Vorlesung 03

1.4 Arrays – der Datentyp `list`

Bisher haben wir Variablen benutzt,
um ein einfaches „Datum“ zu speichern und darauf zuzugreifen –
eine Zahl, einen Text oder einen Wahrheitswert.

Was macht man, wenn man „sehr viele“ Daten speichern muss,
um daraus ein Ergebnis zu berechnen?

Das geht mit einer entsprechenden *Datenstruktur*.

Sie dient der Organisation von Daten zur Verarbeitung mit einem Computer-Programm.

Eine einfache Datenstruktur ist das *Array* (in Python als Datentyp `list` implementiert),
mit dem man z.B. beliebig lange Folgen von Zahlen/Strings/Arrays etc. verarbeiten kann.

Programmier-Auftrag: berechne die mittlere absolute Abweichung einer Zahlenfolge von ihrem Mittelwert

Beispiel:

die Zahlen 17, 23, 16, 22, 7 haben den *Mittelwert*

$$\frac{17+23+16+22+7}{5} = 17.$$

Die *absoluten Abweichungen vom Mittelwert* der Zahlen sind

Zahl:	17	23	16	22	7
absolute Abweichung von 17:	0	6	1	5	10

Also ist

$$\frac{0+6+1+5+10}{5} = 4\frac{2}{5}$$

die mittlere absolute Abweichung der Zahlen von ihrem Mittelwert.

Idee

Idee für den Algorithmus:

- lies die Zahlen ein
- berechne den Mittelwert der Zahlen
- berechne die absoluten Abweichungen vom Mittelwert der Zahlen und summiere sie auf
- das Ergebnis ist der Quotient aus dieser Summe und der Anzahl der Zahlen

Problem:

Es kann eine beliebige Anzahl von Zahlen eingegeben werden.

Man kann die absoluten Abweichungen erst berechnen, wenn man den Mittelwert kennt.

Also muss man alle Zahlen speichern.

Idee

Idee für den Algorithmus:

- lies die Zahlen ein
- berechne den Mittelwert der Zahlen
- berechne die absoluten Abweichungen vom Mittelwert der Zahlen und summiere sie auf
- das Ergebnis ist der Quotient aus dieser Summe und der Anzahl der Zahlen

Problem:

Es kann eine beliebige Anzahl von Zahlen eingegeben werden.

Man kann die absoluten Abweichungen erst berechnen, wenn man den Mittelwert kennt.

Also muss man alle Zahlen speichern.

Datentyp `list` (Array) – Folge von Objekten

Die einfachste Vorstellung von einem Array ist eine Zuordnung von *Indizes* zu *Werten*.

Literal vom Typ `list`	Vorstellung
`[ 17, 23, 16, 23, 7 ]`	Index 0 1 2 3 4 Wert 17 23 16 23 7

Die Programmzeilen ...

```
a = [ 17, 23, 16, 23, 7 ]
print(a)
print(a[3])
a[3] = a[2] + 14
print(a[3])
print(a)
print(len(a))
```

... erzeugen die Ausgaben ...

```
[ 17, 23, 16, 23, 7 ]
23

30
[ 17, 23, 16, 30, 7 ]
5
```

Die Länge von Array a ist 5. Sie ist der Funktionswert von `len(a)`.

Datentyp `list` (Array) – Folge von Objekten

Die einfachste Vorstellung von einem Array ist eine Zuordnung von *Indizes* zu *Werten*.

Literal vom Typ `list`	Vorstellung
`[ 17, 23, 16, 23, 7 ]`	Index 0 1 2 3 4 Wert 17 23 16 23 7

Die Programmzeilen ...

```
a = [ 17, 23, 16, 23, 7 ]
print(a)
print(a[3])
a[3] = a[2] + 14
print(a[3])
print(a)
print(len(a))
```

... erzeugen die Ausgaben ...

```
[ 17, 23, 16, 23, 7 ]
23

30
[ 17, 23, 16, 30, 7 ]
5
```

Die Länge von Array a ist 5. Sie ist der Funktionswert von `len(a)`.

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

```
[ 17, 14, 23, 32, 25 ]
```

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0
summe = summe + zahlen[0]
summe = summe + zahlen[1]
summe = summe + zahlen[2]
summe = summe + zahlen[3]
summe = summe + zahlen[4]

print(summe/5)
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

```
[ 17, 14, 23, 32, 25 ]
```

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0
summe = summe + zahlen[0]
summe = summe + zahlen[1]
summe = summe + zahlen[2]
summe = summe + zahlen[3]
summe = summe + zahlen[4]

print(summe/5)
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

`[ 17, 14, 23, 32, 25 ]`

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0
summe = summe + zahlen[0]
summe = summe + zahlen[1]
summe = summe + zahlen[2]
summe = summe + zahlen[3]
summe = summe + zahlen[4]

print(summe/5)
```

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for i in range(0,5):
  summe = summe + zahlen[i]



print(summe/5)
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

```
[ 17, 14, 23, 32, 25 ]
```

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0
summe = summe + zahlen[0]
summe = summe + zahlen[1]
summe = summe + zahlen[2]
summe = summe + zahlen[3]
summe = summe + zahlen[4]

print(summe/5)
```

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for i in range(5):
  summe = summe + zahlen[i]



print(summe/5)
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

`[ 17, 14, 23, 32, 25 ]`

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0
summe = summe + zahlen[0]
summe = summe + zahlen[1]
summe = summe + zahlen[2]
summe = summe + zahlen[3]
summe = summe + zahlen[4]

print(summe/5)
```

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for i in range(len(zahlen)):
  summe = summe + zahlen[i]



print(summe/len(zahlen))
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

```
[ 17, 14, 23, 32, 25 ]
```

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0
summe = summe + zahlen[0]
summe = summe + zahlen[1]
summe = summe + zahlen[2]
summe = summe + zahlen[3]
summe = summe + zahlen[4]

print(summe/5)
```

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for i in range(len(zahlen)):
  summe += zahlen[i]



print(summe/len(zahlen))
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

```
[ 17, 14, 23, 32, 25 ]
```

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for z in zahlen:
  summe += z



print(summe/len(zahlen))
```

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for i in range(len(zahlen)):
  summe += zahlen[i]



print(summe/len(zahlen))
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

Literal	Vorstellung

`[ 17, 14, 23, 32, 25 ]`

Index	0	1	2	3	4
Wert	17	14	23	32	25

Aufgabe: berechne den Mittelwert der Werte eines Arrays

Idee für den Algorithmus: Alle Zahlenwerte des Arrays werden summiert.
Abschließend wird die Summe geteilt durch die Anzahl der Zahlen ausgegeben.

```
zahlen = [ 17, 14, 23, 32, 25 ]

summe = 0

for z in zahlen:
  summe += z



print(summe/len(zahlen))
```

```
zahlen = [ 17, 14, 23, 32, 25 ]



print(sum(zahlen)/len(zahlen))
```

Beispiel: Berechnung des Mittelwertes der Werte eines Arrays

```
#-------------------------------------------------------------------------------
# mittelwert1.py
#-------------------------------------------------------------------------------
# Berechne den Mittelwert der Werte, die in einem Array zahlen gespeichert sind.
#-------------------------------------------------------------------------------

# Erzeuge das Array mit den Zahlen.
zahlen = [17, 4, 23, 999, 46, 24, 1, 1, 13, 18]

# summe ist die Summe der bisher aufsummierten Werte.
summe = 0

# Durchlaufe alle Indizes des Arrays zahlen[] und
# summiere die zugeordneten Zahlenwerte auf.
for i in range(len(zahlen)):
  summe = summe + zahlen[i]
  print('i: ' + str(i) + '  summe: ' + str(summe))

# Gib die berechnete Summe geteilt durch die Anzahl der aufsummierten Zahlenwerte aus.
print( 'Der Mittelwert ist ' + str(summe/len(zahlen)) + ' .' )
```

```
#----------------------------
# python3 mittelwert1.py
# i: 0  summe: 17
# i: 1  summe: 21
# i: 2  summe: 44
# i: 3  summe: 1043
# i: 4  summe: 1089
# i: 5  summe: 1113
# i: 6  summe: 1114
# i: 7  summe: 1115
# i: 8  summe: 1128
# i: 9  summe: 1146
# Der Mittelwert ist 114.6 .
```

`zahlen` ist ein Array der Länge 10.

Die Elemente von `zahlen` sind mit `zahlen[0]`, `zahlen[1]`,`zahlen[2]`,..., `zahlen[9]` ansprechbar.

`len(zahlen)` ist die Länge des Arrays `zahlen`.

Das gleiche nochmal – aber anders aufgeschrieben

```
#-------------------------------------------------------------------------------
# mittelwert2.py
#-------------------------------------------------------------------------------
# Berechne den Mittelwert der Werte, die in einem Array gespeichert sind.
#-------------------------------------------------------------------------------

zahlen = [17, 4, 23, 999, 46, 24, 1, 1, 13, 18]
summe = 0    # Die Summe der bisher aufsummierten Zahlenwerte.

# Durchlaufe alle Werte, die im Array zahlen[] gespeichert
# sind, und summiere sie auf.
for z in zahlen:
  summe += z
  print('z: ' + str(z) + '  summe: ' + str(summe))

# Gib die berechnete Summe geteilt durch die Anzahl der aufsummierten Zahlenwerte aus.
print('Der Mittelwert ist ' + str(summe/len(zahlen)) + ' .')
```

```
#------------------------
# python3 mittelwert2.py
# i: 17  summe: 17
# i: 4  summe: 21
# i: 23  summe: 44
# i: 999  summe: 1043
# i: 46  summe: 1089
# i: 24  summe: 1113
# i: 1  summe: 1114
# i: 1  summe: 1115
# i: 13  summe: 1128
# i: 8  summe: 1146
# Der Mittelwert ist 114.6 .
```

Operatoren + und * für den Datentyp list

Die Operation + zwischen list und list schreibt zwei Arrays hintereinander (Konkatenation).

Der Ausdruck `[2, 4, 6] + [170, 14]`
ergibt das neue Array `[2, 4, 6, 170, 14]` .

`a += [3, 4]` hängt die Einträge `3, 4` an das Array `a` an (ähnlich zu `a = a + [3, 4]`).

Die Operation * zwischen list und int schreibt ein Array vielfach hintereinander.

Der Ausdruck `[10, 20, 30] * 4`
ergibt das neue Array `[10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]` .

`a *= 4` hängt das Array a 3mal an a an.

Operatoren + und * für den Datentyp `list`

Die Operation + zwischen `list` und `list` schreibt zwei Arrays hintereinander (Konkatenation).

Der Ausdruck `[2, 4, 6] + [170, 14]`
ergibt das neue Array `[2, 4, 6, 170, 14]` .

`a += [3, 4]` hängt die Einträge `3, 4` an das Array `a` an (ähnlich zu `a = a + [3, 4]`).

Die Operation * zwischen `list` und `int` schreibt ein Array vielfach hintereinander.

Der Ausdruck `[10, 20, 30] * 4`
ergibt das neue Array `[10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]` .

`a *= 4` hängt das Array `a` 3mal an `a` an.

Die grobe Struktur des Programmes

Zurück zum Programmier-Auftrag: Berechnung der mittleren absoluten Abweichung vom Mittelwert

1. lies die Zahlen ein und speichere sie in einem Array
2. summiere die Zahlen auf und berechne ihren Mittelwert
3. summiere die absoluten Abweichungen der Zahlen auf und berechne deren Mittelwert
4. gib das Ergebnis aus

Idee für den Algorithmus:
- lies die Zahlen ein
- berechne den Mittelwert der Zahlen
- berechne die absoluten Abweichungen vom Mittelwert der Zahlen
  und summiere sie auf
- das Ergebnis ist der Quotient aus dieser Summe und der Anzahl der Zahlen

Umsetzung des Einlesens und Speichern der Zahlen

Idee:

- Die Zahlen werden über die Konsole eingegeben.
- Jede Zahl wird durch Eintippen der Zahl gefolgt von `<Enter>` eingegeben.
- Wenn eine Zahl eingegeben wurde, wird sie an ein Array angehängt.
- Die Eingabe der Zahlen wird beendet, indem `<Enter>` (ohne zuvor eine Zahl einzugeben) gedrückt wird.

Grobe Programm-Struktur:

1. Wir nehmen ein Array `zahlen`, um die eingegebenen Zahlen zu speichern. Dieses Array ist am Anfang leer `[ ]`.
2. solange etwas eingegeben wurde, wiederhole:
   2.1 hänge die eingegebene Zahl an das Array `zahlen` an

```
zahlen = []

z = input('Bitte geben Sie eine Zahl ein: ')
while z != '':
  zahlen += [ int(z) ]
  z = input('Bitte geben Sie eine Zahl ein: ')
```

Umsetzung des Einlesens und Speichern der Zahlen

Idee:

- Die Zahlen werden über die Konsole eingegeben.
- Jede Zahl wird durch Eintippen der Zahl gefolgt von `<Enter>` eingegeben.
- Wenn eine Zahl eingegeben wurde, wird sie an ein Array angehängt.
- Die Eingabe der Zahlen wird beendet, indem `<Enter>` (ohne zuvor eine Zahl einzugeben) gedrückt wird.

Grobe Programm-Struktur:

1. Wir nehmen ein Array `zahlen`, um die eingegebenen Zahlen zu speichern. Dieses Array ist am Anfang leer `[ ]`.
2. solange etwas eingegeben wurde, wiederhole:
   - 2.1 hänge die eingegebene Zahl an das Array `zahlen` an

```
zahlen = []

z = input('Bitte geben Sie eine Zahl ein: ')
while z != '':
  zahlen += [ int(z) ]
  z = input('Bitte geben Sie eine Zahl ein: ')
```

Umsetzung des Einlesens und Speichern der Zahlen

Idee:

- Die Zahlen werden über die Konsole eingegeben.
- Jede Zahl wird durch Eintippen der Zahl gefolgt von <Enter> eingegeben.
- Wenn eine Zahl eingegeben wurde, wird sie an ein Array angehängt.
- Die Eingabe der Zahlen wird beendet, indem <Enter> (ohne zuvor eine Zahl einzugeben) gedrückt wird.

Grobe Programm-Struktur:

1. Wir nehmen ein Array `zahlen`, um die eingegebenen Zahlen zu speichern. Dieses Array ist am Anfang leer `[ ]`.
2. solange etwas eingegeben wurde, wiederhole:
   - 2.1 hänge die eingegebene Zahl an das Array `zahlen` an

```
zahlen = []

z = input('Bitte geben Sie eine Zahl ein: ')
while z != '':
  zahlen += [ int(z) ]
  z = input('Bitte geben Sie eine Zahl ein: ')
```

Das Programm mittlere-abweichung.py

```
print('Bitte geben Sie ganze Zahlen ein. Die Eingabe endet mit <Enter>.')

# Die eingegebenen Zahlen werden im Array zahlen gespeichert.
zahlen = []

# Lies Zahlen ein und füge sie zum Array zahlen hinzu.
z = input('Bitte geben Sie eine Zahl ein: ')
while z != '':
  zahlen += [ int(z) ]
  z = input('Bitte geben Sie eine Zahl ein: ')

# Berechne den Mittelwert der eingegebenen Zahlen.
mittelwert = sum(zahlen)/len(zahlen)

# Summiere die absoluten Abweichungen der eingegebenen Zahlen vom Mittelwert auf.
summe = 0
for z in zahlen:
   summe += abs(z-mittelwert)

# Gib den Mittelwert der aufsummierten Werte aus.
print('Die mittlere absolute Abweichung vom Mittelwert ist ' + str(summe/len(zahlen)) + ' .')
```

Das Programm mittlere-abweichung.py

```
print('Bitte geben Sie ganze Zahlen ein. Die Eingabe endet mit <Enter>.')

# Die eingegebenen Zahlen werden im Array zahlen gespeichert.
zahlen = []

# Lies Zahlen ein und füge sie zum Array zahlen hinzu.
z = input('Bitte geben Sie eine Zahl ein: ')
while z != '':
  zahlen += [ int(z) ]
  z = input('Bitte geben Sie eine Zahl ein: ')

# Berechne den Mittelwert der eingegebenen Zahlen.
mittelwert = sum(zahlen)/len(zahlen)

# Summiere die absoluten Abweichungen der eingegebenen Zahlen vom Mittelwert auf.
summe = 0
for z in zahlen:
   summe += abs(z-mittelwert)

# Gib den Mittelwert der aufsummierten Werte aus.
print('Die mittlere absolute Abweichung vom Mittelwert ist ' + str(summe/len(zahlen)) + ' .')
```

```
algo@mm: ~/Python/VL03

mundhenk@home:~/Python/VL03$ python3 mittlere-abweichung.py
Bitte geben Sie ganze Zahlen ein. Die Eingabe endet mit <Enter>.
Bitte geben Sie eine Zahl ein: 1
Bitte geben Sie eine Zahl ein: 3
Bitte geben Sie eine Zahl ein: 5
Bitte geben Sie eine Zahl ein: 9
Bitte geben Sie eine Zahl ein: 7
Bitte geben Sie eine Zahl ein:
Die mittlere absolute Abweichung vom Mittelwert ist 2.4 .
mundhenk@home:~/Python/VL03$
```

Zusammenfassung: Benutzung des Datentyps `list`

Erzeugen eines Arrays:

`[`a_0`, `a_1`, ... , `a_{n-1}`]`	ein Array der Länge n mit den Elementen $a_0, a_1, \ldots, a_{n-1}$
`[]` oder `list()`	ein Array der Länge 0 (ohne Elemente)

Zugriff auf Elemente von Array `a`:

`a[i]`	das `i`-te Element von `a` (`i` hat Typ `int`)	(indizierter Zugriff)
`a[i] = x`	ersetze das `i`-te Element von `a` durch `x`	(indizierte Zuweisung)
`for v in a:`	weise `v` jedes Element von `a` zu	(Durchlaufen)
`a[i:j]`	ein neues Objekt `[a[i], a[i+1], ..., a[j-1]]` vom Typ `list`, `i` hat Default 0 und `j` hat Default `len(a)`	(Slicing)
`v in a`	ergibt `True`, falls Element `v` in Array `a` vorkommt, sonst `False`	

Operationen und Funktionen zum Arbeiten mit Arrays `a` und `b`:

`a + b`	ein neues Objekt vom Typ `list` mit den Elementen von `a` und `b`
`a * i`	ein neues Objekt vom Typ `list` aus `i` Kopien von `a`
`a += b`	hänge `b` an `a` an
`len(a)`	die Anzahl der Elemente von `a`
`sum(a)`	die Summe der Elemente von `a` (sie müssen summierbar sein)
`min(a)`	ein kleinstes Element von `a` (sie müssen vergleichbar sein)
`max(a)`	ein größtes Element von `a` (sie müssen vergleichbar sein)

Programmier-Auftrag: mit welchen Wahrscheinlichkeiten würfelt man $i = 2, 3, \ldots, 7, \ldots, 12$ mit zwei Würfeln?

Wir hatten bereits experimentell die Wahrscheinlichkeit bestimmt, 7 zu würfeln.
Nun wollen wir die Wahrscheinlichkeiten für *alle* möglichen Würfelergebnisse bestimmen.

Idee:

- richte eine Tabelle mit einer Spalte für jedes Würfelergebnis ein
- würfele sehr oft mit zwei Würfeln
  - und mache nach jedem Wurf einen Strich in die Spalte mit dem Würfeleregebnis
- die Ergebnisse sind *Anzahl der Würfe mit Ergebnis i / Anzahl aller Würfe*

Programmier-Auftrag: mit welchen Wahrscheinlichkeiten würfelt man $i = 2, 3, \ldots, 7, \ldots, 12$ mit zwei Würfeln?

Wir hatten bereits experimentell die Wahrscheinlichkeit bestimmt, 7 zu würfeln.
Nun wollen wir die Wahrscheinlichkeiten für *alle* möglichen Würfelergebnisse bestimmen.

Idee:

- richte eine Tabelle mit einer Spalte für jedes Würfelergebnis ein
- würfele sehr oft mit zwei Würfeln
  - und mache nach jedem Wurf einen Strich in die Spalte mit dem Würfeleregebnis
- die Ergebnisse sind *Anzahl der Würfe mit Ergebnis i / Anzahl aller Würfe*

Die grobe Programm-Struktur

1. *Vorbereitung der Experimentreihe:*
   als Tabelle nehmen wir ein Array `tabelle` mit Indizes `2, 3,..., 12`
2. *Wiederholte Durchführung und Auswertung des Experiments:*
   - **2.1** *Experiment:* würfele mit 2 Würfeln
   - **2.2** *Auswertung:* addiere 1 zu `tabelle`[Würfelergebnis]
3. *Auswertung der Experimentreihe:*
   gib die Ergebnisse `tabelle[i] / Anzahl der Experimente` aus

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“ `i` `tabelle`

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	0	0	0	0	0	0	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	0	0	0	0	0

würfele 8

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	1	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	2	0	0	0	0	1	0	0	0	0

⋮ ⋮

würfele 9

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“ | `i` | `tabelle`

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	0	0	0	0	0	0	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	0	0	0	0	0

würfele 8

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	1	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	2	0	0	0	0	1	0	0	0	0

⋮ ⋮

würfele 9

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	0	0	0	0	0	0	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	0	0	0	0	0

würfele 8

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	1	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	2	0	0	0	0	1	0	0	0	0

⋮ ⋮

würfele 9

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	0	0	0	0	0	0	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	0	0	0	0	0

würfele 8

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	1	0	0	0	0	1	0	0	0	0

würfele 3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	0	2	0	0	0	0	1	0	0	0	0

⋮ ⋮

würfele 9

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
	Wert	0	0	0	0	0	0	0	0	0	0	0	0	0

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
würfele	3	Wert	0	0	0	1	0	0	0	0	0	0	0	0	0

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
würfele	8	Wert	0	0	0	1	0	0	0	0	1	0	0	0	0

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
würfele	3	Wert	0	0	0	2	0	0	0	0	1	0	0	0	0

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
würfele	9	Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`												
		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	`i`	`tabelle`

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	i	tabelle

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	i	tabelle
		Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 / Wert: 0 0 0 0 0 0 0 0 0 0 0 0 0
würfele	3	Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 / Wert: 0 0 0 1 0 0 0 0 0 0 0 0 0
würfele	8	Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 / Wert: 0 0 0 1 0 0 0 0 1 0 0 0 0
würfele	3	Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 / Wert: 0 0 0 2 0 0 0 0 1 0 0 0 0
⋮	⋮	⋮
würfele	9	Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 / Wert: 0 0 10 20 30 40 50 60 50 40 30 20 10

Vorstellung vom Ablauf der Experimentreihe

Algorithmus zur Durchführung des Experiments: würfele – das Ergebnis sei `i`

```
tabelle[i] = tabelle[i] + 1
```

„Aktion“	i	tabelle

		Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	0	0	0	0	0	0	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	0	0	0	0	0
würfele	8	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	1	0	0	0	0	1	0	0	0	0
würfele	3	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	0	2	0	0	0	0	1	0	0	0	0
⋮	⋮	⋮													
würfele	9	Index	0	1	2	3	4	5	6	7	8	9	10	11	12
		Wert	0	0	10	20	30	40	50	60	50	40	30	20	10

```
#-------------------------------------------------------------------------------------------------------------------
# wuerfeln.py
#-------------------------------------------------------------------------------------------------------------------
# Das Programm bestimmt für alle Zahlen i=2,3,...,12 experimentell,
# mit welcher Wahrscheinlichkeit sie mit 2 Würfeln gewürfelt werden.
#-------------------------------------------------------------------------------------------------------------------
import random

# Die Experimentreihe wird vorbereitet.
n = int( input('Anzahl der Experimente: ') )  # Die Anzahl der Experimente wird eingelesen.
tabelle = [0]*13          # tabelle[i] speichert, wieoft i gewürfelt wurde

# Die Experimentreihe wird durchgeführt.
for e in range(n):     # es werden n Experimente durchgeführt
  # Das Experiment: zwei Würfel werden geworfen und ihre Summe wird berechnet.
  wuerfelsumme = random.randint(1,6) + random.randint(1,6)
  # Auswertung des Experiments:
  tabelle[ wuerfelsumme ] += 1

# Das Ergebnis der Experimentreihe wird ausgegeben.
for i in range(2,13):
  erfolgsWkeit = tabelle[i]/n
  print('Die Wahrscheinlichkeit, eine ' + str(i) + ' zu würfeln, ist ' + str(erfolgsWkeit) + '.' )
  print('  D.h. etwa einer von ' + str(round(1/erfolgsWkeit)) + ' Würfen ist ' + str(i) + '.' )
```

```
#-----------------------------------------------------------------------------------------------------------------
# python3 wuerfeln.py
# Anzahl der Experimente: 1000000
# Die Wahrscheinlichkeit, eine 2 zu würfeln, ist 0.027759.
#   D.h. etwa einer von 36 Würfen ist 2.
# Die Wahrscheinlichkeit, eine 3 zu würfeln, ist 0.055723.
#   D.h. etwa einer von 18 Würfen ist 3.
# Die Wahrscheinlichkeit, eine 4 zu würfeln, ist 0.083432.
#   D.h. etwa einer von 12 Würfen ist 4.
# Die Wahrscheinlichkeit, eine 5 zu würfeln, ist 0.110923.
#   D.h. etwa einer von 9 Würfen ist 5.
# Die Wahrscheinlichkeit, eine 6 zu würfeln, ist 0.138998.
#   D.h. etwa einer von 7 Würfen ist 6.
# Die Wahrscheinlichkeit, eine 7 zu würfeln, ist 0.166332.
#   D.h. etwa einer von 6 Würfen ist 7.
# Die Wahrscheinlichkeit, eine 8 zu würfeln, ist 0.138795.
#   D.h. etwa einer von 7 Würfen ist 8.
# Die Wahrscheinlichkeit, eine 9 zu würfeln, ist 0.111227.
#   D.h. etwa einer von 9 Würfen ist 9.
# Die Wahrscheinlichkeit, eine 10 zu würfeln, ist 0.083128.
#   D.h. etwa einer von 12 Würfen ist 10.
# Die Wahrscheinlichkeit, eine 11 zu würfeln, ist 0.055919.
#   D.h. etwa einer von 18 Würfen ist 11.
# Die Wahrscheinlichkeit, eine 12 zu würfeln, ist 0.027764.
#   D.h. etwa einer von 36 Würfen ist 12.
```

Programmier-Auftrag: wie oft muss man würfeln, bis man erstmals eine Zahl zum zweiten Mal würfelt?

Idee:

Um diesen Erwartungswert experimentell zu bestimmen, führen wir das Experiment

würfele solange, bis erstmals eine Zahl zum zweiten Mal gewürfelt wird;
das Ergebnis des Experiments ist die Anzahl der Würfe

häufig durch und berechnen den Mittelwert der Ergebnisse.

Wie merken wir uns, welche Zahlen bereits gewürfelt wurden?

Wir nehmen eine Tabelle mit Spalten $1, 2, \ldots, 6$
und machen ein Kreuz in die Spalte, wenn die Zahl gewürfelt wird.

Umsetzung der Idee der Tabelle mit Kreuzen

Wir benutzen ein Array `gewuerfelt` mit Indizes `1,2,...,6`.

Wenn ein „Kreuz“ in Spalte *i* gemacht werden soll,
tragen wir dort den Wahrheitswert `True` ein: `gewuerfelt[i] = True` .

Zu Beginn steht in keiner Spalte ein Kreuz,
d.h. `gewuerfelt[i]` ist `False` (für alle `i`).

Die grobe Programm-Struktur

1. *Vorbereitung der Experimentreihe:*
   als Tabelle nehmen wir ein Array `gewuerfelt` mit Indizes `1,2, ... ,6`
2. *Wiederholte Durchführung und Auswertung des Experiments:*
   - **2.1** *Experiment:* würfele solange, bis eine Zahl zum zweiten Mal gewürfelt wird
   - **2.2** *Auswertung:* addiere die Anzahl der Würfe zu Gesamtanzahl der Würfe
3. *Auswertung der Experimentreihe:*
   gib das Ergebnis *Gesamtanzahl der Würfe / Anzahl der Experimente* aus

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	

Index	0	1	2	3	4	5	6
Wert	False	False	False	False	False	False	False

„Aktion“	`i`	zaehler	`gewuerfelt`
würfele	3	1	

Index	0	1	2	3	4	5	6
Wert	False	False	False	True	False	False	False

„Aktion“	`i`	zaehler	`gewuerfelt`
würfele	5	2	

Index	0	1	2	3	4	5	6
Wert	False	False	False	True	False	True	False

„Aktion“	`i`	zaehler	`gewuerfelt`
würfele	2	3	

Index	0	1	2	3	4	5	6
Wert	False	False	True	True	False	True	False

„Aktion“	`i`	zaehler	`gewuerfelt`
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	

Index	0	1	2	3	4	5	6
Wert	False	False	False	False	False	False	False

würfele | 3 | 1

Index	0	1	2	3	4	5	6
Wert	False	False	False	True	False	False	False

würfele | 5 | 2

Index	0	1	2	3	4	5	6
Wert	False	False	False	True	False	True	False

würfele | 2 | 3

Index	0	1	2	3	4	5	6
Wert	False	False	True	True	False	True	False

würfele | 5 | 4

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

Etwas genauere Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele einmal – das Ergebnis sei `w`
erhöhe einen Zähler für die Würfe um 1
falls `gewuerfelt[w]`,
  dann: beende das Experiment
  sonst: `gewuerfelt[w] = True`
    und würfele weiter (gehe nach oben)

„Aktion“	`i`	zaehler	`gewuerfelt`
		0	Index: 0 1 2 3 4 5 6 / Wert: False False False False False False False
würfele	3	1	Index: 0 1 2 3 4 5 6 / Wert: False False False True False False False
würfele	5	2	Index: 0 1 2 3 4 5 6 / Wert: False False False True False True False
würfele	2	3	Index: 0 1 2 3 4 5 6 / Wert: False False True True False True False
würfele	5	4	

```python
#-----------------------------------------------------------------------------------------------
# doppeltgewuerfelt_v1.py
#-----------------------------------------------------------------------------------------------
# Es wird experimentell bestimmt, wie oft man mit einem 6er-Würfel
# würfeln muss, bis man eine Zahl doppelt gewürfelt hat.
#-----------------------------------------------------------------------------------------------
import random

# Das Experiment wird vorbereitet.
runden = 100000       # die Anzahl der Wiederholungen des Experiments
wurfzaehler = 0       # die Anzahl, wie oft insgesamt gewürfelt wurde

# Das Experiment wird runden mal ausgeführt.
# Es besteht darin, solange zu würfeln, bis eine Zahl zweimal gewürfelt wurde.
for i in range(runden):
  # gewuerfelt[i] ist zu Beginn False und wird auf True gesetzt, wenn i gewürfelt wurde.
  gewuerfelt = [False]*7
  # Es wird wiederholt gewürfelt, bis erstmals eine bereits gewürfelte Zahl gewürfelt wird.
  wurf = random.randint(1,6)           # würfele und zähle den Wurf
  wurfzaehler += 1
  while not gewuerfelt[wurf]:
     gewuerfelt[wurf] = True           # speichere den Wurf in gewuerfelt
     wurf = random.randint(1,6)        # würfele nochmal und zähle den Wurf
     wurfzaehler += 1

# Die Experimente werden ausgewertet.
print('Um eine Zahl zweimal zu würfeln, braucht man im Mittel ' + str(wurfzaehler/runden) + ' Würfe.')
print('Das wurde mit ' + str(runden) + ' Experimenten ermittelt.')
```

```
# python3 doppeltgewuerfelt_v1.py
Um eine Zahl zweimal zu würfeln, braucht man im Mittel 3.78344 Würfe.
Das wurde mit 100000 Experimenten ermittelt.

# python3 doppeltgewuerfelt_v1.py
Um eine Zahl zweimal zu würfeln, braucht man im Mittel 3.77197 Würfe.
Das wurde mit 100000 Experimenten ermittelt.

# python3 doppeltgewuerfelt_v1.py
Um eine Zahl zweimal zu würfeln, braucht man im Mittel 3.77012 Würfe.
Das wurde mit 100000 Experimenten ermittelt.
```

Eine andere Idee zum Speichern der Würfe

Die Idee zur Durchführung des Experiments:

Daten: wir speichern die gewürfelten Zahlen im Array `gewuerfelt`
jede gewürfelte Zahl wird an `gewuerfelt` angehängt

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	False	[3]
würfele	1	False	[3,1]
würfele	4	False	[3,1,4]
würfele	5	False	[3,1,4,5]
würfele	4	True	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	False	[3]
würfele	1	False	[3,1]
würfele	4	False	[3,1,4]
würfele	5	False	[3,1,4,5]
würfele	4	True	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	`3`	`False`	`[3]`
würfele	`1`	`False`	`[3,1]`
würfele	`4`	`False`	`[3,1,4]`
würfele	`5`	`False`	`[3,1,4,5]`
würfele	`4`	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	3	`False`	`[3]`
würfele	1	`False`	`[3,1]`
würfele	4	`False`	`[3,1,4]`
würfele	5	`False`	`[3,1,4,5]`
würfele	4	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

Vorstellung vom Ablauf eines Experiments

Algorithmus zur Durchführung des Experiments:

würfele – das Ergebnis sei `w`
falls `w in gewuerfelt`,
  dann beende das Experiment
  sonst hänge `w` an `gewuerfelt[]` an
    und setze das Experiment fort

„Aktion“	`w`	`w in gewuerfelt`	`gewuerfelt[]`
			`[]`
würfele	`3`	`False`	`[3]`
würfele	`1`	`False`	`[3,1]`
würfele	`4`	`False`	`[3,1,4]`
würfele	`5`	`False`	`[3,1,4,5]`
würfele	`4`	`True`	

Ergebnis: es wurde 5mal gewürfelt, bis erstmals eine Zahl zum zweiten Mal gewürfelt wurde.

```
#-----------------------------------------------------------------------------------------
# doppeltgewuerfelt_v2.py
#-----------------------------------------------------------------------------------------
import random

# Das Experiment wird vorbereitet.
runden = 100000       # die Anzahl der Wiederholungen des Experiments
wurfzaehler = 0       # die Anzahl, wie oft insgesamt gewürfelt wurde

# Das Experiment wird runden mal ausgeführt.
# Es besteht darin, solange zu würfeln, bis eine Zahl zweimal gewürfelt wurde.
for i in range(runden):
  # gewuerfelt speichert die bereits gewürfelten Zahlen
  gewuerfelt = []
  # Es wird wiederholt gewürfelt, bis erstmals eine bereits gewürfelte Zahl gewürfelt wird.
  wurf = random.randrange(1,7)
  wurfzaehler += 1
  while wurf not in gewuerfelt:
     gewuerfelt += [wurf]           # speichere den Wurf in gewuerfelt
     wurf = random.randrange(1,7)   # würfele nochmal
     wurfzaehler += 1

# Die Experimente werden ausgewertet.
print('Um eine Zahl zweimal zu würfeln, braucht man im Mittel ' + str(wurfzaehler/runden) + ' Würfe.')
print('Das wurde mit ' + str(runden) + ' Experimenten ermittelt.')
```

```
# python3 doppeltgewuerfelt_v2.py
Um eine Zahl zweimal zu würfeln, braucht man im Mittel 3.77905 Würfe.
Das wurde mit 100000 Experimenten ermittelt.

# python3 doppeltgewuerfelt_v2.py
Um eine Zahl zweimal zu würfeln, braucht man im Mittel 3.77618 Würfe.
Das wurde mit 100000 Experimenten ermittelt.

# python3 doppeltgewuerfelt_v2.py
Um eine Zahl zweimal zu würfeln, braucht man im Mittel 3.77245 Würfe.
Das wurde mit 100000 Experimenten ermittelt.
```

Wieviele Leute muss man fragen . . . ?

Wir verallgemeinern das Programm zum Würfeln für Würfel mit beliebig vielen Zahlen.

Damit kann man auch das als *Geburtstagsparadoxon* bekannte Experiment ausführen.

Es fragt, was die erwartete Anzahl von Personen ist,
die ich nach ihrem Geburtstag fragen muss,
bis ich zwei Personen gefunden habe, die am gleichen Tag des Jahres Geburtstag haben.

Abstrakter ausgedrückt: man würfelt mit einem Würfel mit den Zahlen 0..364.
Wie lange muss man würfeln, bis man eine Zahl zweimal gewürfelt hat?

Dazu müssen wir `doppeltgewuerfelt.py` so ändern,
dass man mit beliebig großen Würfeln würfeln kann.
Die Größe des Würfels wird von der Kommandozeile gelesen.

```
# geburtstagsparadoxon.py
#-------------------------------------------------------------------------------------------
import random, sys

# Die Versuchsreihe wird vorbereitet.
n = int(sys.argv[1])  # lies die Anzahl n der Tage eines Jahres von der Kommandozeile
runden = 100000       # die Anzahl der Wiederholungen des Experiments
fragenzaehler = 0     # die Anzahl, wie viele Personen gefragt wurden

# Das Experiment wird runden mal ausgeführt.
# Es besteht darin, solange zu fragen, bis zwei gleiche Geburstage geantwortet wurden.
# Die Geburtstage stammen aus dem Bereich 0...n-1.
for i in range(runden):
  # genannt enthält die bereits genannten Geburtstage
  genannt = []
  # Es wird wiederholt gefragt, bis erstmals eine bereits genannte Zahl auftaucht.
  fragenzaehler += 1
  antwort = random.randrange(n)
  while not antwort in genannt:
     # Speichere, dass die Antwort genannt wurde.
     genannt += [antwort]
     # Frage nach der nächsten Antwort.
     fragenzaehler += 1
     antwort = random.randrange(n)

# Die Versuchsreihe wird ausgewertet.
print('Um auf einem Planeten mit einem ' + str(n) + '-Tage-Jahr zwei Personen mit gleichem')
print('Geburtstag zu finden, braucht man im Mittel ' + str(fragenzaehler/runden) + ' Personen.')
print('Das wurde mit ' + str(runden) + ' Experimenten ermittelt.')
```

Wir können jetzt das Geburtstagsparadoxon für Erden-, Merkur-, Mars- und Neptunbewohner experimentell ausprobieren ...

```
# python3 geburtstagsparadoxon.py 365
Um auf einem Planeten mit einem 365-Tage-Jahr zwei Personen mit gleichem
Geburtstag zu finden, braucht man im Mittel 24.54053 Personen.
Das wurde mit 100000 Experimenten ermittelt.

# python3 geburtstagsparadoxon.py 88
Um auf einem Planeten mit einem 88-Tage-Jahr zwei Personen mit gleichem
Geburtstag zu finden, braucht man im Mittel 12.40962 Personen.
Das wurde mit 100000 Experimenten ermittelt.

# python3 geburtstagsparadoxon.py 687
Um auf einem Planeten mit einem 687-Tage-Jahr zwei Personen mit gleichem
Geburtstag zu finden, braucht man im Mittel 33.44678 Personen.
Das wurde mit 100000 Experimenten ermittelt.

# python3 geburtstagsparadoxon.py 60265
Um auf einem Planeten mit einem 60265-Tage-Jahr zwei Personen mit gleichem
Geburtstag zu finden, braucht man im Mittel 309.387 Personen.
Das wurde mit 10000 Experimenten ermittelt.
```

Allgemeines über Variablen, Referenzen und Objekte in Python

Die „grobe Vorstellung“ bei den informellen Programmabläufen über die Speicherung von Daten stimmt natürlich nicht – wir verfeinern die Vorstellung jetzt.

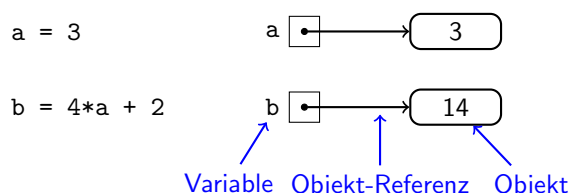
Die Daten werden durch *Objekte* repräsentiert.
Jedes Objekt hat eine *Identität*, einen *Typ* und einen *Wert*.

- Die *Identität* bestimmt das Objekt eindeutig.
  (Vorstellung: Adresse des Objektes im Speicher des Computers.)
- Der *Typ* beschreibt das Verhalten des Objektes
  (d.h. die Werte, die es annehmen kann, und die Operationen, die auf ihm erlaubt sind).
- Der *Wert* ist der Datentyp-Wert, den es repräsentiert/speichert.
  Bsp.: ein Objekt vom Typ `int` kann den Wert 99 speichern.

Eine *Objekt-Referenz* ist eine konkrete Darstellung der Identität des Objektes.
Python-Programme benutzen Objekt-Referenzen um

- auf den Wert des Objektes zuzugreifen oder ihn zu ändern,
- auf die Objekt-Referenz zuzugreifen oder sie zu ändern.

- Ein *Literal* weist Python an, ein Objekt mit einem bestimmten Wert zu erzeugen.
  Bsp.: das Literal 99 weist Python an, ein `int`-Objekt mit Wert 99 zu erzeugen.
- Eine *Variable* ist ein Name für eine Objekt-Referenz.
- Ein *Ausdruck* weist Python an, ein Objekt mit dem Wert des Ausdrucks zu liefern.
  Bsp.: wenn das an die Variable `a` gebundene `int`-Objekt den Wert 3 hat,
  dann weist der Ausdruck `4*a+2` Python an, ein `int`-Objekt mit Wert 14 zu liefern.
- Eine *Zuweisung* ⟨Variable⟩=⟨Ausdruck⟩ weist Python an,
  ein Objekt mit dem Wert des Ausdrucks zu liefern und die Variable daran zu binden.
  Bsp.: wenn das an die Variable `a` gebundene Objekt den Wert 3 hat,
  dann wird durch den Ausdruck `4*a+2` ein Objekt mit dem Wert 14 geliefert und
  durch die Zuweisung `b = 4*a+2` wird es über die Objekt-Referenz an Variable `b` gebunden.



Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```



```
b = 47
```

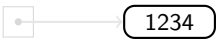


```
c = a + b
```

Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```



```
c = a + b
```

Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```


```
b = 47
```

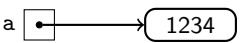


```
c = a + b
```

Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```

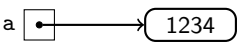


```
c = a + b
```

Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```

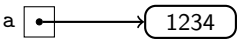


```
c = a + b
```

Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```



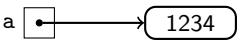

```
b = 47
```


```
c = a + b
```

Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```



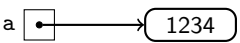

```
b = 47
```


```
c = a + b
```

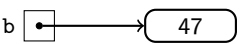
Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```

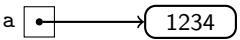



```
c = a + b
```

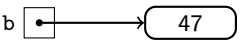
Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```

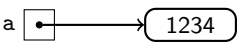



```
c = a + b
```

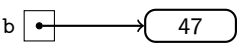
Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```

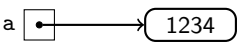



```
c = a + b
```

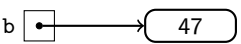
Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

```
a = 1234
```




```
b = 47
```

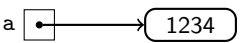



```
c = a + b
```

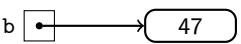
Wirkung eines Programms auf dem Objekt-Level

Ein 3-zeiliges Python-Programm und dessen Wirkung auf dem Objekt-Level:

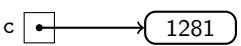
```
a = 1234
```



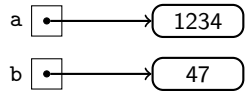

```
b = 47
```



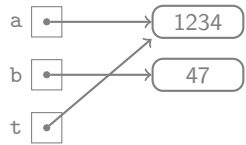

```
c = a + b
```



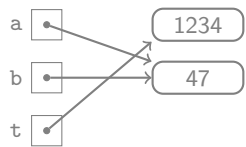
Vertauschung der Werte zweier Variablen.



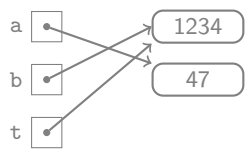

```
t = a
```



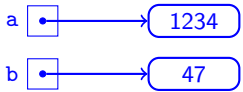

```
a = b
```



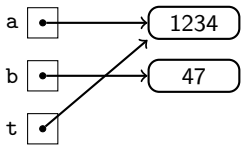

```
b = t
```



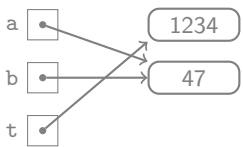
Vertauschung der Werte zweier Variablen.



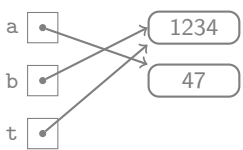

```
t = a
```



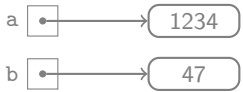

```
a = b
```



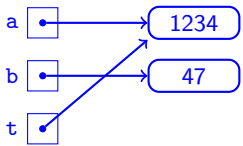

```
b = t
```



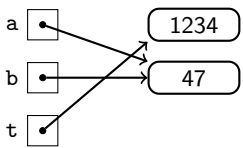
Vertauschung der Werte zweier Variablen.



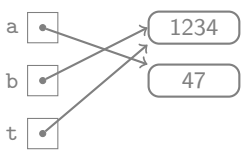

```
t = a
```



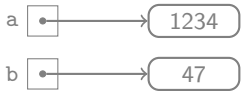

```
a = b
```



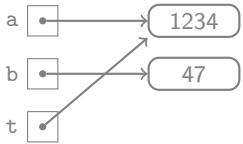

```
b = t
```



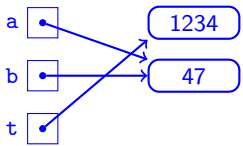
Vertauschung der Werte zweier Variablen.



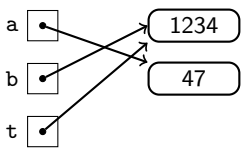

```
t = a
```




```
a = b
```




```
b = t
```



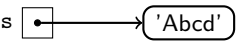
Nicht-veränderbare Datentypen

`int`, `float`, `string` und `bool`

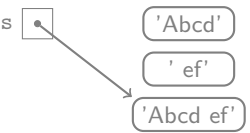
Objekte der Standard-Typen `int`, `float`, `string` und `bool` sind *nicht veränderbar*.

Wenn einer Variablen ein Wert eines Ausdrucks eines dieser Typen zugewiesen wird, dann wird die Bindung der Variablen geändert.

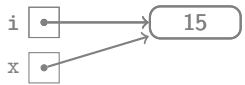
```
s = 'Abcd'
```



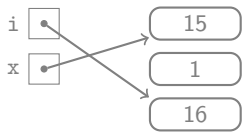

```
s = s + ' ef'
```




```
i = 15
x = i
```




```
i = i + 1
```



Nicht-veränderbare Datentypen

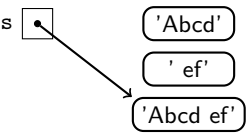
`int`, `float`, `string` und `bool`

Objekte der Standard-Typen `int`, `float`, `string` und `bool` sind *nicht veränderbar*.

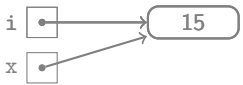
Wenn einer Variablen ein Wert eines Ausdrucks eines dieser Typen zugewiesen wird, dann wird die Bindung der Variablen geändert.

```
s = 'Abcd'
```

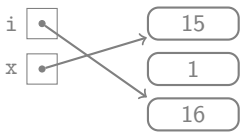

```
s = s + ' ef'
```




```
i = 15
x = i
```




```
i = i + 1
```



Nicht-veränderbare Datentypen

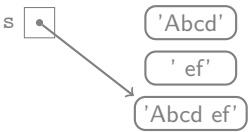
`int`, `float`, `string` und `bool`

Objekte der Standard-Typen `int`, `float`, `string` und `bool` sind *nicht veränderbar*.

Wenn einer Variablen ein Wert eines Ausdrucks eines dieser Typen zugewiesen wird, dann wird die Bindung der Variablen geändert.

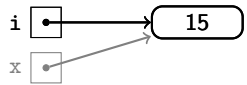
```
s = 'Abcd'
```

```
s = s + ' ef'
```

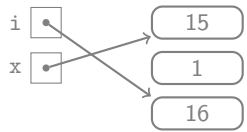



```
i = 15
```

```
x = i
```




```
i = i + 1
```



Nicht-veränderbare Datentypen

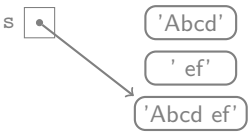
`int`, `float`, `string` und `bool`

Objekte der Standard-Typen `int`, `float`, `string` und `bool` sind *nicht veränderbar*.

Wenn einer Variablen ein Wert eines Ausdrucks eines dieser Typen zugewiesen wird, dann wird die Bindung der Variablen geändert.

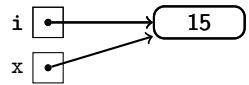
```
s = 'Abcd'
```

```
s = s + ' ef'
```

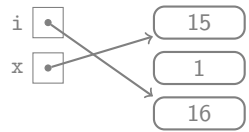



```
i = 15
```

```
x = i
```




```
i = i + 1
```



Nicht-veränderbare Datentypen

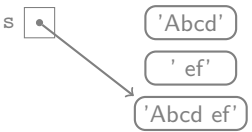
`int`, `float`, `string` und `bool`

Objekte der Standard-Typen `int`, `float`, `string` und `bool` sind *nicht veränderbar*.

Wenn einer Variablen ein Wert eines Ausdrucks eines dieser Typen zugewiesen wird, dann wird die Bindung der Variablen geändert.

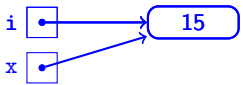
```
s = 'Abcd'
```

```
s = s + ' ef'
```

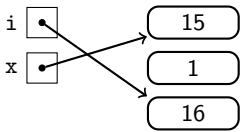



```
i = 15
```

```
x = i
```

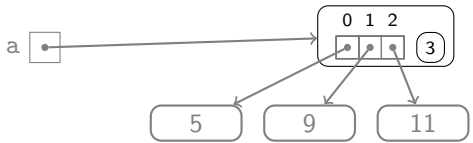



```
i = i + 1
```

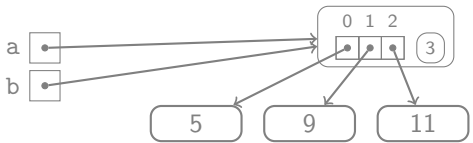


`list` ist ein veränderbarer Datentyp

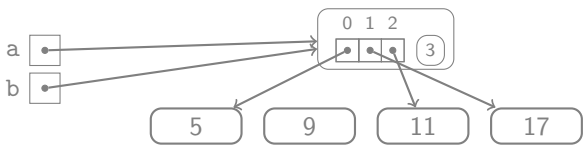
```
a = [5, 9, 11]
```




```
b = a
```

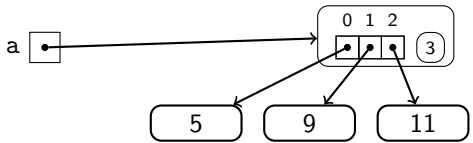



```
b[1] = 17
```

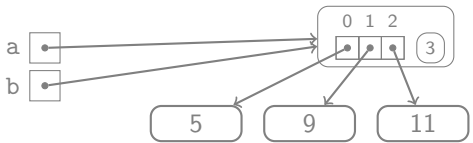


`list` ist ein veränderbarer Datentyp

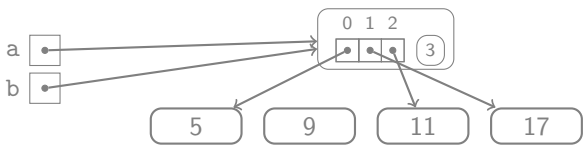
```
a = [5, 9, 11]
```




```
b = a
```

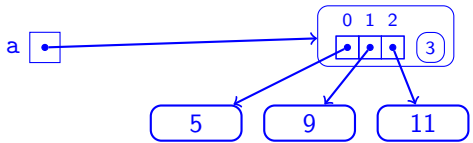



```
b[1] = 17
```

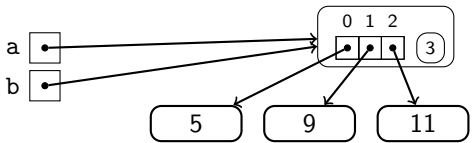


`list` ist ein veränderbarer Datentyp

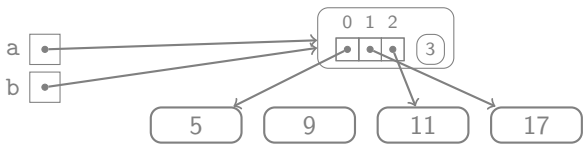
```
a = [5, 9, 11]
```




```
b = a
```

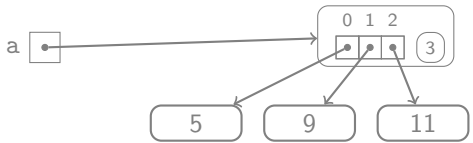



```
b[1] = 17
```

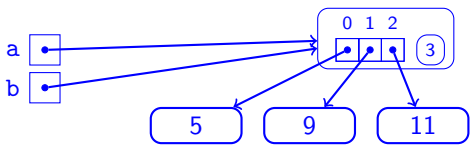


`list` ist ein veränderbarer Datentyp

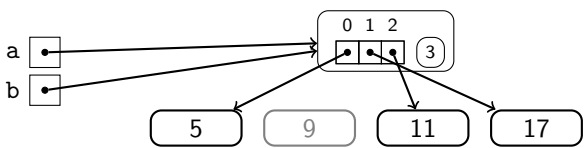
```
a = [5, 9, 11]
```




```
b = a
```

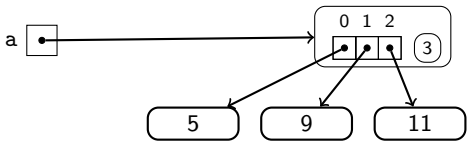



```
b[1] = 17
```



Kopieren eines `list`-Objektes

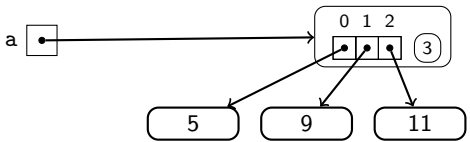
```
a = [5, 9, 11]
```



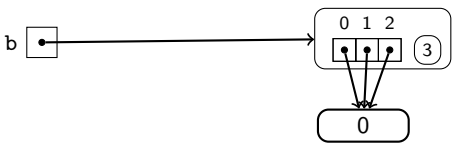

```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```

Kopieren eines `list`-Objektes

```
a = [5, 9, 11]
```

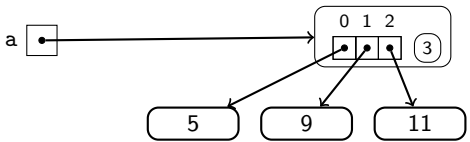



```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```

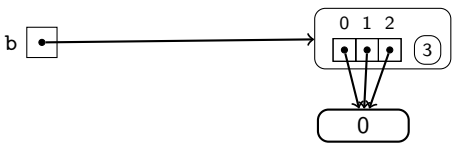


Kopieren eines `list`-Objektes

```
a = [5, 9, 11]
```

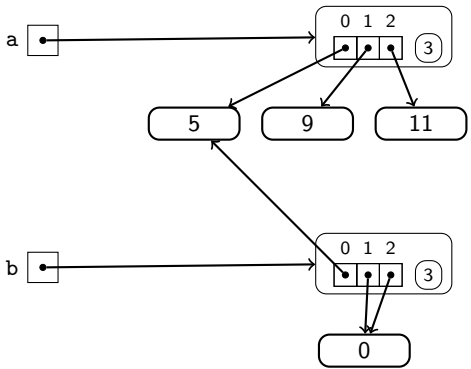



```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```



Kopieren eines `list`-Objektes

```
a = [5, 9, 11]
```

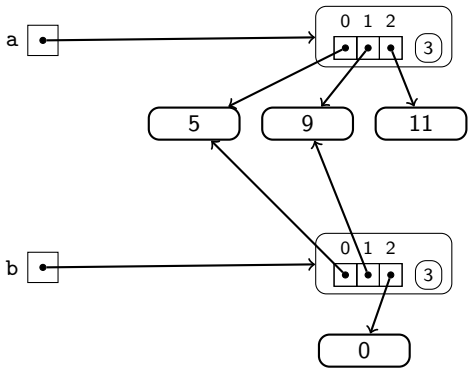



```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]
```

```
b[1] = 17
```

Kopieren eines `list`-Objektes

```
a = [5, 9, 11]
```

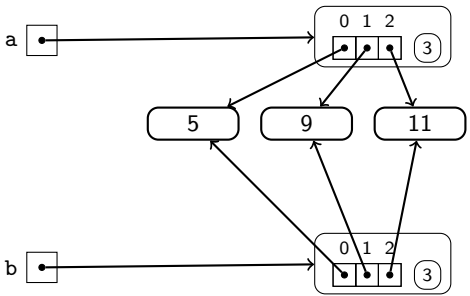



```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]
```

```
b[1] = 17
```

Kopieren eines `list`-Objektes

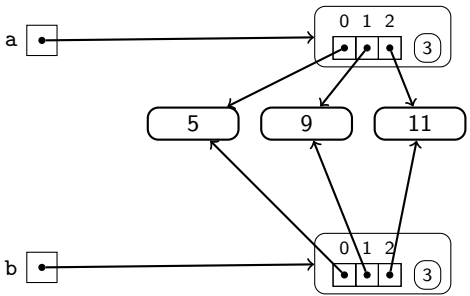
```
a = [5, 9, 11]
```




```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```

Kopieren eines `list`-Objektes

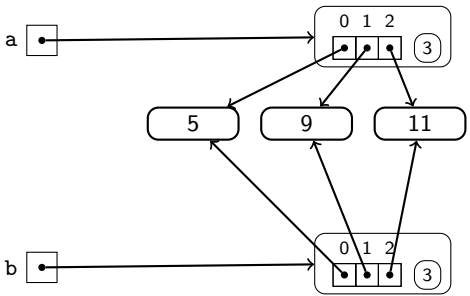
```
a = [5, 9, 11]
```




```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```

Kopieren eines `list`-Objektes

```
a = [5, 9, 11]
```

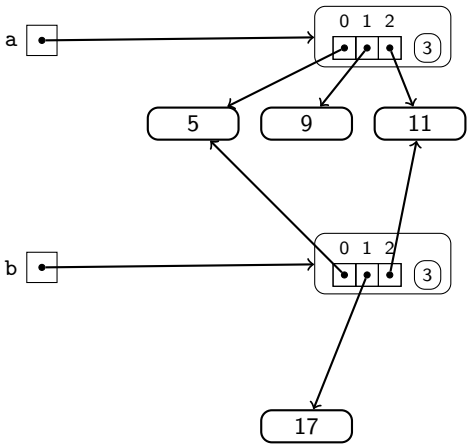



```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```

17

Kopieren eines `list`-Objektes

```
a = [5, 9, 11]
```




```
b = [0]*len(a)

for i in range(len(a)):
    b[i] = a[i]

b[1] = 17
```

```
#-------------------------------------------------------------
# array-1d-kopieren.py   Drei Wege, ein 1d-Array zu kopieren.
#-------------------------------------------------------------

a = [1, 2, 3, 4, 5]

# Kopiere Array a ins Array b.
# Dazu wird zuerst ein Array der Länge len(a) erzeugt.
b = [0] * len(a)
print('Länge von b: ' + str(len(b)))
# Anschließend wird für jeden Index der Wert eingetragen.
for i in range(len(a)):  b[i] = a[i]
print('b: ' + str(b))

# Kopiere Array a ins Array c.
# Dazu wird zuerst ein Array der Länge 0 erzeugt.
c = []
# Anschließend wird das Array für jeden Index i um den Wert a[i] verlängert.
for i in range(len(a)):
  print('Länge von c: ' + str(len(c)))
  c += [a[i]]     # hänge Wert a[i] an das Array c an
print('c: ' + str(c))

# Abschließend die kürzeste Variante.
d = a[:]   # das ist Abkürzung von  d = a[0:len(a)]
print('d: ' + str(d))
```

```
#----------------------------
# python3 array-1d-kopieren.py
# Länge von b: 5
# b: [1, 2, 3, 4, 5]
# Länge von c: 0
# Länge von c: 1
# Länge von c: 2
# Länge von c: 3
# Länge von c: 4
# c: [1, 2, 3, 4, 5]
# d: [1, 2, 3, 4, 5]
```

Der Datentyp `tuple` – Tupel

`tuple` ist wie `list` (Array), aber *nicht-veränderbar*.

Literale vom Typ `tuple` werden mit *runden Klammern* beschrieben:

```
( 17, 4, 23, 12 )          ( 'a', )
```

Benutzung von Elementen von Tupeln geht wie bei Arrays:

```
t = ('a', 1, 'b')
b = t[2]
```

Da Objekte vom Typ `tuple` nicht-veränderbar sind, geht Folgendes nicht:

```
t[2] = 42
```

Gerne benutzt wird das *Unpacking* (geht auch bei Arrays):

```
a = 15
b = 'hello'
( b, a ) = ( a, b )
```

vertauscht die Werte von `a` und `b`.

`(a,b)` auf der rechten Seite der Zuweisung
erzeugt ein Tupel mit den Werten von `a` und `b` (*packing*).

`(b,a)` auf der linken Seite der Zuweisung
packt das Tupel von der rechten Seite aus (*unpacking*):
Variable `b` wird als Wert das erste Element des „rechten“ Tupels zugewiesen
Variable `a` erhält als Wert das nächste Element des „rechten“ Tupels.

Als Kurzschreibweise können die Klammern weggelassen werden.

```
b, a = a, b
```

Nochmal was zum Datentyp `string`

Ein String ist wie ein Tupel aus Zeichen.

Also ist Folgendes mit String-Variable `s` möglich:

- `s[i]` das Zeichen mit Index `i`
- `s[i:j]` die Zeichen mit Indizes `i...j-1`
- `c in s` ergibt `True`, falls Zeichen `c` in `s` vorkommt
- `for c in s: ...` durchläuft alle Zeichen in `s`

Zusammenfassung

Wir kennen den Datentyp `list` (Array),
kennen den Unterschied zwischen veränderbaren und nicht-veränderbaren Datentypen,
können 1-dimensionale Arrays erzeugen und auf ihre Elemente zugreifen,
kennen die Probleme beim Kopieren von Arrays,
und kennen die „Variante“ `tuple` (Tupel).

Wir haben gesehen, dass es beim Entwickeln von Programmen
einfacher sein kann, erst ein Programm für ein einfaches Beispiel zu schreiben
und das dann zu verallgemeinern.