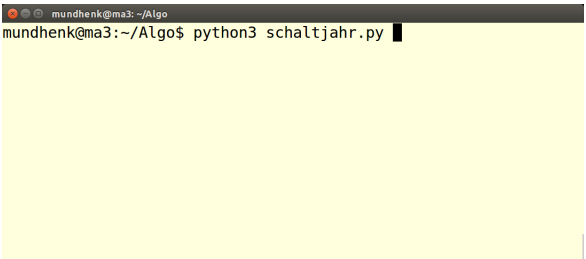
`(False or not False) and False` hat den gleichen Wert wie

`(False or True) and False` hat den gleichen Wert wie

`True and False` hat den Wert `False`.

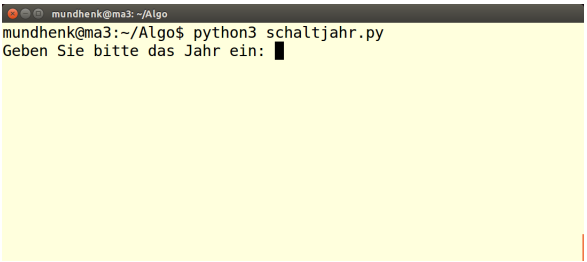
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



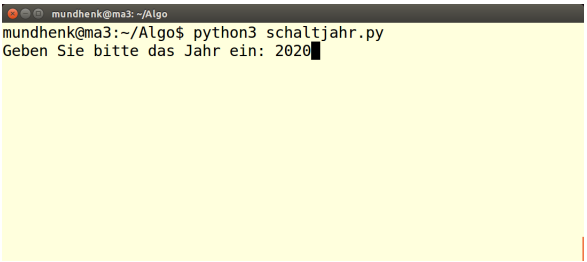
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



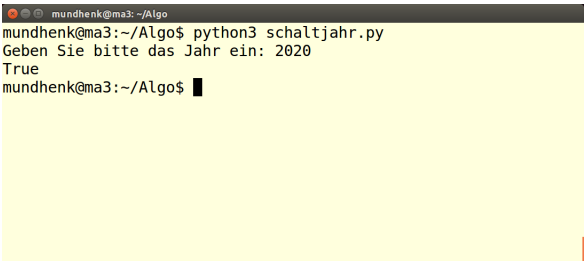
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



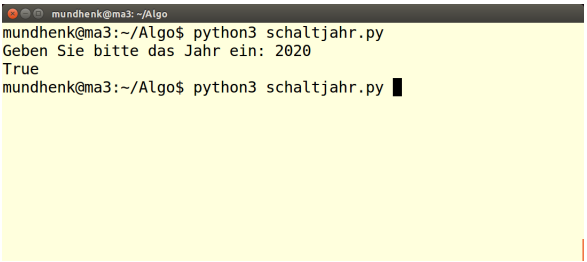
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



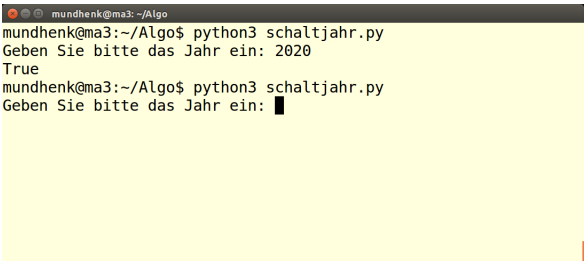
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



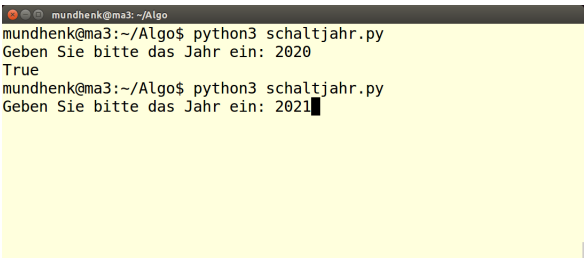
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



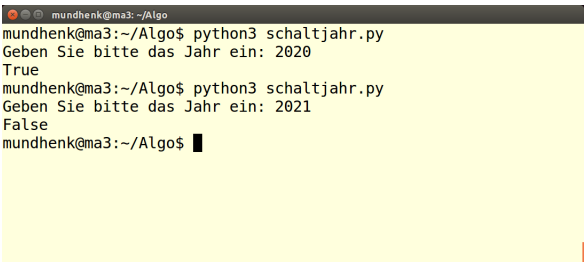
Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



Programmier-Auftrag: bestimme, ob ein Jahr ein Schaltjahr ist

Das Programm erhält ein Jahr als Eingabe
und gibt `True` aus, falls das Jahr ein Schaltjahr ist.
Anderenfalls wird `False` ausgegeben



Die Idee

Schaltjahre werden in unserem Kalender wie folgt bestimmt:

1. Die (ohne Rest) durch 4 teilbaren Jahre sind Schaltjahre, bis auf
2. die (ohne Rest) durch 100 teilbaren Jahre – sie sind keine Schaltjahre.
   Damit sind z.B. 1800, 1900, 2000, 2100 und 2200 keine Schaltjahre.
3. Aber die (ohne Rest) durch 400 teilbaren Jahre sind doch Schaltjahre.
   Damit sind z.B. 1600, 2000 und 2400 doch wieder Schaltjahre.

Jahr x ist also ein Schaltjahr, wenn

- x ist durch 4 teilbar und nicht durch 100 teilbar, oder
- x ist durch 400 teilbar.

Anderenfalls ist Jahr x kein Schaltjahr.

Der grobe Ablauf des Programms

1. Lies das Jahr von der Konsole ein (als `int`-Wert).
2. Berechne die Wahrheitswerte der drei Bedingungen.
   - Bedingung 1: Das Jahr ist durch 4 teilbar. `jahr % 4 == 0`
   - Bedingung 2: Das Jahr ist durch 100 teilbar `jahr % 100 == 0`
   - Bedingung 3: Das Jahr ist durch 400 teilbar `jahr % 400 == 0`
3. Berechne das Ergebnis – es ist eine logische Kombination der drei Bedingungen.
   (Bedingung 1 und nicht Bedingung 2) oder Bedingung 3
4. Gib das Ergebnis aus.

Der grobe Ablauf des Programms

1. Lies das Jahr von der Konsole ein (als `int`-Wert).

2. Berechne die Wahrheitswerte der drei Bedingungen.

   Bedingung 1: Das Jahr ist durch 4 teilbar. `jahr % 4 == 0`

   Bedingung 2: Das Jahr ist durch 100 teilbar `jahr % 100 == 0`

   Bedingung 3: Das Jahr ist durch 400 teilbar `jahr % 400 == 0`

3. Berechne das Ergebnis – es ist eine logische Kombination der drei Bedingungen.

   (Bedingung 1 und nicht Bedingung 2) oder Bedingung 3

4. Gib das Ergebnis aus.

Der grobe Ablauf des Programms

1. Lies das Jahr von der Konsole ein (als `int`-Wert).

2. Berechne die Wahrheitswerte der drei Bedingungen.

Bedingung 1: Das Jahr ist durch 4 teilbar.	`jahr % 4 == 0`
Bedingung 2: Das Jahr ist durch 100 teilbar	`jahr % 100 == 0`
Bedingung 3: Das Jahr ist durch 400 teilbar	`jahr % 400 == 0`

3. Berechne das Ergebnis – es ist eine logische Kombination der drei Bedingungen.

   (Bedingung 1 und nicht Bedingung 2) oder Bedingung 3

4. Gib das Ergebnis aus.

```
#-------------------------------------------------------------------------------
# schaltjahr.py
#-------------------------------------------------------------------------------
# Liest int jahr von der Konsole
# und gibt True aus, wenn das Jahr jahr ein Schaltjahr ist.
# Sonst wird False ausgegeben.
#-------------------------------------------------------------------------------

# Lies das Jahr von der Konsole und speichere es in Variable jahr (Typ int).
jahr = int(input('Geben Sie bitte das Jahr ein: '))

# Bestimme, welche der 3 Bedingungen erfüllt werden
# und speichere die Ergebnisse in den Variablen
# bedingung1, bedingung2 und bedingung3 (alle vom Typ bool).
bedingung1 = (jahr % 4 == 0)      # jahr ist durch 4 teilbar
bedingung2 = (jahr % 100 == 0)    # jahr ist durch 100 teilbar
bedingung3 = (jahr % 400 == 0)    # jahr ist durch 400 teilbar

# Bestimme, ob jahr ein Schaltjahr ist
# und speichere das Ergebnis in Variable istSchaltjahr (Typ bool).
istSchaltjahr = ( bedingung1 and not bedingung2 ) or bedingung3

# Gib das Ergebnis aus.
print(istSchaltjahr)
#-------------------------------------------------------------------------------
```

5 Der Datentyp `float` für Dezimalbrüche

Zum Arbeiten mit gebrochenen Zahlen benutzt man den Datentyp `float`
(*floating point numbers*, Fließkommazahlen, Dezimalbrüche).

Literale sind z.B.

`4.0`	(hat den Wert	4)
`123.45`	(hat den Wert	$123,45$)
`3.141e+6`	(hat den Wert	$3{,}141 \cdot 10^6 = 3141000$)
`12.34e-4`	(hat den Wert	$12{,}34 \cdot 10^{-4} = 0,001234$)

Anders als mit dem Datentyp `int`,
mit dem man (im Prinzip) beliebig große ganze Zahlen (also die Menge $\mathbb{Z}$) darstellen kann,
kann man mit dem Datentyp `float` nur Dezimalbrüche mit einer Vorkommastelle und maximal ca. 17 Stellen sowie Exponenten im Bereich etwa $-308 \ldots 308$ darstellen.
Unsere Vorstellung, dass die reellen Zahlen (die Menge $\mathbb{R}$) genau die Dezimalbrüche sind,
deckt sich also nicht mit dem Datentyp `float`.
Dieser Diskrepanz muss man sich bewusst sein, um Programmfehler zu vermeiden.

Operatoren für den Datentyp `float` sind fast die gleichen wie für `int`.

`+`	Addition
`-`	Subtraktion, Vorzeichen
`*`	Multiplikation
`/`	Division
`**`	Potenz

Der Operator `/` liefert ein Ergebnis mit Datentyp `float`.

Die anderen Operatoren können auf `int` und `float` angewendet werden (z.B. `4.5+6`).
Sie haben ein Ergebnis vom Datentyp `int`, falls beide Argumente Typ `int` haben.
Ihr Ergebnis ist vom Datentyp `float`, wenn mindestens ein Argument den Typ `float` hat.

Beispiele:

`4+2` hat Typ `int`

`4.0+2` und `2+4.2` und `2.3+2.7` haben Typ `float`

Funktionen:

`abs(x)`	der Betrag von `x`
`max(x,y)`, `min(x,y)`	Maximum bzw. Minimum von `x` und `y`
`round(x)`	nächster `int`-Wert zu `x` (in Python3), nächster ganzzahliger `float`-Wert zu `x` (in Python2).
`str(x)`, `int(x)`, `bool(x)`	Umwandlung in Objekt des jeweiligen Typs

Funktionen aus dem Modul `math` von Python:

`math.sqrt(x)`	Wurzelfunktion (*square root*)
`math.log(x)`	natürlicher Logarithmus von `x`
`math.log(x,b)`	Logarithmus von `x` zur Basis `b`

`float(x)` hat als Ergebnis den `float`-Wert, der von String `x` dargestellt wird.

Programmier-Auftrag:
berechne die Lösungen von $a \cdot x^2 + b \cdot x + c = 0$

Das Programm erhält die Werte von a, b und c als Eingabe
und gibt die Werte von x, die die Gleichung erfüllen, als Ergebnis aus.

Die Idee

Das nötige Vorgehen haben wir mal in der Schule gelernt („Mitternachtsformel“):

Die quadratische Gleichung $a \cdot x^2 + b \cdot x + c = 0$ wird von

$$x = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a} \quad \text{und}$$

$$x = \frac{-b - \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

erfüllt.

Der grobe Ablauf des Programmes

1. Lies die Werte von a, b und c von der Konsole ein (als `float`-Werte).
2. Berechne die *Diskriminante* $\sqrt{b^2 - 4 \cdot a \cdot c}$, die beide Lösungen benutzen.
3. Gib die beiden Ergebnisse aus – sie werden „beim Ausgeben“ berechnet.

```
(-b + diskriminante) / (2*a)
(-b - diskriminante) / (2*a)
```

Der grobe Ablauf des Programmes

1. Lies die Werte von a, b und c von der Konsole ein (als `float`-Werte).
2. Berechne die *Diskriminante* $\sqrt{b^2 - 4 \cdot a \cdot c}$, die beide Lösungen benutzen.
3. Gib die beiden Ergebnisse aus – sie werden „beim Ausgeben“ berechnet.

```
(-b + diskriminante) / (2*a)
(-b - diskriminante) / (2*a)
```

```python
# mitternacht.py
#-------------------------------------------------------------------------
# Liest float-Werte a, b und c von der Konsole ein
# und gibt die beiden durch die Mitternachtsformel bestimmten Lösungen aus.
#-------------------------------------------------------------------------

import math

print('Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.')
a = float(input('  a: '))
b = float(input('  b: '))
c = float(input('  c: '))

diskriminante = math.sqrt(b**2 - 4 * a * c)

print('Die Lösungen sind:')
print( (-b + diskriminante) / (2*a) )
print( (-b - diskriminante) / (2*a) )

#-------------------------------------------------------------------------
# python3 mitternacht.py
# Geben Sie die Werte von a, b und c für die Mitternachtsformel ein.
#   a: 4
#   b: 15
#   ...
```

- `import math` erlaubt die Benutzung von Funktionen aus dem `math`-Modul – hier `math.sqrt()`
- alle anderen Programmzeilen enthalten Funktionsaufrufe
- die Argumente sind häufig nicht nur Variablen sondern Ausdrücke
- bei einigen Eingaben gibt das Programm kein Ergebnis, sondern *stürzt ab*

Zusammenfassung

- Wir haben die grundlegende Daten-Typen `int`, `float`, `str` und `bool` von Python kennengelernt.
- Wir können Ausdrücke aus Literalen, Variablen, Operatoren und Funktionen schreiben und deren Typen bestimmen.
- Wir können Programme mit Variablen der grundlegenden Daten-Typen und
  Eingabe von Argumenten (von der Tastatur),
  Abarbeitung einer festen Folge von Anweisungen und
  Ausgabe von Ergebnissen (auf dem Bildschirm)
  schreiben und deren Ausführung nachvollziehen.