2 Funktionen und Module

Funktionen erlauben (mehr als Verzweigungen und Schleifen) den Programmfluss zwischen verschiedenen Stellen des Programmcodes hin und her springen zu lassen. Sie machen es möglich, den gleichen Programmcode an verschiedenen Stellen des Programms wiederzubenutzen.
Schließlich kann man Funktionen in *Module* auslagern, so dass sie in verschiedenen Programmen (*Klienten*) benutzt werden können.
Die Möglichkeit, Funktionen *rekursiv* zu definieren, liefert eine neue Sicht auf Strukturen in Problemen, die man mit Programmen lösen will.

Vorlesung 07

2.1 Funktionen programmieren

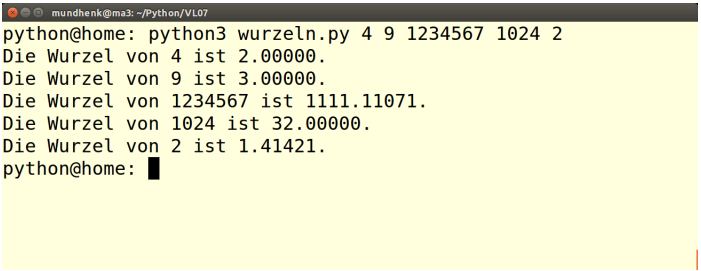
Wir haben bereits verschiedene Funktionen benutzt:

```
max(a,b)
int(sys.argv[1])
print( '%d %.2f' % (jahr, t) )
math.sqrt(5)
random.randrange(1,7)
range(len(a))
stddraw.line(1,2, 0.1,0.2)
```

Aufgaben eines Programms, die klar abgegrenzt sind,
sollte man auch abgrenzen und z.B. als Funktion aufschreiben.

Dadurch bekommt der Programmcode eine bessere Struktur.
Entwicklung, Fehlersuche, Wartung und Wiederbenutzung werden einfacher.

1 Einleitung:
lies Zahlen ein und gib deren Quadratwurzeln aus

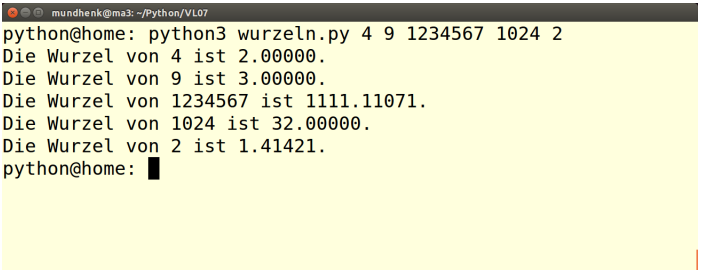


Grobe Programm-Struktur:

1. Die Eingabewerte stehen in der Kommandozeile.
2. Gib für jeden Eingabewert die Quadratwurzel aus.

```
import sys, math

for i in range(1,len(sys.argv)):
  z = float(sys.argv[i])
  w = math.sqrt(z)
  print('Die Wurzel von %.f ist %.5f.' % (z,w))
```

1 Einleitung:
lies Zahlen ein und gib deren Quadratwurzeln aus



Grobe Programm-Struktur:

1. Die Eingabewerte stehen in der Kommandozeile.
2. Gib für jeden Eingabewert die Quadratwurzel aus.

```
import sys, math

for i in range(1,len(sys.argv)):
  z = float(sys.argv[i])
  w = math.sqrt(z)
  print('Die Wurzel von %.f ist %.5f.' % (z,w))
```

Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
```

```
import sys, math

for i in range(1,len(sys.argv)):

    z = float(sys.argv[i])

    w = math.sqrt(z)

    print('Die Wurzel von %.2f ist %.5f.' % (z,w))
```

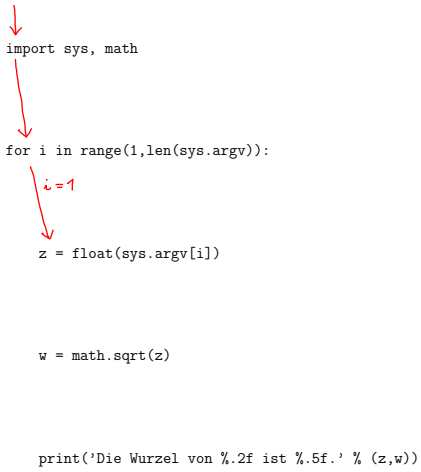
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
```

```
import sys, math

for i in range(1,len(sys.argv)):

    z = float(sys.argv[i])

    w = math.sqrt(z)

    print('Die Wurzel von %.2f ist %.5f.' % (z,w))
```

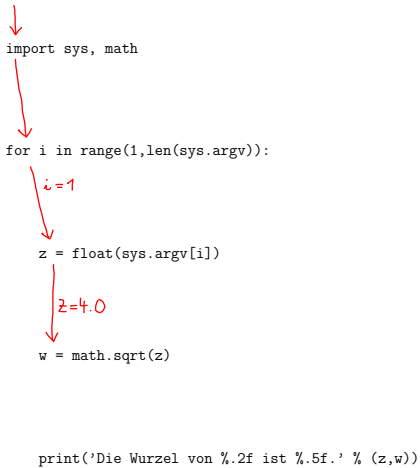
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
```



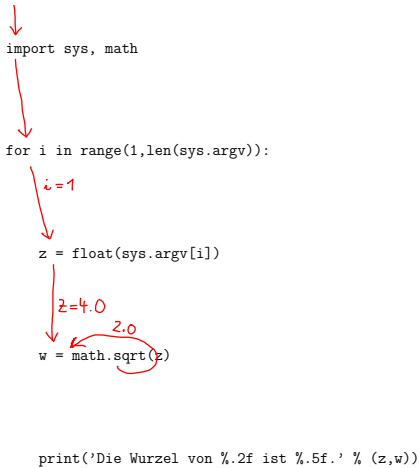
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
```



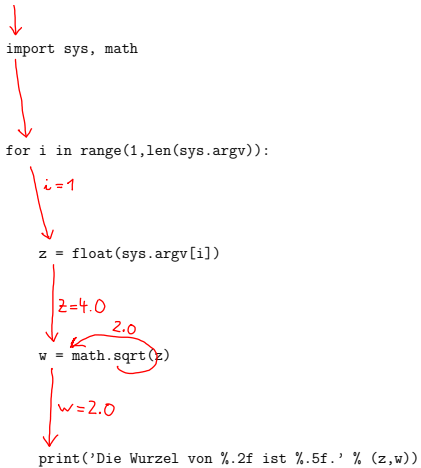
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
```



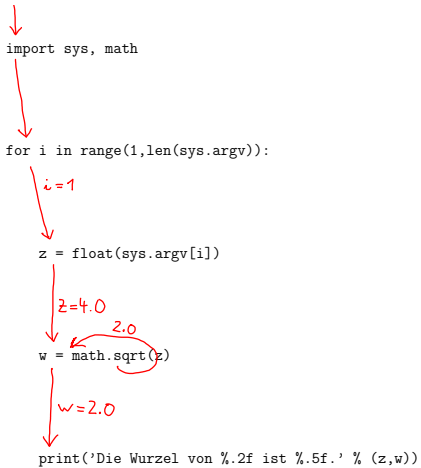
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
```



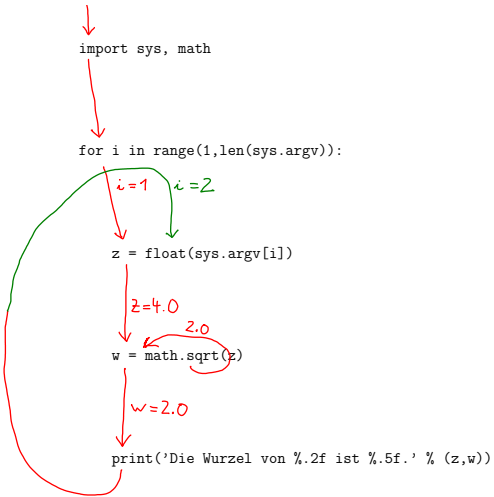
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
```



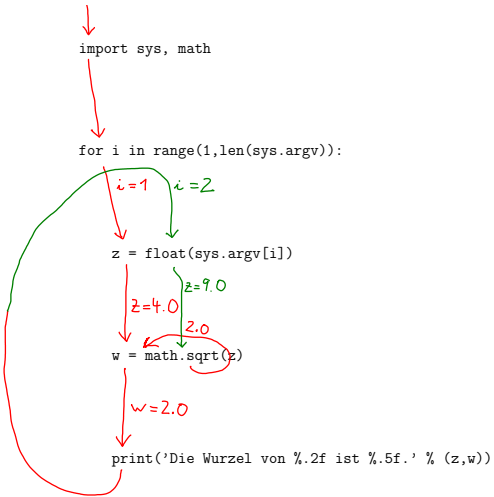
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
```



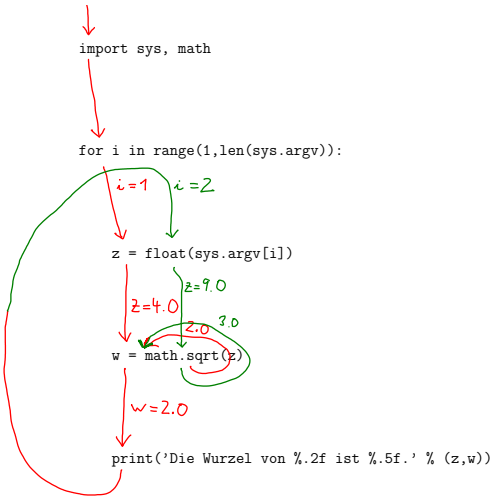
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
```



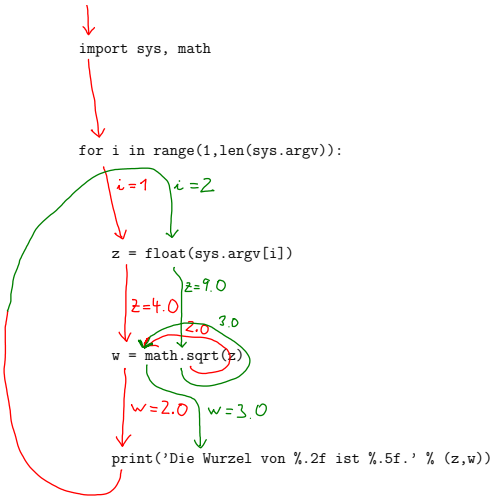
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
```



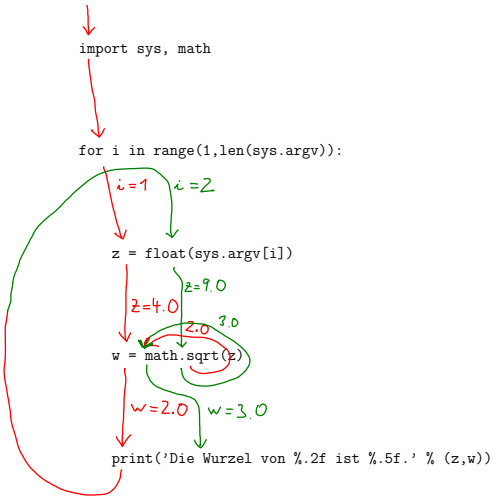
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
```



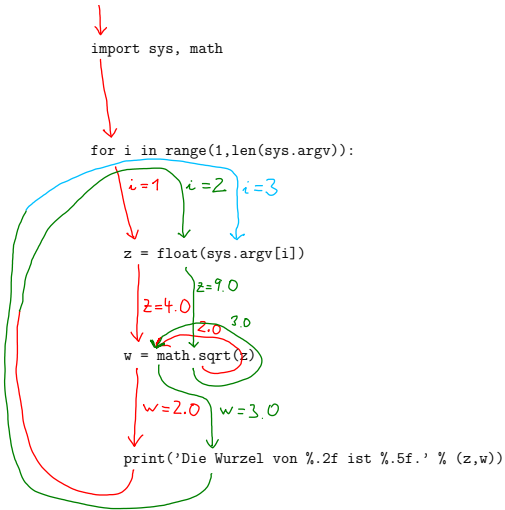
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
Die Wurzel von 9.00 ist 3.00000.
```



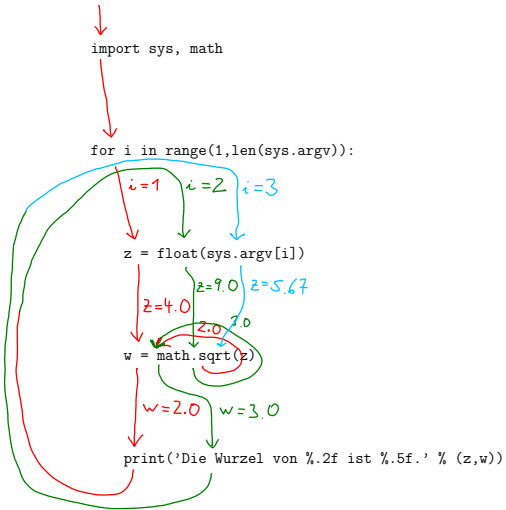
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
Die Wurzel von 9.00 ist 3.00000.
```



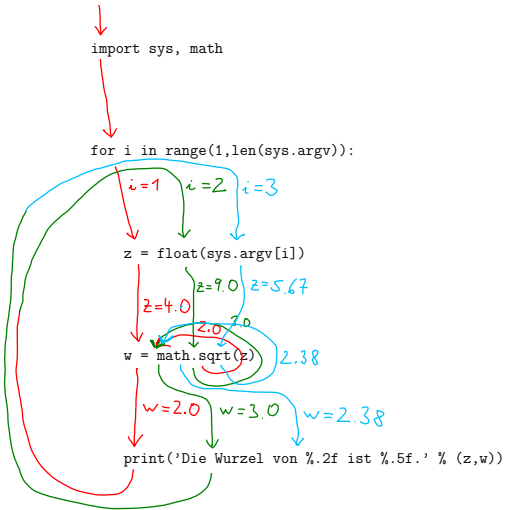
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
Die Wurzel von 9.00 ist 3.00000.
```



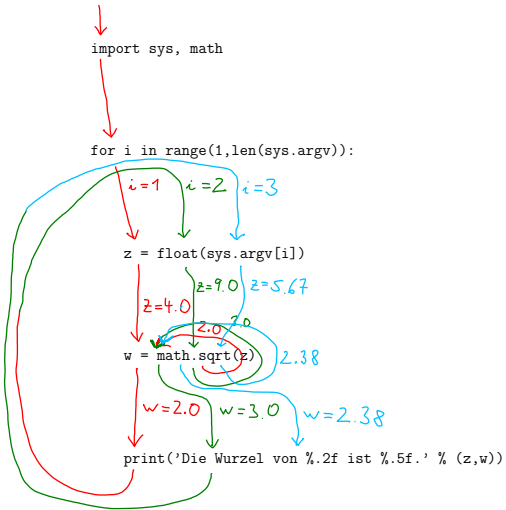
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
Die Wurzel von 9.00 ist 3.00000.
```



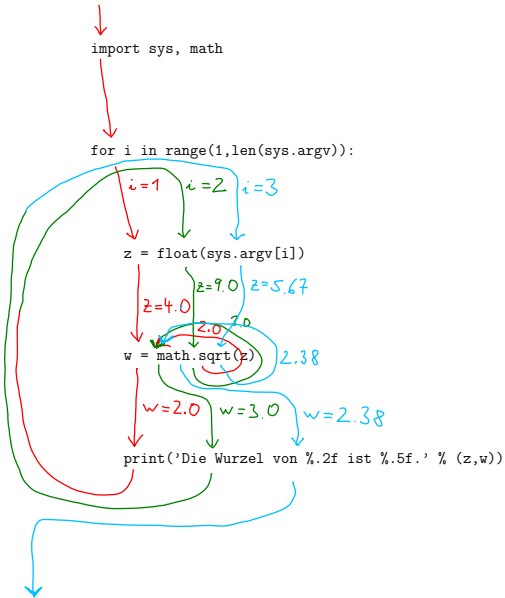
Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
Die Wurzel von 9.00 ist 3.00000.
Die Wurzel von 5.67 ist 2.38117.
```



Die grobe Vorstellung, wie das Programm abläuft

```
python3 wurzeln.py 4 9 5.67
Die Wurzel von 4.00 ist 2.00000.
Die Wurzel von 9.00 ist 3.00000.
Die Wurzel von 5.67 ist 2.38117.
```



Eine kleine Überraschung ...

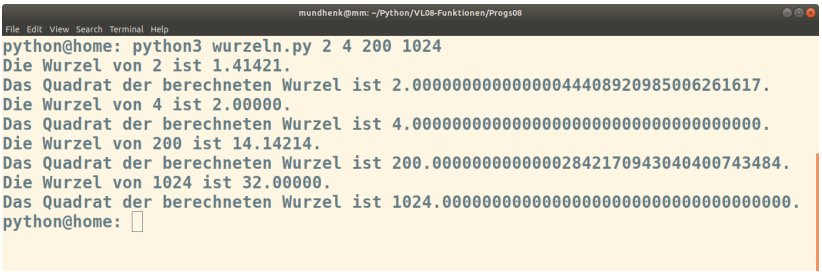
Wir bauen einen Test des Ergebnisses in das Programm ein.

```
import sys, math

for i in range(1,len(sys.argv)):
  z = float(sys.argv[i])
  w = math.sqrt(z)
  print('Die Wurzel von %.f ist %.5f.' % (z,w))
  print('Das Quadrat der berechneten Wurzel ist %.35f.' % (w*w))
```

Eine kleine Überraschung . . .

Wir bauen einen Test des Ergebnisses in das Programm ein.

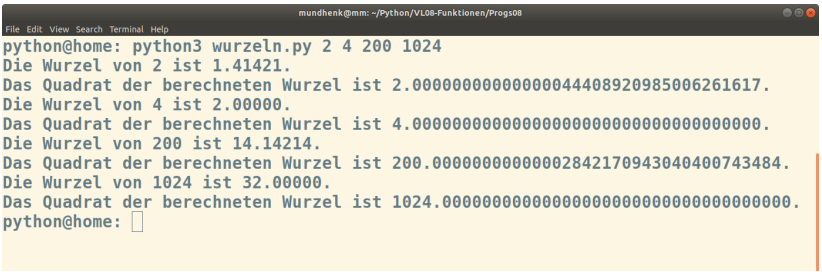
```
import sys, math

for i in range(1,len(sys.argv)):
  z = float(sys.argv[i])
  w = math.sqrt(z)
  print('Die Wurzel von %.f ist %.5f.' % (z,w))
  print('Das Quadrat der berechneten Wurzel ist %.35f.' % (w*w))
```



Eine kleine Überraschung . . .

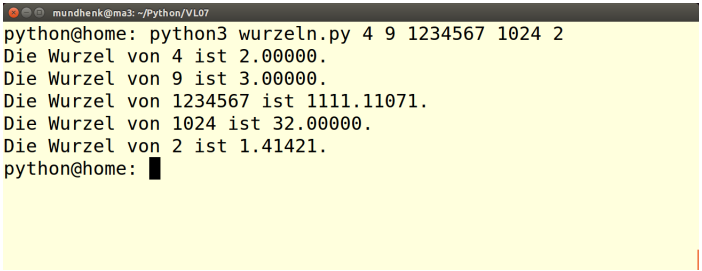
Wir bauen einen Test des Ergebnisses in das Programm ein.

```
import sys, math

for i in range(1,len(sys.argv)):
  z = float(sys.argv[i])
  w = math.sqrt(z)
  print('Die Wurzel von %.f ist %.5f.' % (z,w))
  print('Das Quadrat der berechneten Wurzel ist %.35f.' % (w*w))
```



Die Ergebnisse sind nicht alle mathematisch korrekt!

Was wollen wir machen?

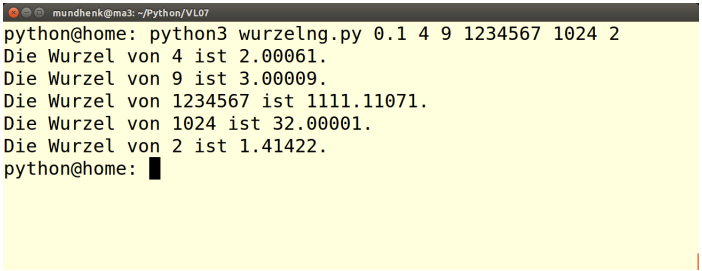


Wir wollen die Funktion zur Berechnung der Quadratwurzel selber programmieren.

Wir werden die Genauigkeit, mit der die Quadratwurzeln berechnet werden, mit in die Eingabe aufnehmen

und uns vorher noch anschauen, warum das „sinnvoll“ ist.

Was wollen wir machen?

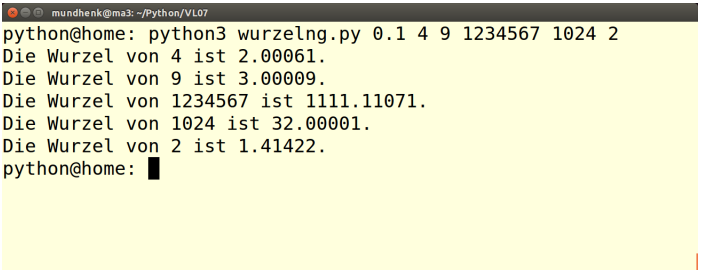


Wir wollen die Funktion zur Berechnung der Quadratwurzel selber programmieren.

Wir werden die Genauigkeit, mit der die Quadratwurzeln berechnet werden, mit in die Eingabe aufnehmen

und uns vorher noch anschauen, warum das „sinnvoll“ ist.

Was wollen wir machen?



Es folgt:

- Welche Zahlen können als `float` verarbeitet werden – oder: was ist `float`?
- Wie kann man Quadratwurzeln berechnen (mit vorgegebener Genauigkeit)?
- Wie programmiert man eine Funktion (zur Berechnung der Quadratwurzel)?

1 Was ist `float`?

- Darstellung von positiven Dezimalzahlen als Binärzahlen.
- Darstellung von positiven Dezimalbrüchen als Binärbrüche.
- Binärbrüche mit beschränkter Genauigkeit und die Norm dafür.

Dezimalzahlen sind Summen von 10er-Potenzen

Wir stellen Zahlen als *Dezimalzahlen* im 10er-System mit 10 Ziffern $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ dar.
Zum Beispiel ist 25404 eine Dezimalzahl.

Jede Ziffer einer Dezimalzahl hat einen durch ihre Stelle bestimmten Wert
und liefert einen Summanden zum Wert der gesamten Zahl.

Als Beispiel betrachten wir die Dezimalzahl 25404:

Ziffer:	2	5	4	0	4
Stellenwert:	Zehntausender 10^4	Tausender 10^3	Hunderter 10^2	Zehner 10^1	Einer 10^0
Wert der Zahl:	$2 \cdot 10^4$	$+ \; 5 \cdot 10^3$	$+ \; 4 \cdot 10^2$	$+ \; 0 \cdot 10^1$	$+ \; 4 \cdot 10^0 \; = \; 25404$

Binärzahlen sind Summen von 2er-Potenzen

Binärzahlen bestehen nur aus den beiden Ziffern 0 und 1.
Zum Beispiel ist 100110100 eine Binärzahl.

Jede Ziffer einer Binärzahl hat einen durch ihre Stelle bestimmten Wert und liefert einen Summanden zum Wert der gesamten Zahl.

Als Beispiel betrachten wir die Binärzahl 10110:

Ziffer:	1	0	1	1	0
Stellenwert:	2^4	2^3	2^2	2^1	2^0
	16	8	4	2	1
Wert der Zahl:	$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 22$				

Damit ist bereits klar: jede Binärzahl kann man auch als Dezimalzahl darstellen.

Jede (positive) Dezimalzahl lässt sich als Binärzahl darstellen

Beispiele:

$$\begin{aligned} 46 &= 32+8+4+2 &&= 2^5+2^3+2^2+2^1 \\ &&&= 1\cdot 2^5+0\cdot 2^4+1\cdot 2^3+1\cdot 2^2+1\cdot 2^1+0\cdot 2^0 \\ &&&\mathrel{\hat{=}} 101110 \end{aligned}$$

$$\begin{aligned} 139 &= 128+8+2+1 &&= 2^7+2^3+2^1+2^0 \\ &&&= 1\cdot 2^7+0\cdot 2^6+0\cdot 2^5+0\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0 \\ &&&\mathrel{\hat{=}} 10001011 \end{aligned}$$

Die Darstellung einer Zahl n als Binärzahl ist

eine Folge $b_m b_{m-1} \ldots b_1 b_0$ (mit $b_i \in \{0,1\}$) mit der Eigenschaft $n = \sum_{i=0}^{m} 2^i \cdot b_i$.

Dezimalbrüche sind Summen von 10er-Potenzen

Ein Dezimalbruch ist z.B. 2134,79402 oder 26.56241 .

Jede Ziffer eines Dezimalbruchs hat einen durch ihre Stelle bestimmten Wert und liefert einen Summanden zum Wert der gesamten Zahl.

Als Beispiel betrachten wir den Dezimalbruch 13,567:

Ziffer:	1	3	,	5	6	7
Stellenwert:	Zehner	Einer				
	10^1	10^0		10^{-1}	10^{-2}	10^{-3}
Wert der Zahl:	$1 \cdot 10^1$	$+ 3 \cdot 10^0$		$+ 5 \cdot 10^{-1}$	$+ 6 \cdot 10^{-2}$	$+ 7 \cdot 10^{-3}$

Es gibt Zahlen, die man nur durch unendliche Dezimalbrüche darstellen kann – z.B. $\frac{1}{3}$ und $\sqrt{2}$.

Binärbrüche sind Summen von 2er-Potenzen

Ein Binärbruch ist z.B. 1011,001101 oder 0.110100 .

Jede Ziffer eines Binärbruchs hat einen durch ihre Stelle bestimmten Wert und liefert einen Summanden zum Wert der gesamten Zahl.

Als Beispiel betrachten wir den Binärbruch 10,101:

Ziffer:	1	1	0	,	1	0	1	
Stellenwert:	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	
Wert der Zahl:	$1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$							$= 4{,}625$

Jeder Binärbruch lässt sich durch einen Dezimalbruch darstellen.

Nicht jeder Dezimalbruch lässt sich als Binärbruch darstellen

Wir wollen 0,6 als Binärbruch darstellen:

Zahl:	0	,	1	0	0	1	1	0	0	1	⋯
Stellenwert:	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-7}	⋯
Wert der Zahl:			$\frac{1}{2}$ +			$\frac{1}{16}$ +	$\frac{1}{32}$ +			$\frac{1}{256}$ +	⋯

Problem:

der Dezimalbruch 0,6 hat eine *unendlich* lange Darstellung als Binärbruch.

Das gilt auch für viele andere Dezimalbrüche.

`float` stellt Binärbrüche mit einer festgelegten Stellenzahl dar.

Dadurch wird z.B. der Dezimalbruch 0,6 mit `float` als 0,599999999999999977795... dargestellt.

Normen für die Darstellung von Dezimalbrüchen

ISO/IEC/IEEE 60559:2011 und IEEE 754

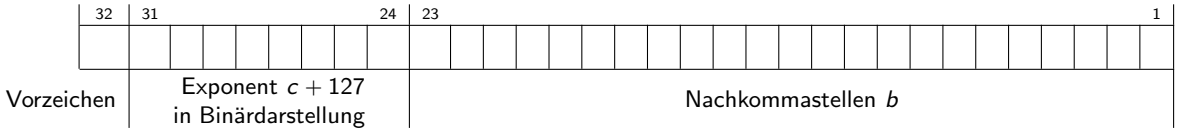
Die Darstellung orientiert sich an der „wissenschaftlichen Notation“.
Dezimalbrüche schreibt man in der Form $\pm a,b \cdot 10^c$, wobei a einstellig und ungleich 0 ist.

Beispiele:

Wert	wiss. Notation
$23,456$	$2,3456 \cdot 10^1$
1234	$1,234 \cdot 10^3$
$0,1234$	$1,234 \cdot 10^{-1}$
$-0,007$	$-7 \cdot 10^{-3}$

In der Binärschreibweise ist die entsprechende Form $\pm a,b \cdot 2^c$, wobei a die 1 ist.
Also muss man nur das Vorzeichen, die Nachkommastellen b und den Exponenten c darstellen.

Die Binärzahl $\pm 1, b \cdot 2^c$ wird in *single precision* mit 32 Bit wie folgt dargestellt.

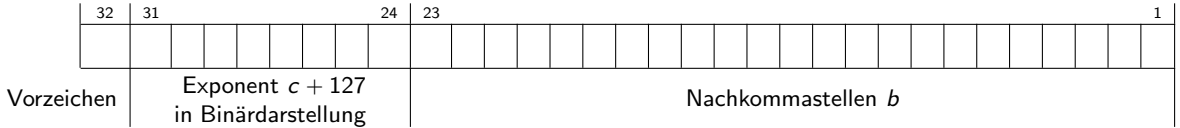


- Das Vorzeichen ist 0 für $+$ und 1 für $-$.
- Die Darstellung des Exponenten braucht wegen des *bias* 127 kein Vorzeichen.
- Wenn alle Exponenten-Bits 0 sind, wird die Zahl $\pm 0, b \cdot 2^{-126}$ dargestellt.
- Es gibt spezielle Werte `float('inf')` und `float('-inf')` für „unendlich“.
  Dabei sind alle Exponenten-Bits 1 und b besteht nur aus 0en.

Die Darstellung mit *double precision* benutzt 64 Bit.
Der Exponent hat 11 Bit (bias 1023), und b hat 52 Bit.

Pythons `float` benutzt (üblicherweise) die Darstellung in double precision.

Die Binärzahl $\pm 1, b \cdot 2^c$ wird in *single precision* mit 32 Bit wie folgt dargestellt.

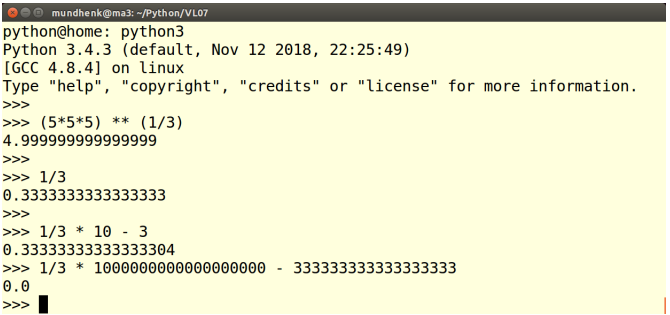


- Das Vorzeichen ist 0 für $+$ und 1 für $-$.
- Die Darstellung des Exponenten braucht wegen des *bias* 127 kein Vorzeichen.
- Wenn alle Exponenten-Bits 0 sind, wird die Zahl $\pm 0, b \cdot 2^{-126}$ dargestellt.
- Es gibt spezielle Werte `float('inf')` und `float('-inf')` für „unendlich“.
  Dabei sind alle Exponenten-Bits 1 und b besteht nur aus 0en.

Die Darstellung mit *double precision* benutzt 64 Bit.
Der Exponent hat 11 Bit (bias 1023), und b hat 52 Bit.

Pythons `float` benutzt (üblicherweise) die Darstellung in double precision.

- Der Datentyp `int` kann „beliebig große“ ganze Zahlen verarbeiten.
- Der Datentyp `float` verarbeitet Zahlen mit beschränkter Präzision.
  Abweichungen, die zu den mathematisch korrekten Zahlenwerten bestehen,
  nennt man *Rundungsfehler*.
- Wenn man mit `float`-Werten rechnet, muss man immer mit Rundungsfehlern rechnen.
- Rundungsfehler können Rechenergebnisse katastrophal verfälschen.



3 Programmier-Auftrag: schreibe eine Funktion zur Berechnung der Quadratwurzel (wie `math.sqrt()`)

Wir haben gerade gesehen, dass man den Wert nicht genau berechnen kann.

Die Funktion, die wir programmieren werden, soll mit zwei Argumenten n und r aufgerufen werden.
Sie berechnet einen Näherungswert w von $\sqrt{n}$,

der höchstens r „neben“ $\sqrt{n}$ liegt.

Es soll $\sqrt{n} - r \leqslant w \leqslant \sqrt{n} + r$ gelten.

Die Idee: der Heron-Algorithmus

Beispiel: Berechnung der Wurzel von 7 mit Genauigkeit $r = 0.001$.

Wir beginnen mit einem Rechteck mit Fläche 7 und Seitenlängen 1 und 7.

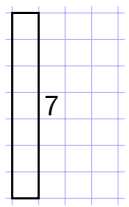
Wir wissen: $1 \leqslant \sqrt{7} \leqslant 7$.

Ziel: ein Quadrat mit Fläche 7. Dann ist die Seitenlänge $\sqrt{7}$.

Vorgehen: solange der Unterschied der Seitenlängen zu groß ist,
dann verkürze die längere Seite auf den Mittelwert der beiden Seitenlängen.

Die Länge der kürzeren Seite wird dann so gewählt, dass das Rechteck Fläche 7 hat.

Wir wissen: kürzere Seite $\leqslant \sqrt{7} \leqslant$ längere Seite.



Das Ergebnis ist $(2.64576 + 2.64573)/2 = 2.645745$

Die Idee: der Heron-Algorithmus

Beispiel: Berechnung der Wurzel von 7 mit Genauigkeit $r = 0.001$.

Wir beginnen mit einem Rechteck mit Fläche 7 und Seitenlängen 1 und 7.

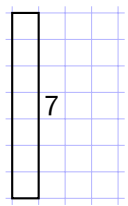
Wir wissen: $1 \leqslant \sqrt{7} \leqslant 7$.

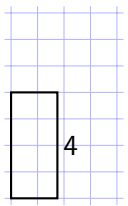
Ziel: ein Quadrat mit Fläche 7. Dann ist die Seitenlänge $\sqrt{7}$.

Vorgehen: solange der Unterschied der Seitenlängen zu groß ist,
dann verkürze die längere Seite auf den Mittelwert der beiden Seitenlängen.

Die Länge der kürzeren Seite wird dann so gewählt, dass das Rechteck Fläche 7 hat.

Wir wissen: kürzere Seite $\leqslant \sqrt{7} \leqslant$ längere Seite.





Das Ergebnis ist $(2.64576 + 2.64573)/2 = 2.645745$

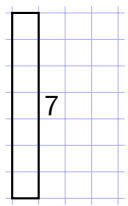
Die Idee: der Heron-Algorithmus

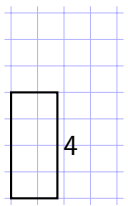
Beispiel: Berechnung der Wurzel von 7 mit Genauigkeit $r = 0.001$.
Wir beginnen mit einem Rechteck mit Fläche 7 und Seitenlängen 1 und 7.

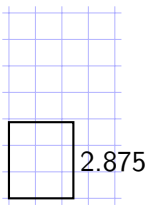
Wir wissen: $1 \leqslant \sqrt{7} \leqslant 7$.

Ziel: ein Quadrat mit Fläche 7. Dann ist die Seitenlänge $\sqrt{7}$.
Vorgehen: solange der Unterschied der Seitenlängen zu groß ist,
dann verkürze die längere Seite auf den Mittelwert der beiden Seitenlängen.
Die Länge der kürzeren Seite wird dann so gewählt, dass das Rechteck Fläche 7 hat.

Wir wissen: kürzere Seite $\leqslant \sqrt{7} \leqslant$ längere Seite.







Das Ergebnis ist $(2.64576 + 2.64573)/2 = 2.645745$.

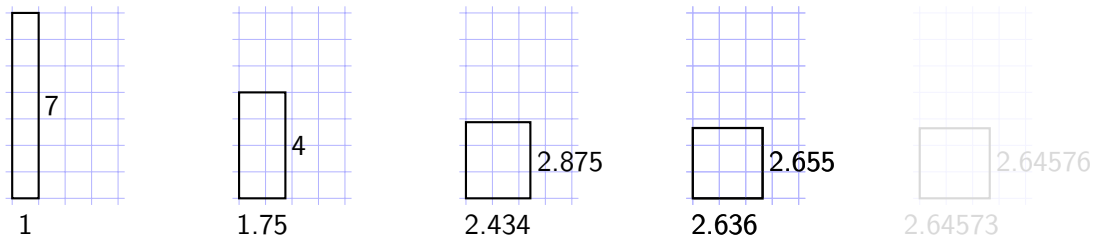
Die Idee: der Heron-Algorithmus

Beispiel: Berechnung der Wurzel von 7 mit Genauigkeit $r = 0.001$.
Wir beginnen mit einem Rechteck mit Fläche 7 und Seitenlängen 1 und 7.

Wir wissen: $1 \leqslant \sqrt{7} \leqslant 7$.

Ziel: ein Quadrat mit Fläche 7. Dann ist die Seitenlänge $\sqrt{7}$.
Vorgehen: solange der Unterschied der Seitenlängen zu groß ist,
dann verkürze die längere Seite auf den Mittelwert der beiden Seitenlängen.
Die Länge der kürzeren Seite wird dann so gewählt, dass das Rechteck Fläche 7 hat.

Wir wissen: kürzere Seite $\leqslant \sqrt{7} \leqslant$ längere Seite.



Das Ergebnis ist $(2.64576 + 2.64573)/2 = 2.645745$.

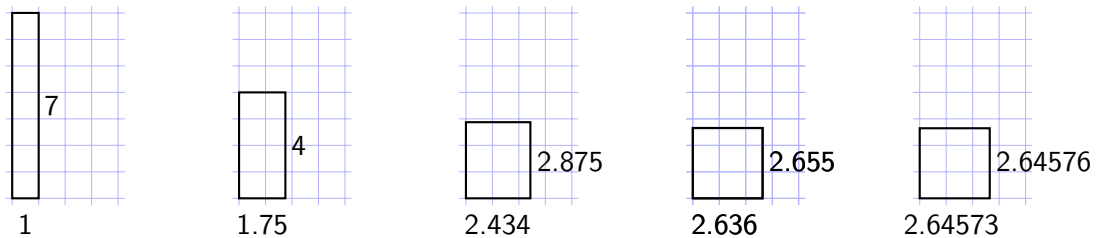
Die Idee: der Heron-Algorithmus

Beispiel: Berechnung der Wurzel von 7 mit Genauigkeit $r = 0.001$.
Wir beginnen mit einem Rechteck mit Fläche 7 und Seitenlängen 1 und 7.

Wir wissen: $1 \leqslant \sqrt{7} \leqslant 7$.

Ziel: ein Quadrat mit Fläche 7. Dann ist die Seitenlänge $\sqrt{7}$.
Vorgehen: solange der Unterschied der Seitenlängen zu groß ist,
dann verkürze die längere Seite auf den Mittelwert der beiden Seitenlängen.
Die Länge der kürzeren Seite wird dann so gewählt, dass das Rechteck Fläche 7 hat.

Wir wissen: kürzere Seite $\leqslant \sqrt{7} \leqslant$ längere Seite.



Das Ergebnis ist $(2.64576 + 2.64573)/2 = 2.645745$.

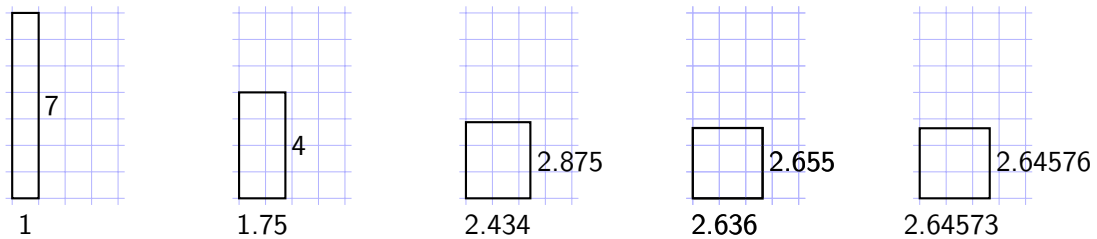
Die Idee: der Heron-Algorithmus

Beispiel: Berechnung der Wurzel von 7 mit Genauigkeit $r = 0.001$.
Wir beginnen mit einem Rechteck mit Fläche 7 und Seitenlängen 1 und 7.

Wir wissen: $1 \leqslant \sqrt{7} \leqslant 7$.

Ziel: ein Quadrat mit Fläche 7. Dann ist die Seitenlänge $\sqrt{7}$.
Vorgehen: solange der Unterschied der Seitenlängen zu groß ist,
dann verkürze die längere Seite auf den Mittelwert der beiden Seitenlängen.
Die Länge der kürzeren Seite wird dann so gewählt, dass das Rechteck Fläche 7 hat.

Wir wissen: kürzere Seite $\leqslant \sqrt{7} \leqslant$ längere Seite.



Das Ergebnis ist $(2.64576 + 2.64573)/2 = 2.645745$.

Die grobe Programm-Struktur (noch nicht als Funktion)

1. Lies n und r.

2. Die Seitenlängen sind n und 1.

3. Solange der Unterschied der Seitenlängen größer als $2 \cdot r$ ist:

3.1 berechne die neuen Seitenlängen:
Die Länge der längeren Seite ist der Mittelwert der beiden Seitenlängen.
Die kürzere ist so, dass das Produkt der beiden Seitenlängen n ergibt.

4. Gib den Mittelwert der beiden Seitenlängen aus.

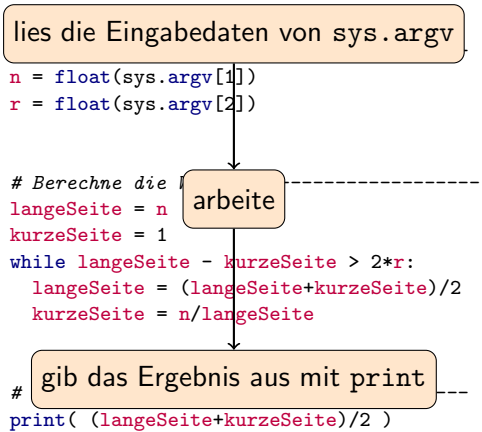
Das Programm vorbereitung_heron.py

```
# vorbereitung_heron.py
#-------------------------------------------------------------------------------------------
# Implementierung des Algorithmus von Heron.
# Er liest zwei Zahlen n und r und berechnet die Quadratwurzel von n mit einem Fehler <= r.
#-------------------------------------------------------------------------------------------
import sys
# Lies die Eingabwerte. ------------------------------------------------------------------
n = float(sys.argv[1])
r = float(sys.argv[2])
# Die Seitenlängen am Anfang sind n und 1. -----------------------------------------------
langeSeite = n
kurzeSeite = 1
# Solange der Unterschied der Seitenlängen > 2r ist,
# werden sie dichter zusammengebracht, ohne die Fläche des Rechtecks zu ändern.
while langeSeite - kurzeSeite > 2*r:
  # Die lange Seite wird zum Mittelwert der beiden Seitenlängen.
  langeSeite = (langeSeite+kurzeSeite)/2
  # Die kurze Seite wird angepasst, so dass das Produkt mit der langen Seite n ist.
  kurzeSeite = n/langeSeite
# Das Ergebnis wird ausgegeben. ----------------------------------------------------------
print( (langeSeite+kurzeSeite)/2 )
```

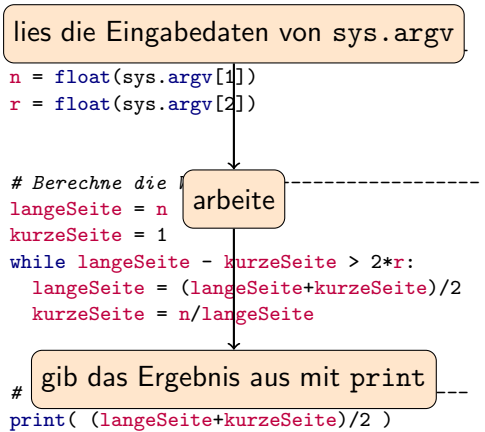
Ein Programm
mit Ein- und Ausgabe über die Konsole:

```
# vorbereitung_heron.py
# Lies die Eingabwerte. -----------------
n = float(sys.argv[1])
r = float(sys.argv[2])


# Berechne die Wurzel. -------------------
langeSeite = n
kurzeSeite = 1
while langeSeite - kurzeSeite > 2*r:
  langeSeite = (langeSeite+kurzeSeite)/2
  kurzeSeite = n/langeSeite


# Gib das Ergebnis aus. ------------------
print( (langeSeite+kurzeSeite)/2 )
```

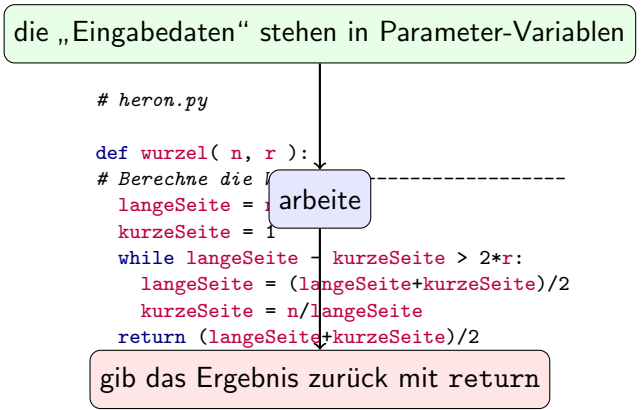
Ein Programm
mit Ein- und Ausgabe über die Konsole:



Ein Programm
mit Ein- und Ausgabe über die Konsole:



Eine Funktion,
die mit ihrer Aufrufstelle kommuniziert:



Ein Programm
mit Ein- und Ausgabe über die Konsole:

```
# vorbereitung_heron.py
# Lies die Eingabwerte. -----------------
n = float(sys.argv[1])
r = float(sys.argv[2])


# Berechne die Wurzel. -------------------
langeSeite = n
kurzeSeite = 1
while langeSeite - kurzeSeite > 2*r:
  langeSeite = (langeSeite+kurzeSeite)/2
  kurzeSeite = n/langeSeite


# Gib das Ergebnis aus. ------------------
print( (langeSeite+kurzeSeite)/2 )
```

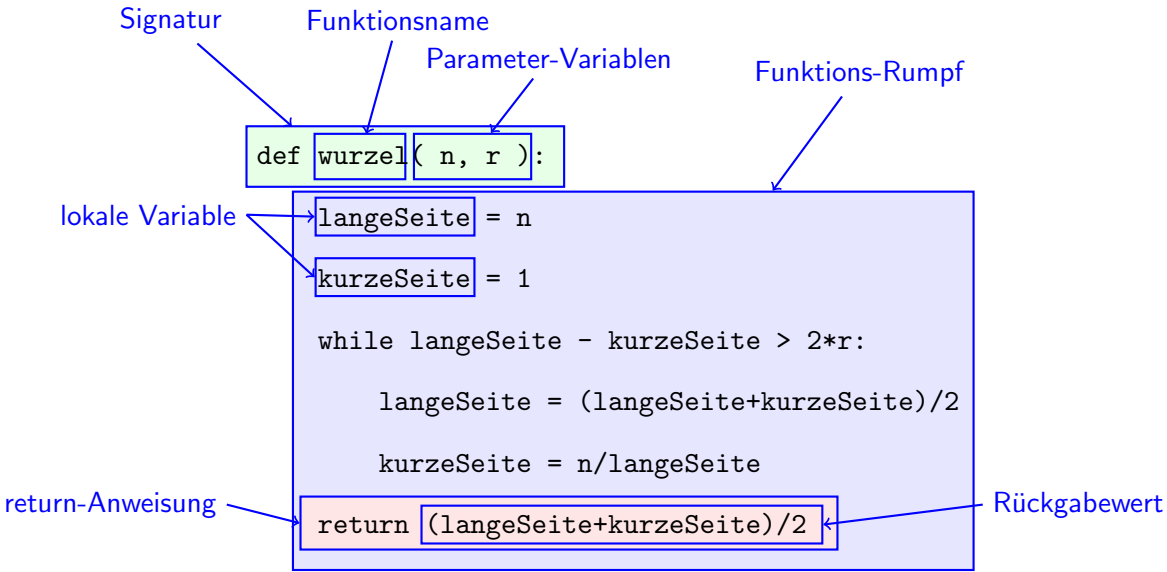
Eine Funktion,
die mit ihrer Aufrufstelle kommuniziert:

```
# heron.py

def wurzel( n, r ):
# Berechne die Wurzel. -------------------
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
```

Definition einer Funktion

```
def wurzel( n, r ):
    langeSeite = n
    kurzeSeite = 1
    while langeSeite - kurzeSeite > 2*r:
        langeSeite = (langeSeite+kurzeSeite)/2
        kurzeSeite = n/langeSeite
    return (langeSeite+kurzeSeite)/2
```

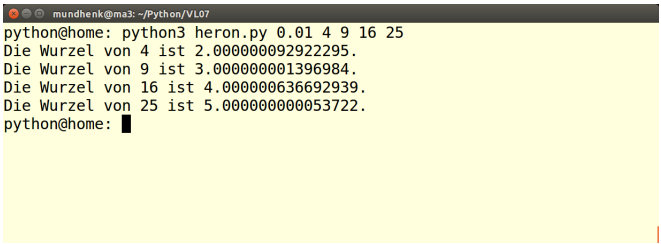
Definition einer Funktion



Benutzung der Funktion: berechne die Wurzeln aller Eingaben

Das Programm liest die Genauigkeit r und Zahlen $n_1, \ldots, n_k$ von der Kommandozeile.

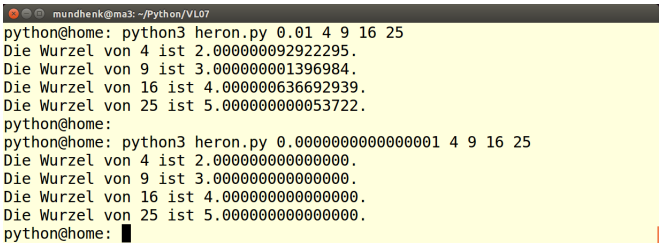
Es gibt die Wurzeln von $n_1, \ldots, n_k$ mit der angegebenen Genauigkeit aus.



Benutzung der Funktion: berechne die Wurzeln aller Eingaben

Das Programm liest die Genauigkeit r und Zahlen $n_1, \ldots, n_k$ von der Kommandozeile.

Es gibt die Wurzeln von $n_1, \ldots, n_k$ mit der angegebenen Genauigkeit aus.



Die grobe Programm-Struktur

1. Die Eingabedaten werden von der Kommandozeile gelesen.
2. Für jede eingegebene Zahl n:
   - **2.1** berechne die Wurzel von n mit der angegebenen Genauigkeit und gib das Ergebnis aus

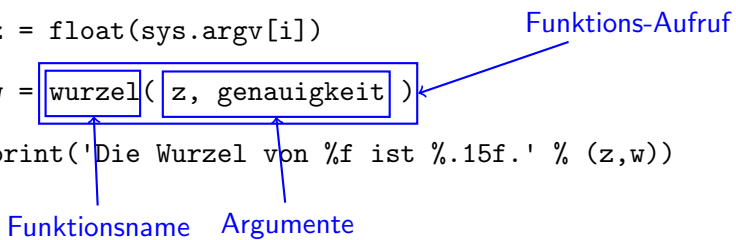
Aufruf einer Funktion

```
genauigkeit = float(sys.argv[1])

for i in range(2,len(sys.argv)):

  z = float(sys.argv[i])

  w = wurzel( z, genauigkeit )

  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

Funktions-Aufruf

Aufruf einer Funktion

```
genauigkeit = float(sys.argv[1])

for i in range(2,len(sys.argv)):

  z = float(sys.argv[i])

  w = wurzel( z, genauigkeit )

  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Das Programm heron.py

```
# heron.py
#-------------------------------------------------------------------------------------------------
import sys

def wurzel(n,r):
  langeSeite = n                                  # Die Seitenlängen am Anfang sind n und 1.
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:            # Solange der Unterschied der Seitenlängen > 2r ist,
    langeSeite = (langeSeite+kurzeSeite)/2        #   werden sie dichter zusammengebracht,
    kurzeSeite = n/langeSeite                     #   ohne die Fläche des Rechtecks zu ändern.
  return (langeSeite+kurzeSeite)/2                # Das Ergebnis wird zurückgegeben.
#-------------------------------------------------------------------------------------------------
# Das Hauptprogramm.
# Lies die Genauigkeit und weitere Werte von der Kommandozeile.
# Gib die Wurzeln der weiteren Werte mit der angegebenen Genauigkeit aus.

genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```

```
import sys

def wurzel(n,r):
    langeSeite = n
    kurzeSeite = 1
    while langeSeite - kurzeSeite > 2*r:
        langeSeite = (langeSeite+kurzeSeite)/2
        kurzeSeite = n/langeSeite
    return (langeSeite+kurzeSeite)/2


genauigkeit = float(sys.argv[1])

for i in range(2,len(sys.argv)):

  z = float(sys.argv[i])


  w = wurzel(z, genauigkeit)


  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

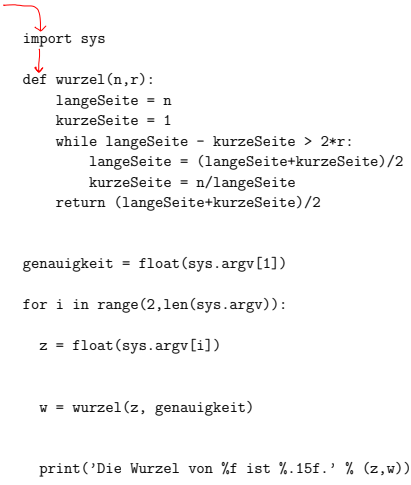
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```

```
import sys

def wurzel(n,r):
    langeSeite = n
    kurzeSeite = 1
    while langeSeite - kurzeSeite > 2*r:
        langeSeite = (langeSeite+kurzeSeite)/2
        kurzeSeite = n/langeSeite
    return (langeSeite+kurzeSeite)/2


genauigkeit = float(sys.argv[1])

for i in range(2,len(sys.argv)):

  z = float(sys.argv[i])


  w = wurzel(z, genauigkeit)


  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

Die sehr grobe Vorstellung vom Ablauf des Programms

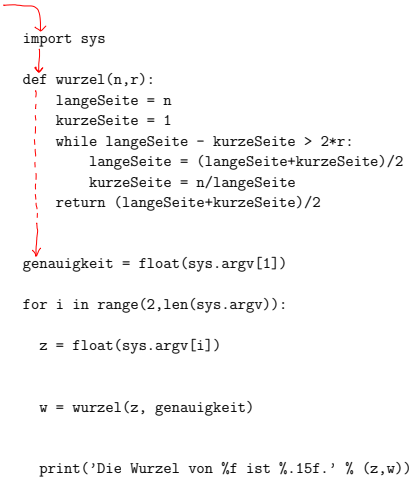
```
python3 heron.py 0.1 2 4
```



```
import sys

def wurzel(n,r):
    langeSeite = n
    kurzeSeite = 1
    while langeSeite - kurzeSeite > 2*r:
        langeSeite = (langeSeite+kurzeSeite)/2
        kurzeSeite = n/langeSeite
    return (langeSeite+kurzeSeite)/2


genauigkeit = float(sys.argv[1])

for i in range(2,len(sys.argv)):

  z = float(sys.argv[i])


  w = wurzel(z, genauigkeit)


  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

Die sehr grobe Vorstellung vom Ablauf des Programms

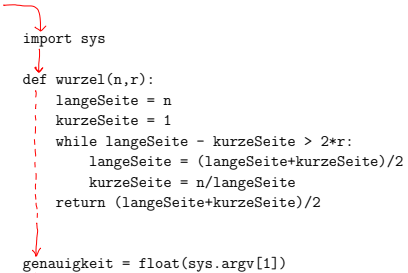
```
python3 heron.py 0.1 2 4
```



```
import sys

def wurzel(n,r):
    langeSeite = n
    kurzeSeite = 1
    while langeSeite - kurzeSeite > 2*r:
        langeSeite = (langeSeite+kurzeSeite)/2
        kurzeSeite = n/langeSeite
    return (langeSeite+kurzeSeite)/2


genauigkeit = float(sys.argv[1])

for i in range(2,len(sys.argv)):

  z = float(sys.argv[i])


  w = wurzel(z, genauigkeit)


  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

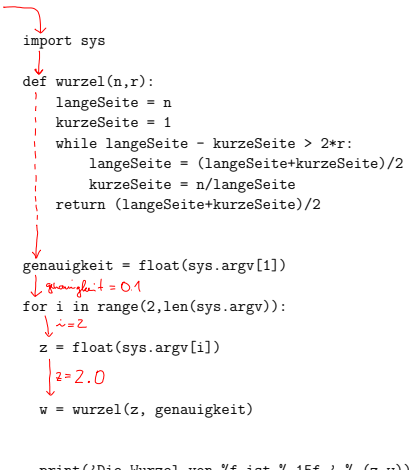
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```



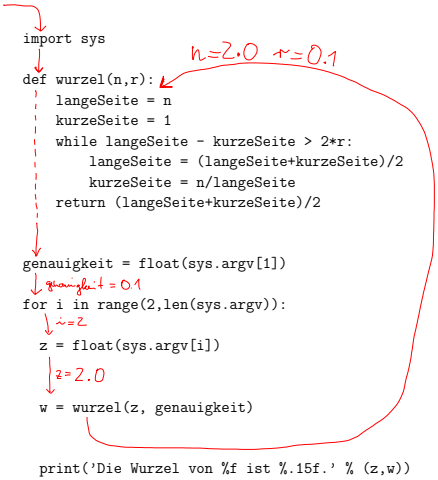
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```



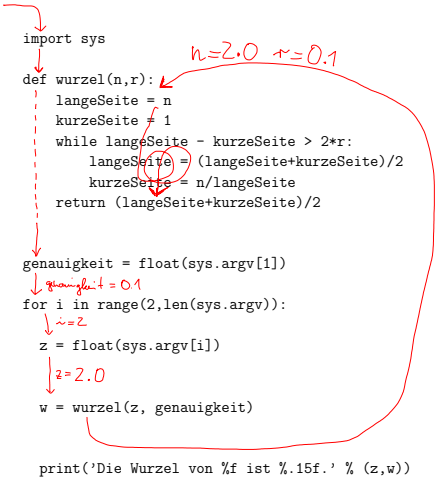
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```



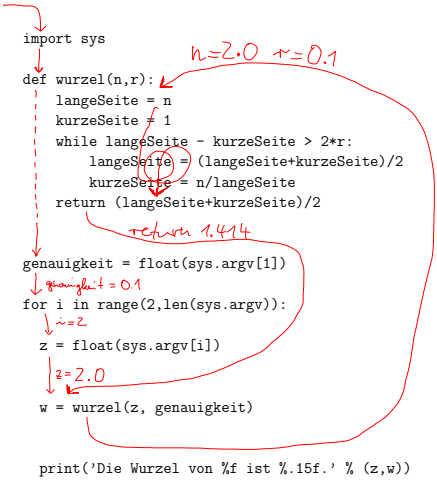
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```



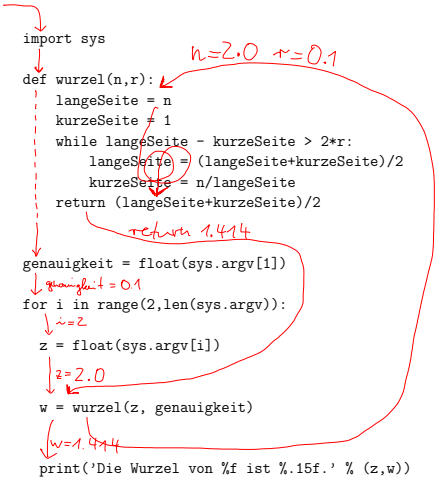
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```



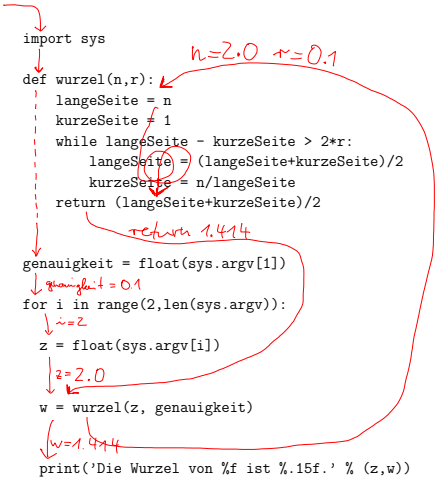
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
```



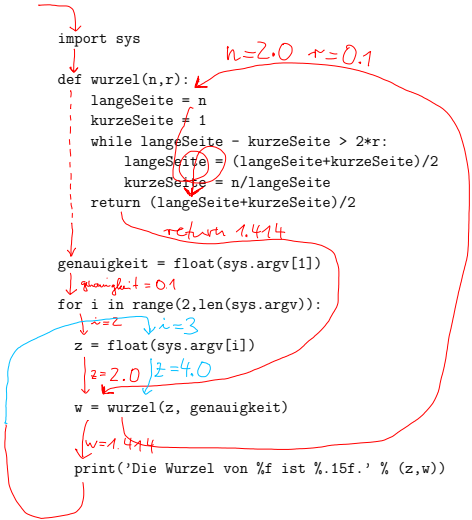
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
```



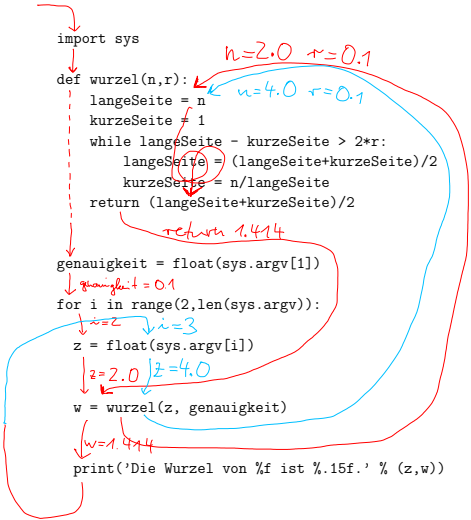
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
```



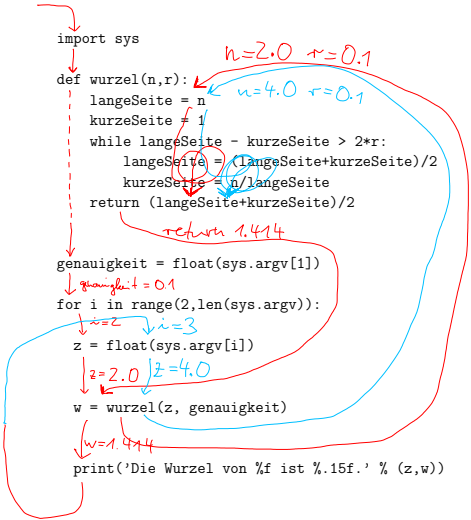
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
Die Wurzel von 4.0 ist 2.00000.
```



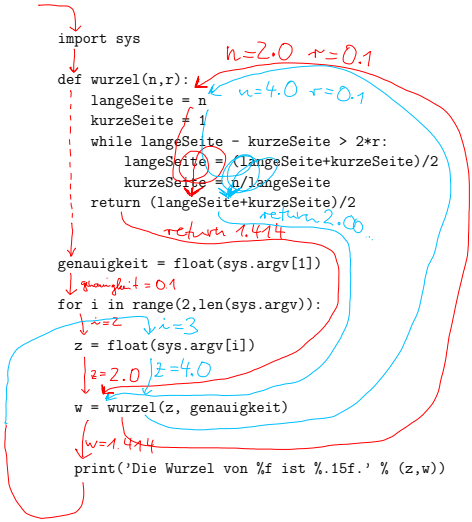
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
```



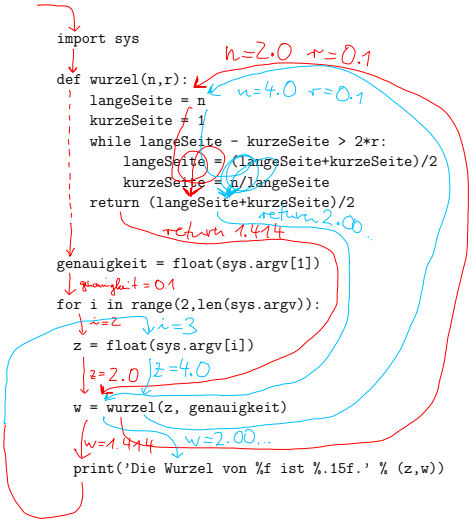
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
```



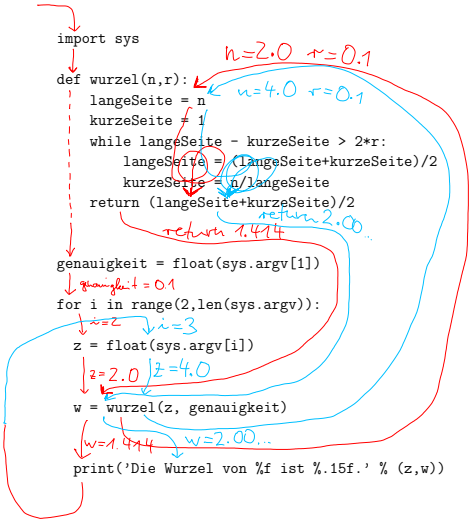
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
```



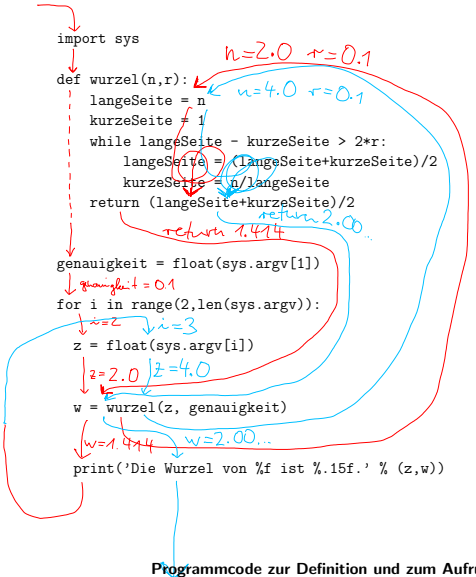
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
Die Wurzel von 4.0 ist 2.00000.
```



Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 heron.py 0.1 2 4
Die Wurzel von 2.0 ist 1.41421.
Die Wurzel von 4.0 ist 2.00000.
```



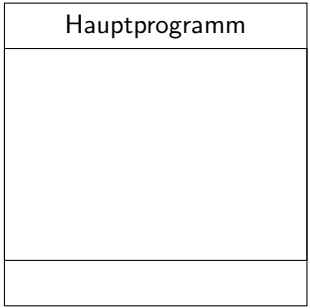
Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

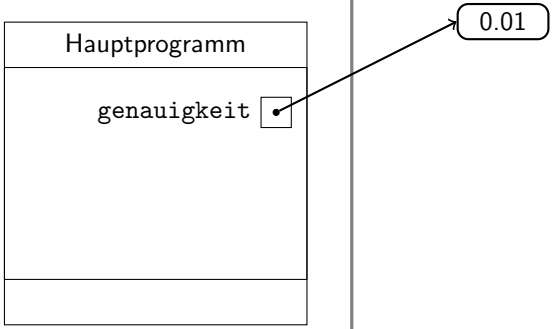
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

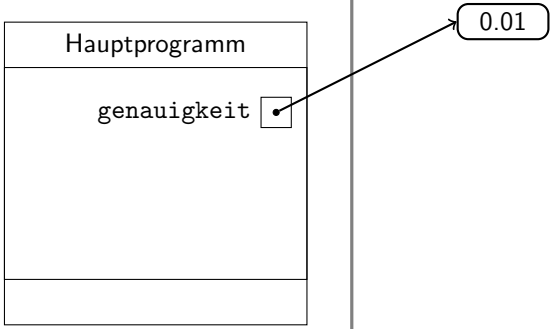
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

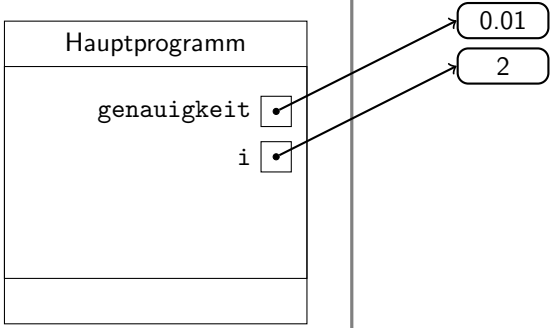
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

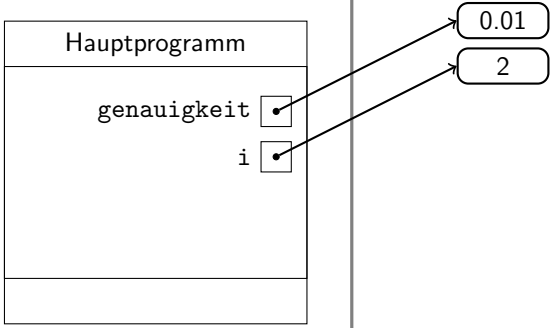
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

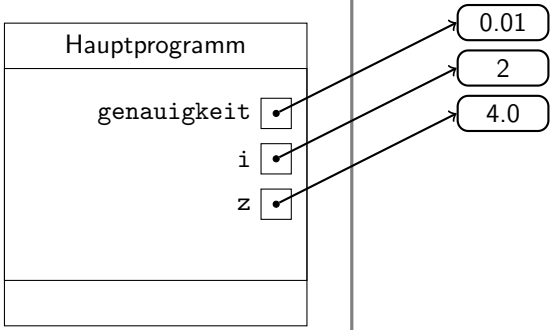
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

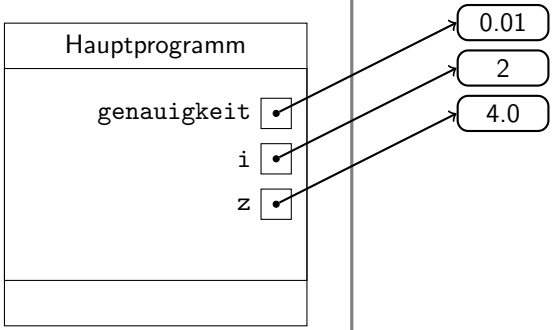
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

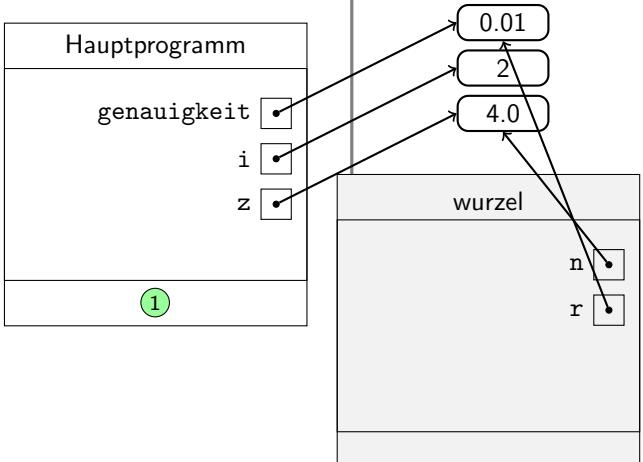
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

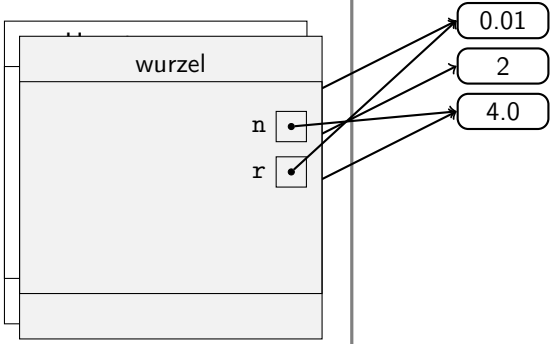
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

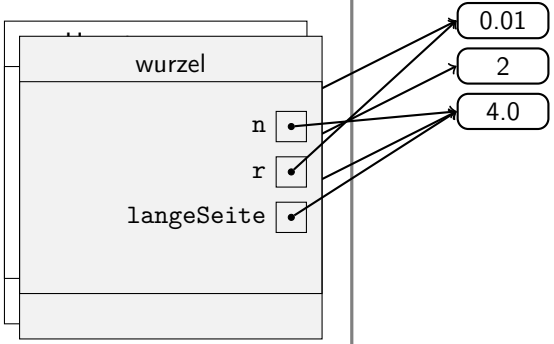
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

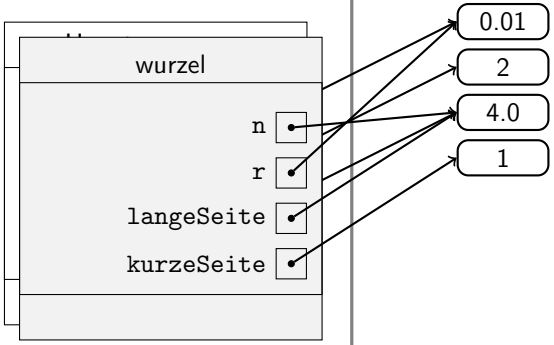
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

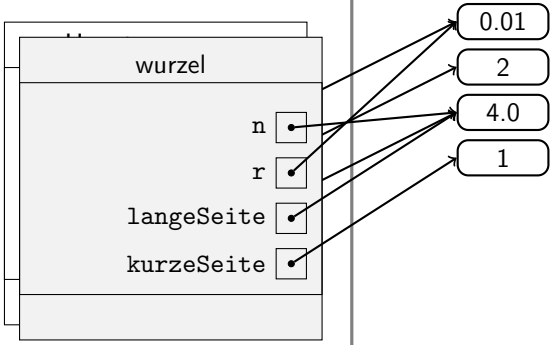
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

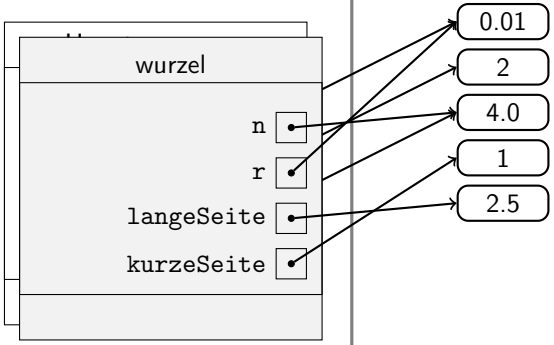
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

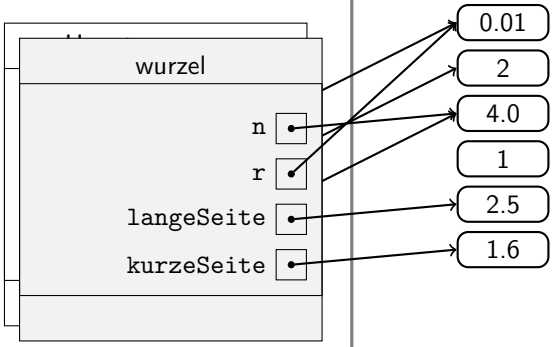
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

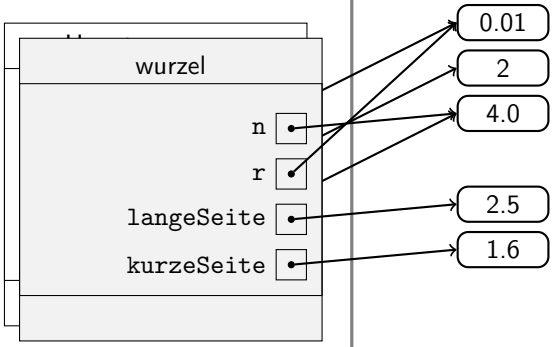
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

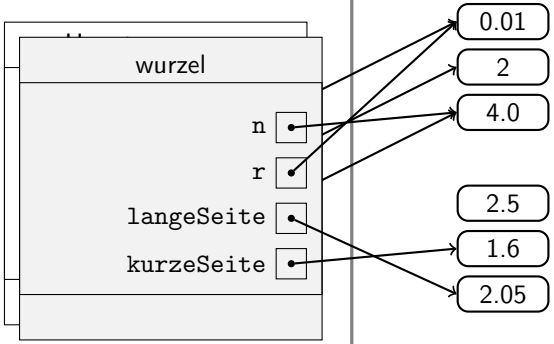
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

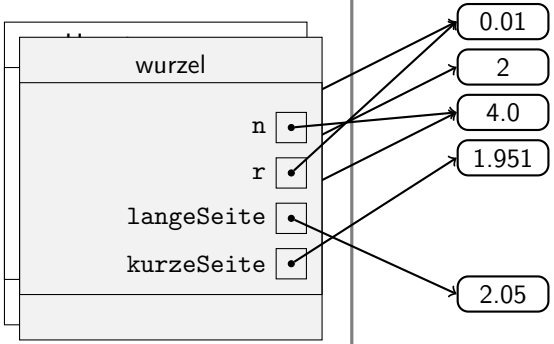
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

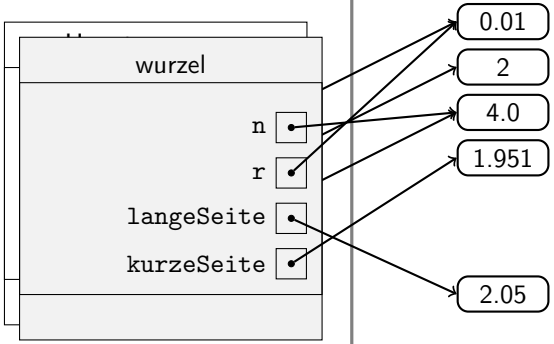
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

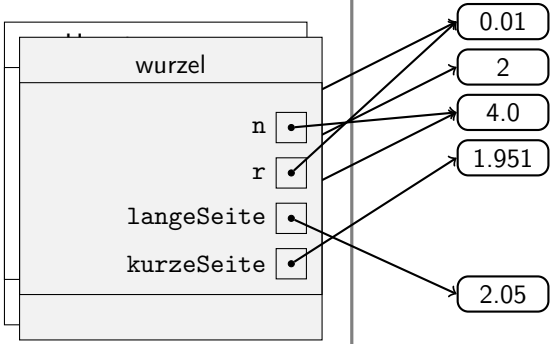
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

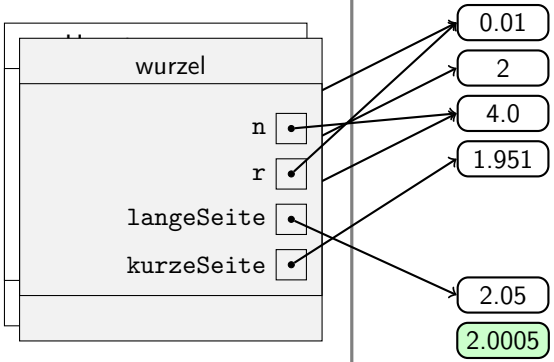
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

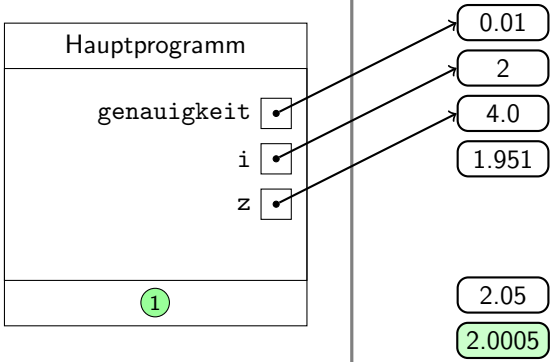
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

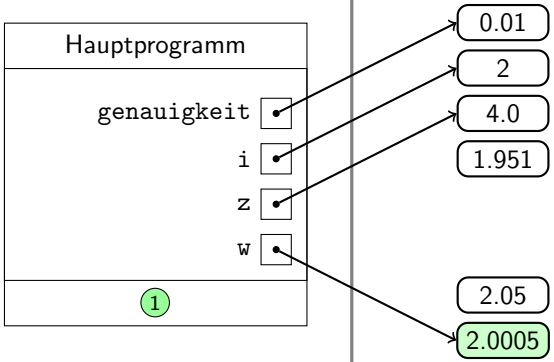
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

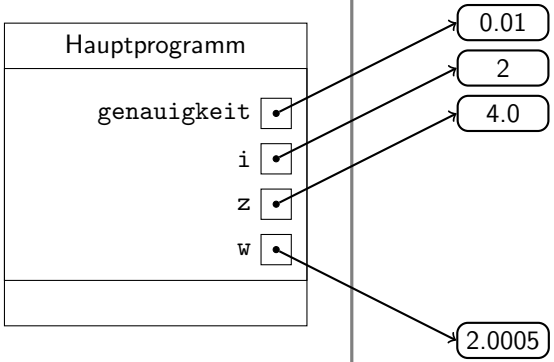
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
```

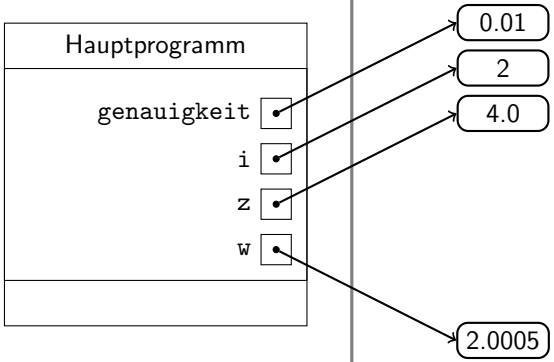
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

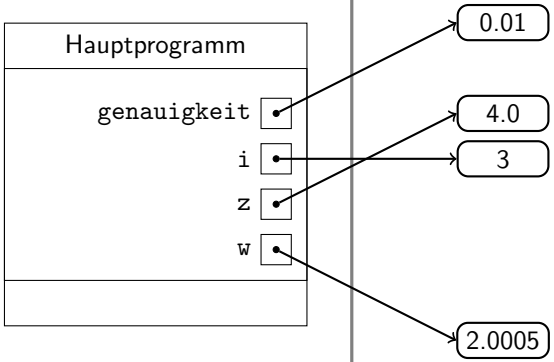
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

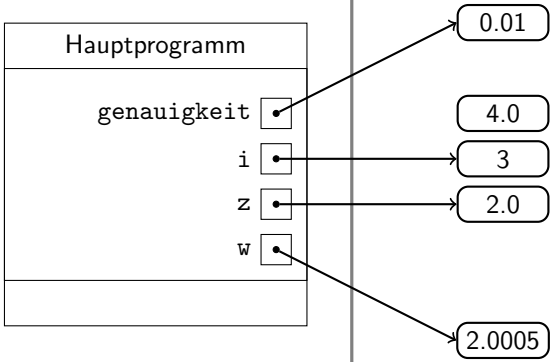
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

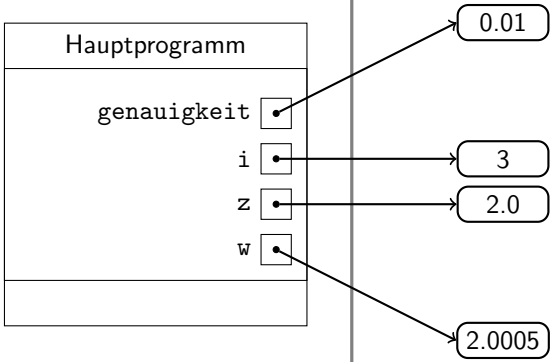
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

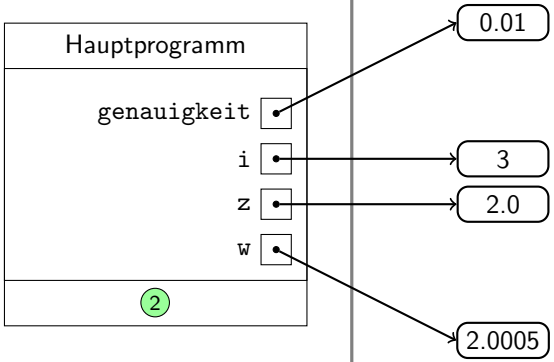
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

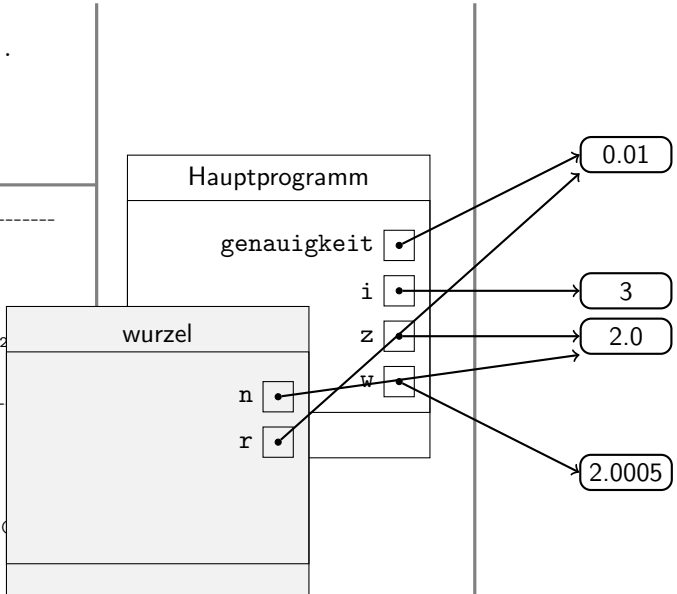
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

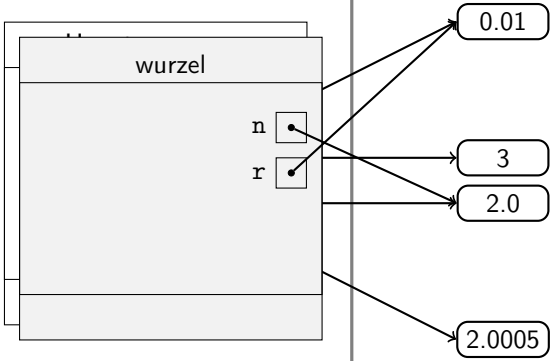
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

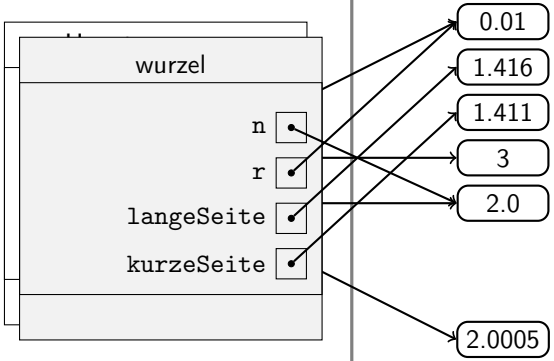
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

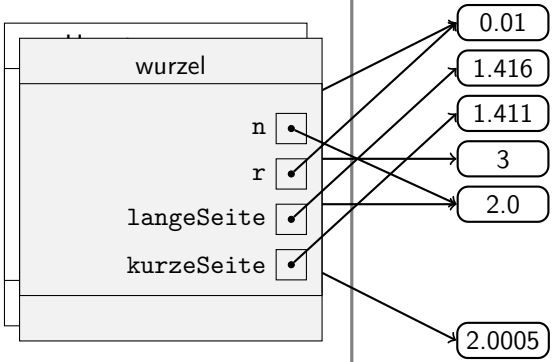
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

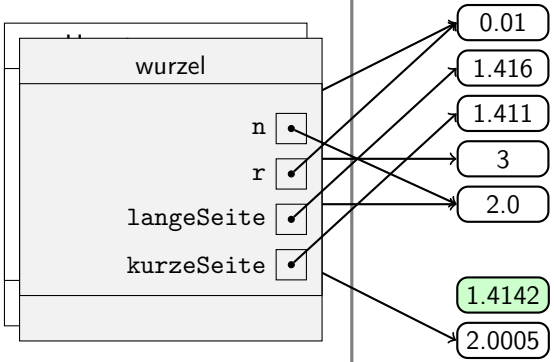
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

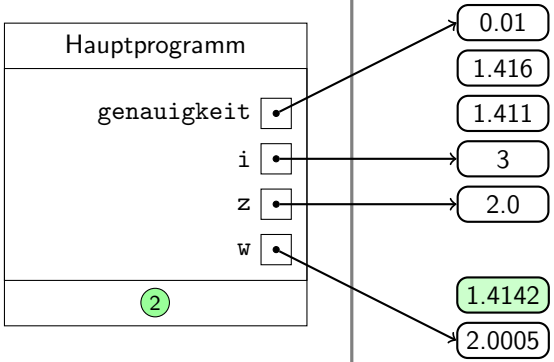
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

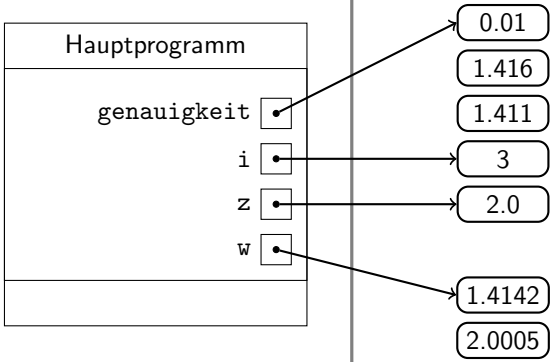
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
```

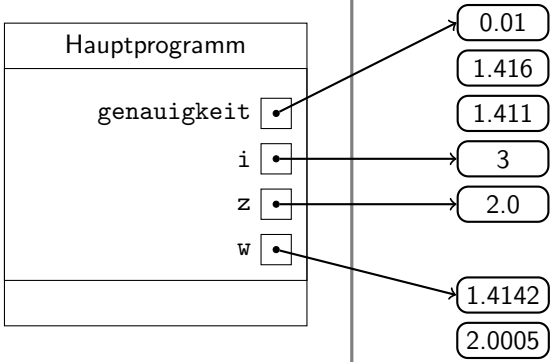
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
Die Wurzel von 2.0 ist 1.4142...
```

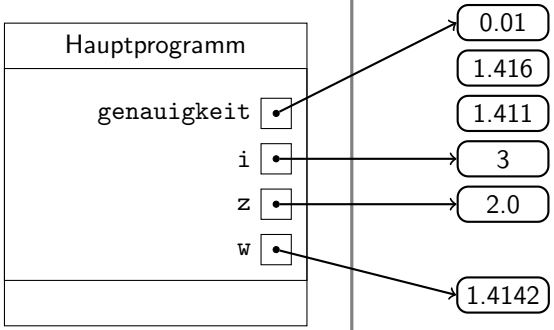
```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Was im Speicher passiert . . .

```
python3 heron.py 0.01 4 2
Die Wurzel von 4.0 ist 2.0005...
Die Wurzel von 2.0 ist 1.4142...
```

```
#-----------------------------------------------
# Die Funktion wurzel().
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
#-----------------------------------------------
# Das Hauptprogramm.
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```



Gültigkeitsbereich (scope) von Variablen (1)

i, z und genauigkeit können hier nicht verändert werden, w ist unbekannt

```
def wurzel(n,r):
  langeSeite = n
  kurzeSeite = 1
  while langeSeite - kurzeSeite > 2*r:
    langeSeite = (langeSeite+kurzeSeite)/2
    kurzeSeite = n/langeSeite
  return (langeSeite+kurzeSeite)/2
```

Gültigkeitsbereich der lokalen Variablen n, r, langeSeite und kurzeSeite

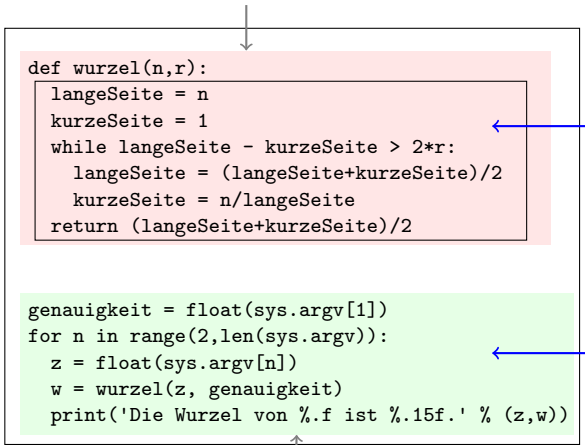
```
genauigkeit = float(sys.argv[1])
for i in range(2,len(sys.argv)):
  z = float(sys.argv[i])
  w = wurzel(z, genauigkeit)
  print('Die Wurzel von %f ist %.15f.' % (z,w))
```

Gültigkeitsbereich der Variablen genauigkeit, i, z und w

n, r, langeSeite und kurzeSeite sind hier unbekannt

Gültigkeitsbereich (scope) von Variablen (2)

z und genauigkeit können hier nicht verändert werden, w ist unbekannt



Gültigkeitsbereich der lokalen Variablen n, r, langeSeite und kurzeSeite

zwei verschiedene Variablen!

Gültigkeitsbereich der Variablen genauigkeit, n, z und w

r, langeSeite und kurzeSeite sind hier unbekannt

Zwischenbilanz

- Aufruf einer Funktion
- Definition einer Funktion mit `def` und `return`
- Programmablauf beim Aufruf einer Funktion
- Gültigkeitsbereiche von Variablen

4 Programmier-Auftrag:
Drehe die Reihenfolge der Elemente eines Arrays um

Eine Funktion mit einem Argument eines veränderbaren Datentyps

Zum Beispiel soll aus dem Array `[1, 2, 3, 4, 5, 6]`
das Array `[6, 5, 4, 3, 2, 1]` werden.

Die Funktion:

- Parameter: Array *a*
- Rückgabewert: keiner
- Seiteneffekt: das Array, mit dem die Funktion aufgerufen wird, wird verändert

Grobe Programm-Struktur der Funktion:

1. Für jeden Index `i` bis zur Mitte des Arrays `a`:
   - 1.1 Tausche das `i`-te Element von links mit dem `i`-ten Element von rechts.

Das Programm arraydrehen.py

```
# arraydrehen.py
#-------------------------------------------------------------------------
# Dieses Programm enthält die Funktion umdrehen(a), die das Array a umdreht.
#-------------------------------------------------------------------------

def umdrehen(a):
  # Für alle Indizes i in der linken Hälfte des Arrays
  # wird das i-te Element in a von links  (das ist a[i])
  # mit dem i-ten Element in a von rechts (das ist a[-1-i]) vertauscht.
  for i in range(len(a)//2):
    a[i], a[-1-i] = a[-1-i], a[i]

#---- Hauptprogramm ----------------------------------------------------------
# Wir testen die Funktion umdrehen() an einem Beispiel.

b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)

#-------------------------------------------------------------------------
# python3 arraydrehen.py
# [1, 2, 3, 4, 5, 6]
# [6, 5, 4, 3, 2, 1]
```

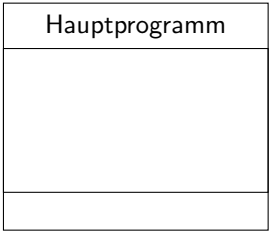
Was im Speicher passiert . . .

```
python3 arraydrehen.py
```

Was im Speicher passiert . . .

```
python3 arraydrehen.py
```

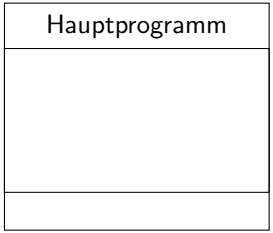
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```

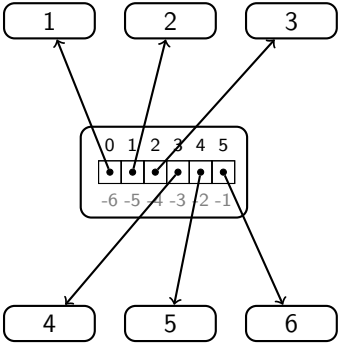


Was im Speicher passiert . . .

```
python3 arraydrehen.py
```

```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```

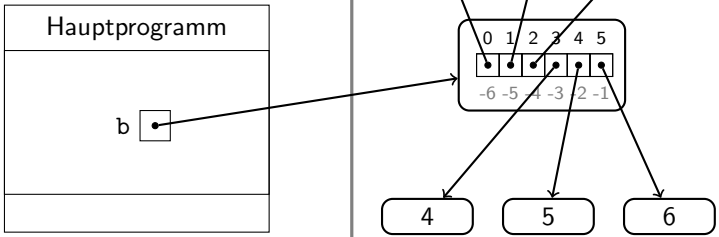




Was im Speicher passiert . . .

```
python3 arraydrehen.py
```

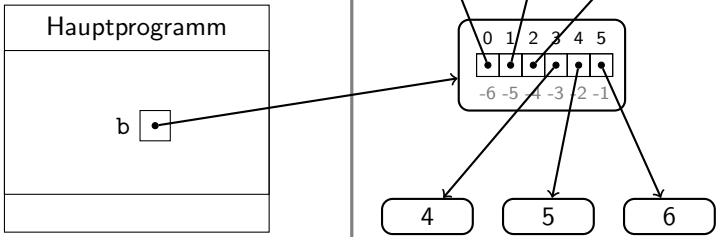
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
```

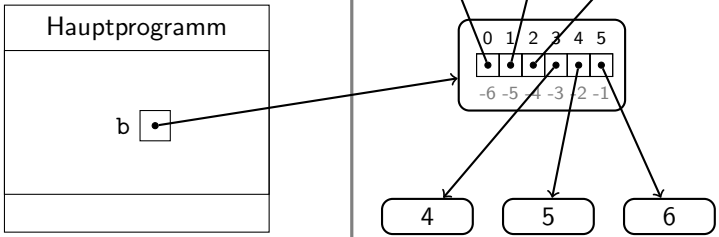
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

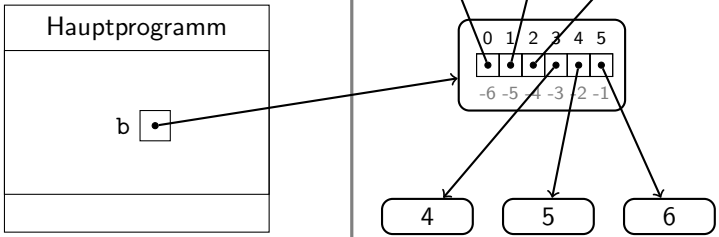
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

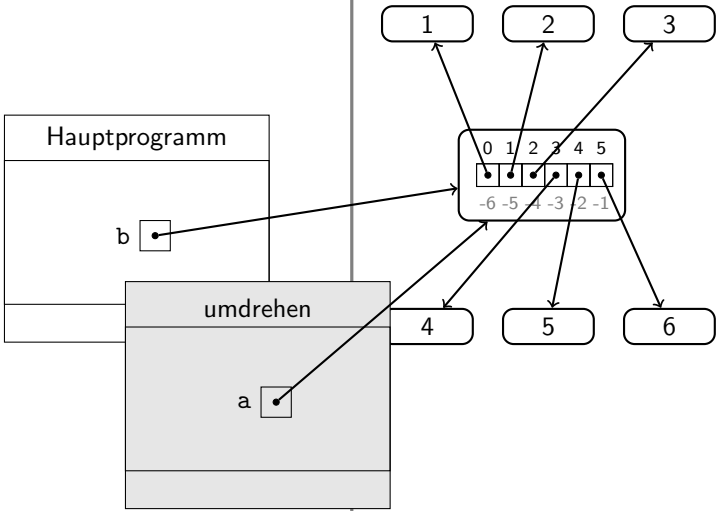
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

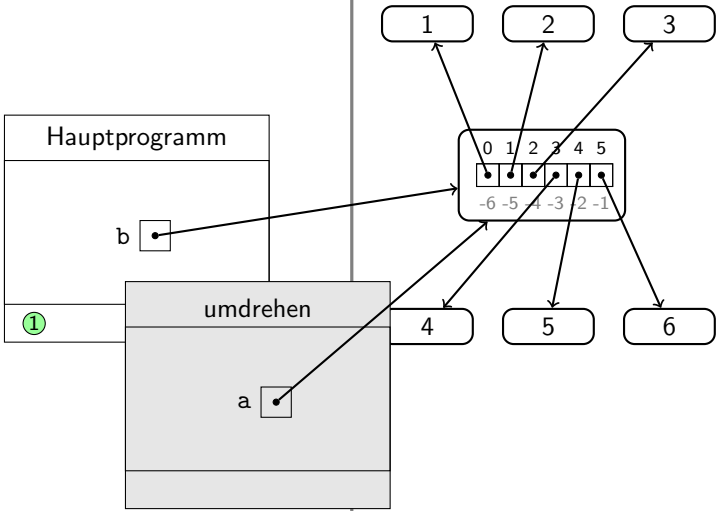
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

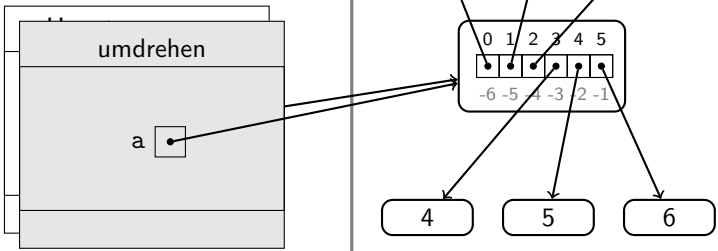
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
u 1 ehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

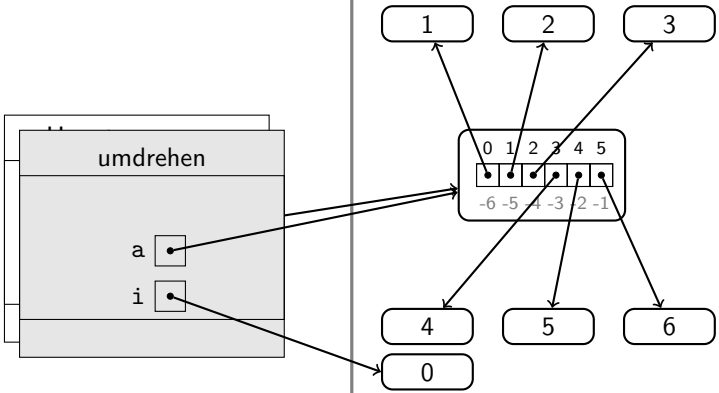
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

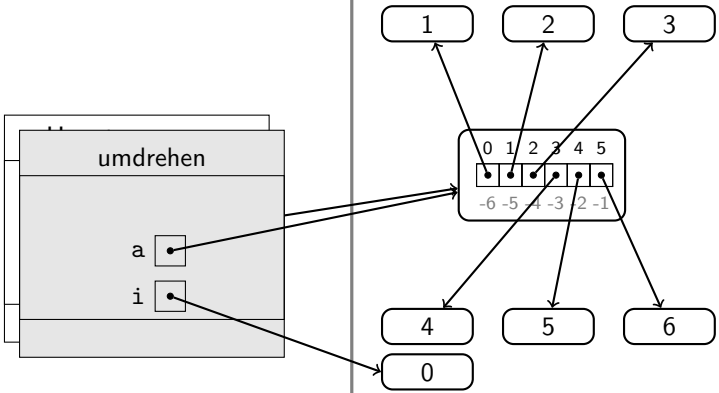
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
u1ehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

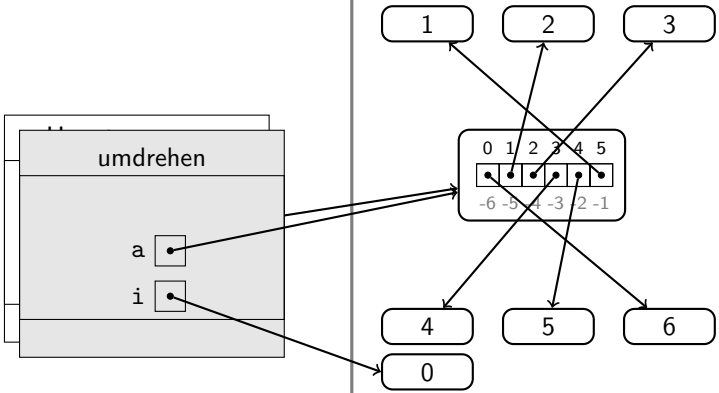
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
u1ehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

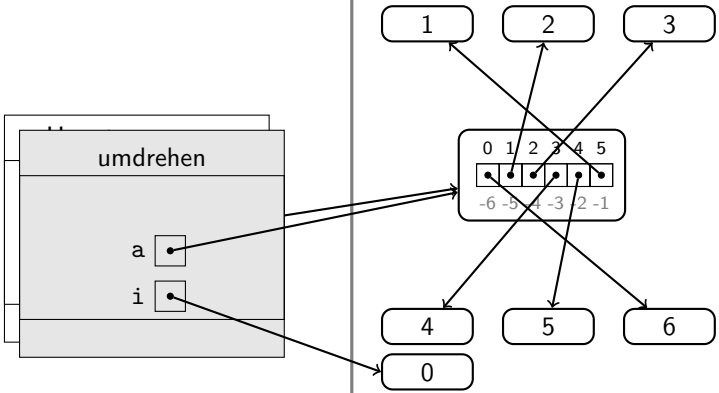
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
u 1 ehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

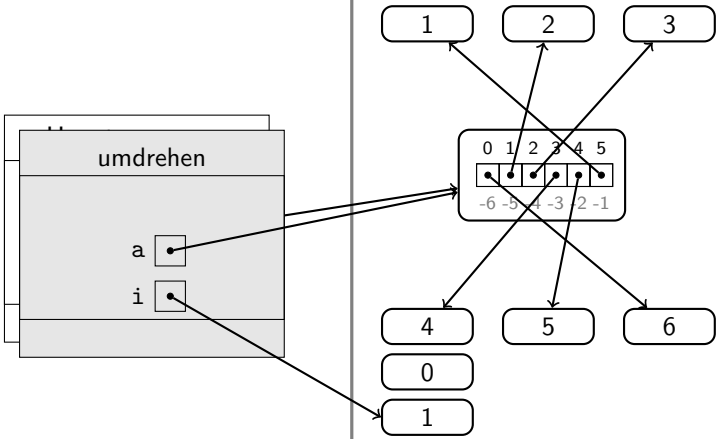
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

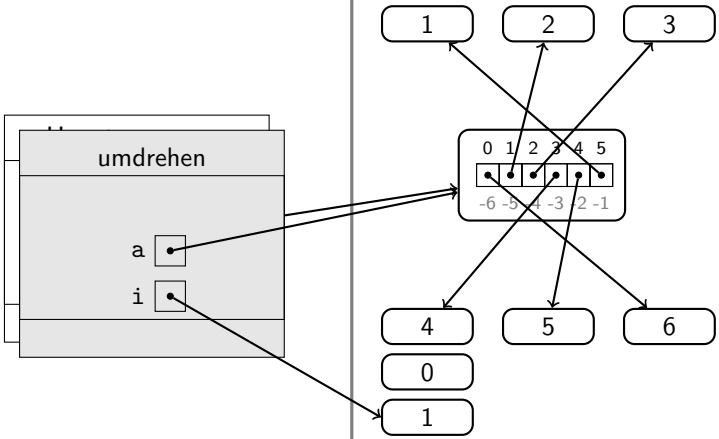
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

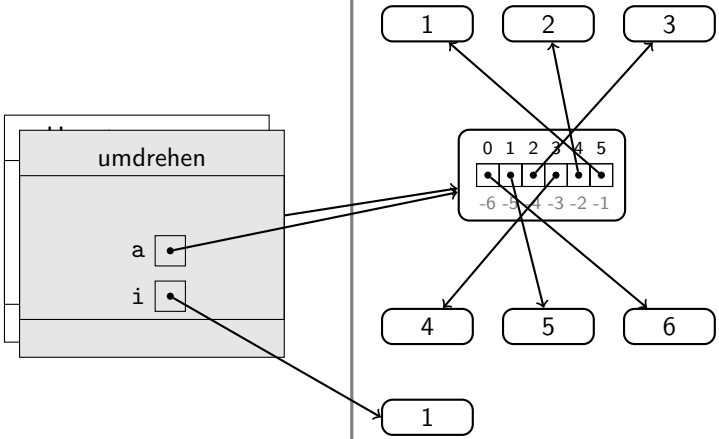
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

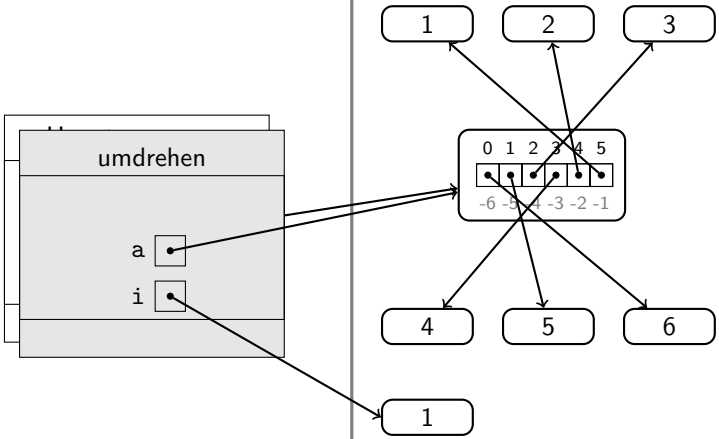
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

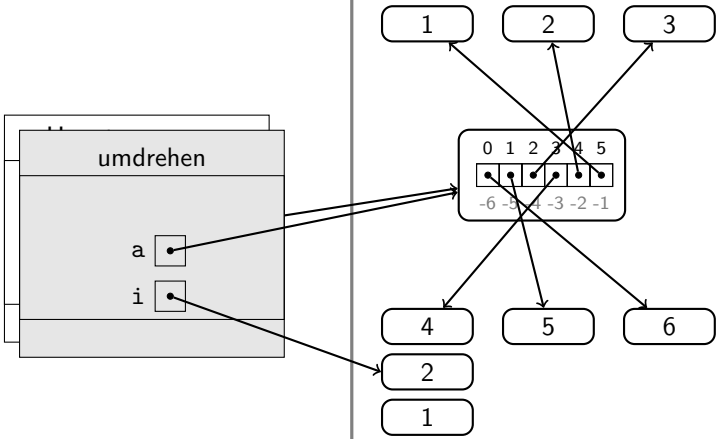
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

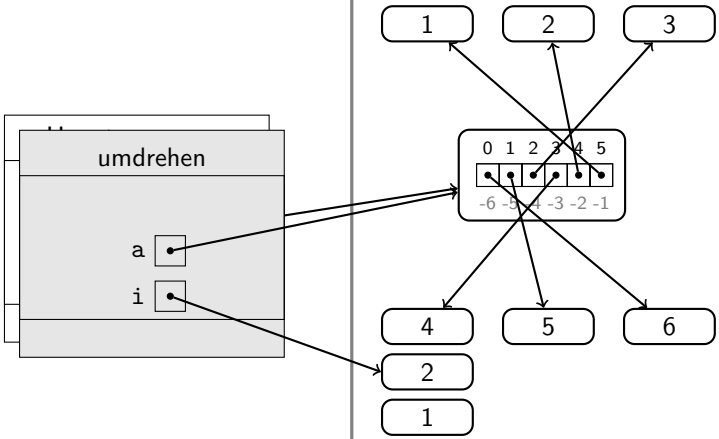
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

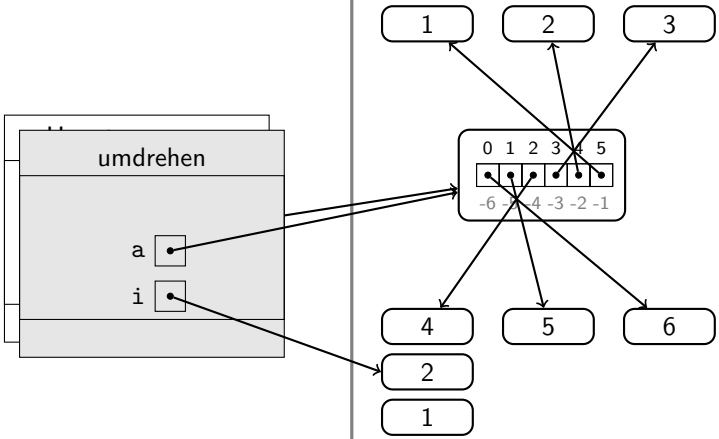
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
u1ehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

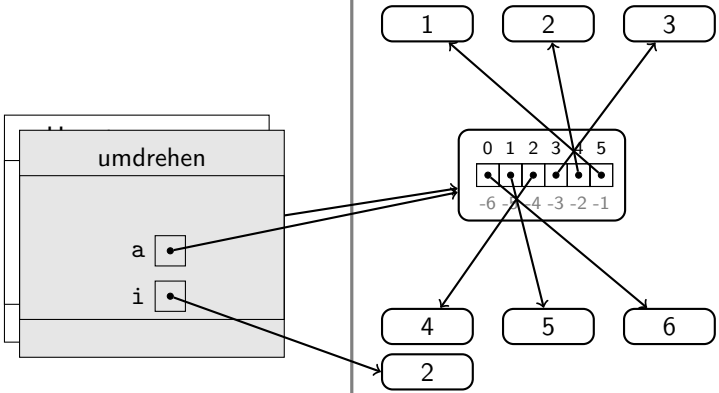
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
u1ehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

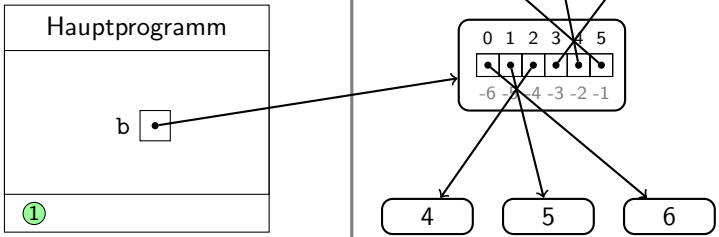
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

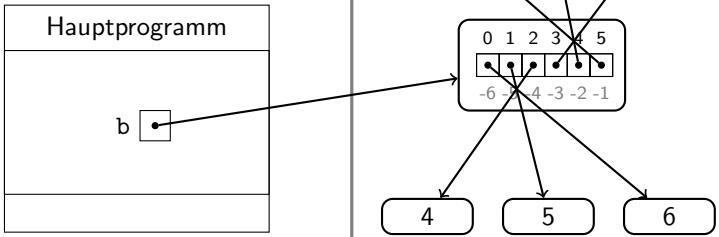
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
```

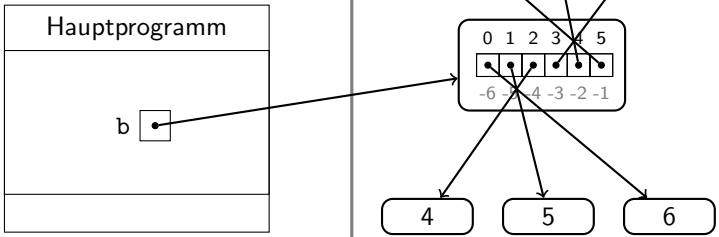
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1]
```

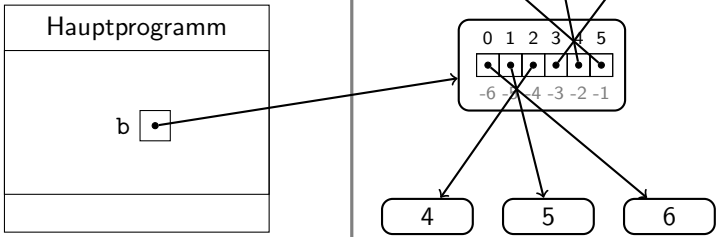
```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



Was im Speicher passiert . . .

```
python3 arraydrehen.py
[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1]
```

```
#-----------------------------------------------
# Die Funktion umdrehen(a) dreht Array a um.
def umdrehen(a):
   for i in range(len(a)//2):
     a[i], a[-1-i] = a[-1-i], a[i]
   return

#-----------------------------------------------
# Das Hauptprogramm.
b = [ 1,2,3,4,5,6 ]
print(b)
umdrehen(b)
print(b)
```



5 Programmier-Auftrag: Funktion zur Berechnung der Primzahleigenschaft

Eine Funktion mit mehreren `return`

Stelle fest, welche der eingegebenen Zahlen Primzahlen sind.

Beispiel: 2 3 5 7 11 13 16 sind Primzahlen, 1 4 6 8 9 10 12 14 15 sind keine Primzahlen

---

Grobe Programm-Struktur:

1. Für jede eingegebene Zahl:
   - 1.1 gib aus, ob sie eine Primzahl ist

Die Idee und die Strukturen der Funktion

Idee: n ist keine Primzahl, falls

- $n < 2$ oder
- n besitzt einen Teiler t mit $t^2 \leqslant n$, der weder 1 noch n ist.

Die Funktion:

- Parameter: n
- Rückgabewert:
  `True`, falls n eine Primzahl ist
  `False`, falls n keine Primzahl ist

Grobe Programm-Struktur der Funktion:

1. Falls $n < 2$: der Rückgabewert ist `False`
2. Teste für alle t mit $t \geqslant 2$ und $t^2 \leqslant n$, ob t ein Teiler von n ist.
   Falls dabei ein Teiler gefunden wird, ist der Rückgabewert `False`.
3. Wenn kein Teiler gefunden wurde, ist der Rückgabewert `True`.

Das Programm primzahlen.py

```
# primzahlen.py
#-------------------------------------------------------------------------------------------
import sys

# Funktion istPrimzahl(n) erhält als Argument int n
# und hat Rückgabewert True, falls n eine Primzahl ist,
# sowie Rückgabewert False, falls n keine Primzahl ist.
def istPrimzahl( n ):
   if n<2: return False    # Falls n<2, dann ist n keine Primzahl.
   t = 2                   # Versuche, bei allen Zahlen t mit 2<=t
   while t*t<=n:           #        und t*t<=n einen Teiler von n zu finden.
     if n%t == 0: return False  # Falls ein Teiler gefunden wurde, ist n keine Primzahl.
     t += 1
   return True             # Falls kein Teiler gefunden wurde, ist n eine Primzahl.

# --------  Hauptprogramm  ------------------------------------------------------------
# Lies die Zahlen von der Kommandozeile und gib aus, ob sie Primzahlen sind.

for i in range (1,len(sys.argv)):
   argument = int(sys.argv[i])
   if istPrimzahl(argument):  print( '%d ist Primzahl.' % argument )
   else:                      print( '%d ist keine Primzahl.' % argument )
```

Das Programm primzahlen.py

```
# primzahlen.py
#-------------------------------------------------------------------------------------------
import sys

# Funktion istPrimzahl(n) erhält als Argument int n
# und hat Rückgabewert True, falls n eine Primzahl ist,
# sowie Rückgabewert False, falls n keine Primzahl ist
def istPrimzahl( n ):
   if n<2: return False    # Falls n<2, dann ist n kein
   t = 2                   # Versuche, bei allen Zahler
   while t*t<=n:           #        und t*t<=n einen Te
     if n%t == 0: return False  # Falls ein Teiler gej
     t += 1
   return True             # Falls kein Teiler gefunder

# --------  Hauptprogramm  ----------------------------
# Lies die Zahlen von der Kommandozeile und gib aus, ob sie Primzahlen sind.

for i in range (1,len(sys.argv)):
   argument = int(sys.argv[i])
   if istPrimzahl(argument):  print( '%d ist Primzahl.' % argument )
   else:                      print( '%d ist keine Primzahl.' % argument )
```

```
mundhenk@mm: ~/Python/VL08-Funktionen
File  Edit  View  Search  Terminal  Help
python@home: python3 primzahlen.py 23 45 1021 13131313
23 ist Primzahl.
45 ist keine Primzahl.
1021 ist Primzahl.
13131313 ist keine Primzahl.
python@home:
```

Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 2
```

```
import sys

def istPrimzahl( n ):
   if n<2: return False
   t = 2
   while t*t<=n:
      if n%t == 0: return False
      t += 1
   return True

# --------  Hauptprogramm  -----------------------

argument = int(sys.argv[1])
if istPrimzahl(argument):
   print( '%d ist Primzahl.' % argument )
else:
   print( '%d ist keine Primzahl.' % argument )
```

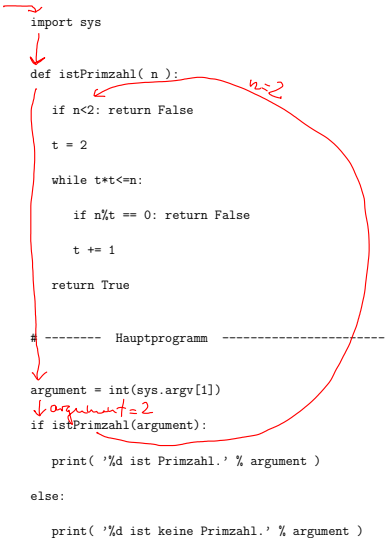
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 2
```

```
import sys

def istPrimzahl( n ):

   if n<2: return False

   t = 2

   while t*t<=n:

      if n%t == 0: return False

      t += 1

   return True


# --------  Hauptprogramm  -----------------------

argument = int(sys.argv[1])
if istPrimzahl(argument):

   print( '%d ist Primzahl.' % argument )

else:

   print( '%d ist keine Primzahl.' % argument )
```

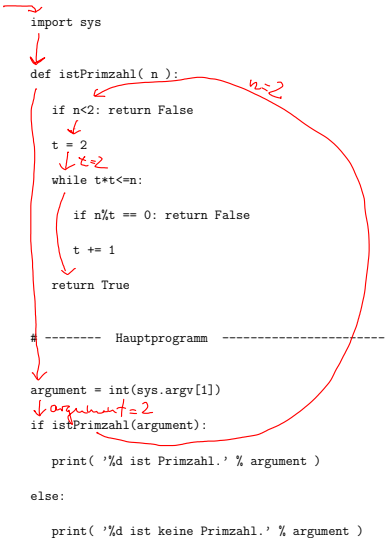
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 2
```



Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 2
```



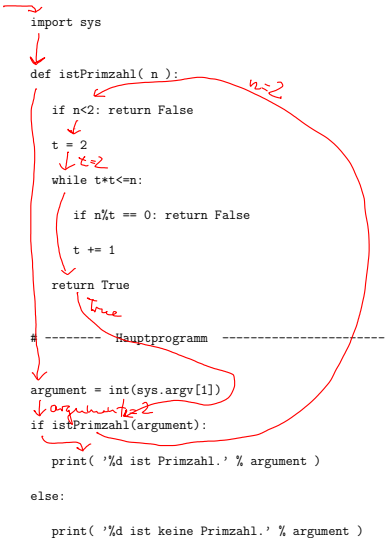
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 2
```

```
import sys

def istPrimzahl( n ):

    if n<2: return False

    t = 2

    while t*t<=n:

        if n%t == 0: return False

        t += 1

    return True

# --------  Hauptprogramm  ----------------------

argument = int(sys.argv[1])

if istPrimzahl(argument):

    print( '%d ist Primzahl.' % argument )

else:

    print( '%d ist keine Primzahl.' % argument )
```

Die sehr grobe Vorstellung vom Ablauf des Programms

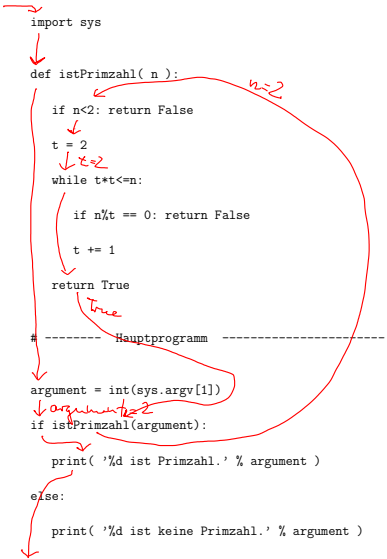
```
python3 primzahlen.py 2
2 ist Primzahl.
```



```
import sys

def istPrimzahl( n ):
    if n<2: return False
    t = 2
    while t*t<=n:
        if n%t == 0: return False
        t += 1
    return True

# --------  Hauptprogramm  ----------------------

argument = int(sys.argv[1])
if istPrimzahl(argument):
    print( '%d ist Primzahl.' % argument )
else:
    print( '%d ist keine Primzahl.' % argument )
```

Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 2
2 ist Primzahl.
```



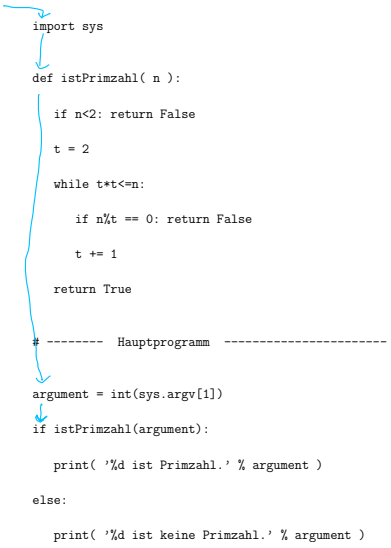
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 1
```

```
import sys

def istPrimzahl( n ):
   if n<2: return False
   t = 2
   while t*t<=n:
      if n%t == 0: return False
      t += 1
   return True

# --------  Hauptprogramm  -----------------------

argument = int(sys.argv[1])
if istPrimzahl(argument):
   print( '%d ist Primzahl.' % argument )
else:
   print( '%d ist keine Primzahl.' % argument )
```

Die sehr grobe Vorstellung vom Ablauf des Programms

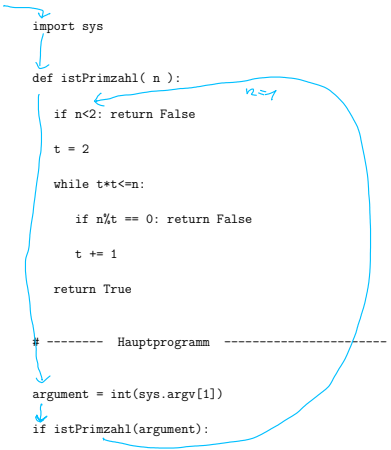
```
python3 primzahlen.py 1
```



```
import sys

def istPrimzahl( n ):
    if n<2: return False
    t = 2
    while t*t<=n:
        if n%t == 0: return False
        t += 1
    return True

# --------  Hauptprogramm  -----------------------

argument = int(sys.argv[1])
if istPrimzahl(argument):
    print( '%d ist Primzahl.' % argument )
else:
    print( '%d ist keine Primzahl.' % argument )
```

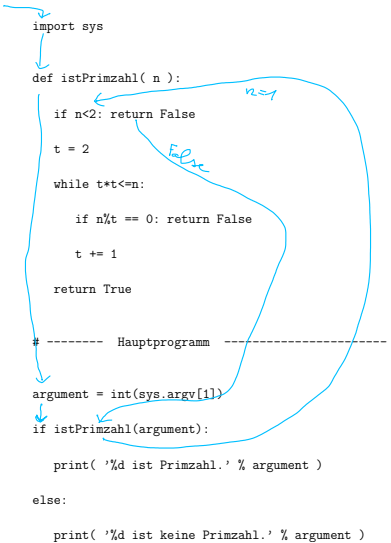
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 1
```



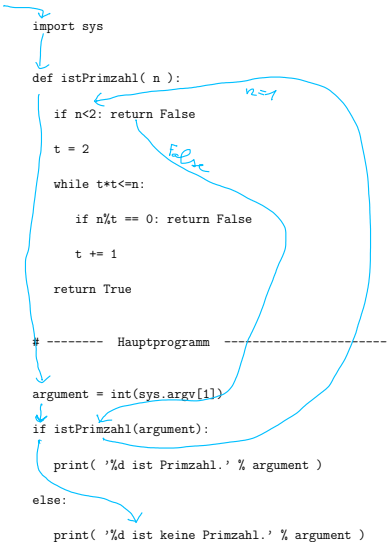
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 1
```



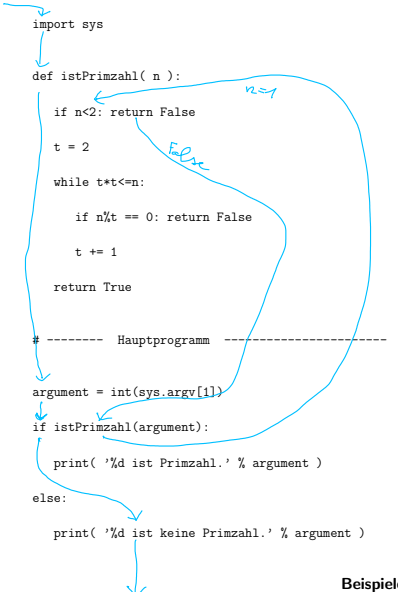
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 1
1 ist keine Primzahl.
```



Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 1
1 ist keine Primzahl.
```



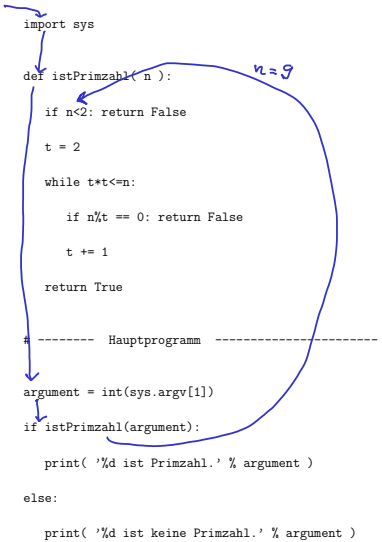
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
```

```
import sys

def istPrimzahl( n ):
   if n<2: return False
   t = 2
   while t*t<=n:
      if n%t == 0: return False
      t += 1
   return True

# --------  Hauptprogramm  -----------------------

argument = int(sys.argv[1])
if istPrimzahl(argument):
   print( '%d ist Primzahl.' % argument )
else:
   print( '%d ist keine Primzahl.' % argument )
```

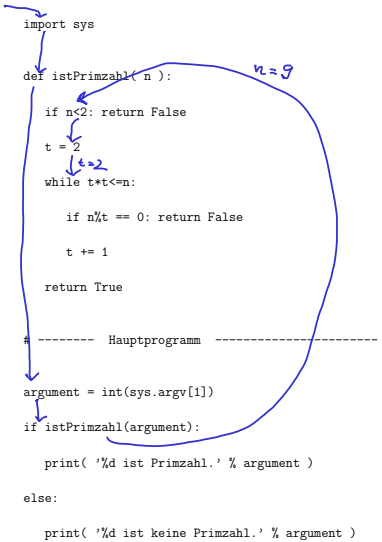
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
```



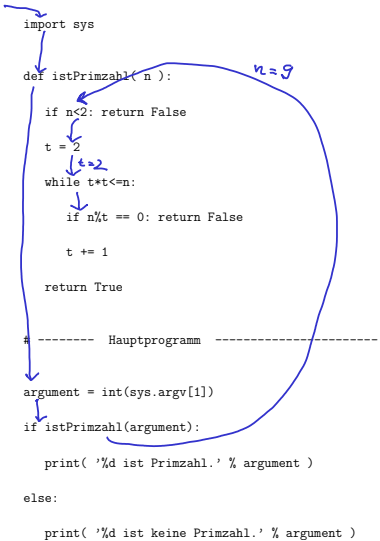
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
```



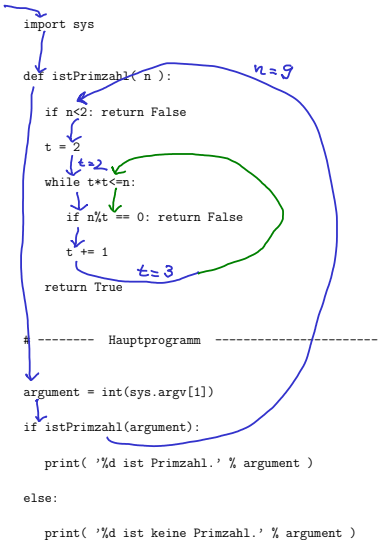
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
```



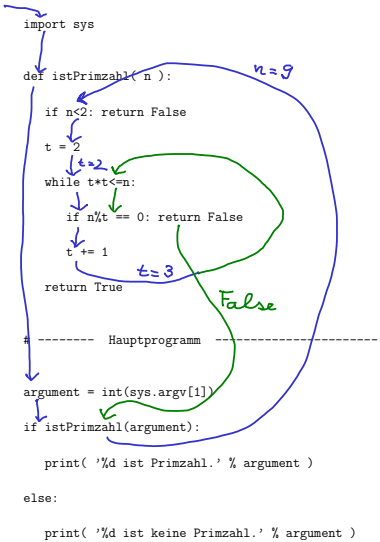
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
```



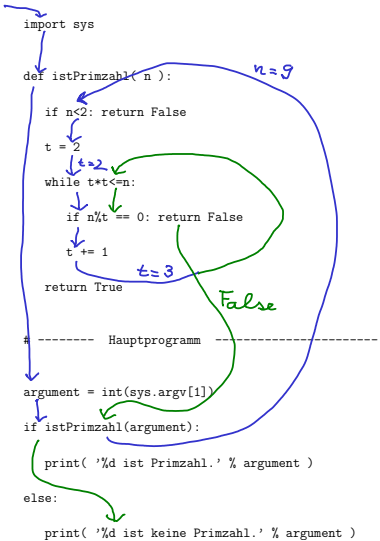
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
```



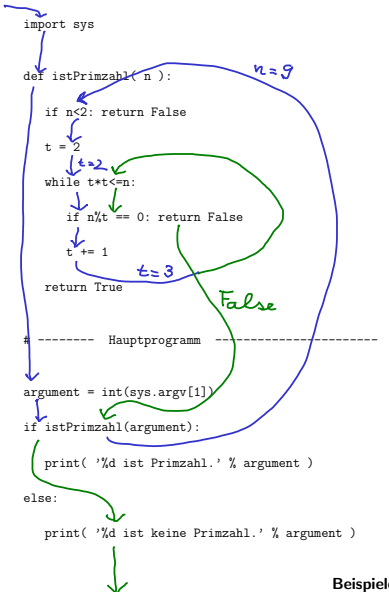
Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
9 ist keine Primzahl.
```



Die sehr grobe Vorstellung vom Ablauf des Programms

```
python3 primzahlen.py 9
9 ist keine Primzahl.
```



Zusammenfassung

- Durch Definition und Benutzung von Funktionen hat man beim Programmieren neue Möglichkeiten zum Strukturieren von Programmen.
- Man muss sich den Ablauf eines Programms nicht mehr Schritt für Schritt in elementaren Anweisungen vorstellen, sondern kann abstrakter über Funktionsaufrufe und Rückgabewerte argumentieren.
- Der Programmcode wird dadurch einfacher
  zu entwickeln, zu verstehen, zu verbessern und zu warten.
- Dadurch lassen sich auch „kompliziertere“ Anwendungen übersichtlicher programmieren.