Vorlesung 04

Das Kapitel über Arrays wird fortgesetzt.
In der letzten Vorlesung wurden 1-dimensionale Arrays betrachtet.
Jetzt geht es weiter mit 2-dimensionalen Arrays.

Wir werden drei Beispiele anschauen:

- Umgang mit einem 2-dimensionalen Array.
- Erzeugen eines 2-dimensionalen Arrays.
- Benutzung eines 2-dimensionalen Arrays.

Programmier-Auftrag: werte Punktetabellen statistisch aus

2-dimensionale Arrays

Über die Ergebnisse der Übungsblätter führen wir Buch mittels einer Tabelle.

Spalten

Zeilen	Name	Blatt 1	Blatt 2	Blatt 3	Blatt 4	Blatt 5
	A	20	15	12	8	13
	B	12	0	20	20	8
	C	13	12	15	11	9
	D	18	19	17	18	20
	E	11	12	0	12	13
	F	16	13	15	18	20

Wir möchten zu jedem Namen die mittlere Punktzahl in allen Übungsblättern und die mittlere Punktzahl jedes Übungsblattes wissen.

Die Idee

Name	Blatt 1	Blatt 2	Blatt 3	Blatt 4	Blatt 5
A	20	15	12	8	13
B	12	0	20	20	8
C	13	12	15	11	9
D	18	19	17	18	20
E	11	12	0	12	13
F	16	13	15	18	20

Wir speichern die Zahlenwerte der Tabelle als 2-dimensionales Array.

Jeder Eintrag im Array `tabelle` ist ein Array, das für eine Zeile der Tabelle steht.

```
tabelle = [  [ 20, 15, 12, 8,  13 ],
             [ 12, 0,  20, 20, 8  ],
             [ 13, 12, 15, 11, 9  ],
             [ 18, 19, 17, 18, 20 ],
             [ 11, 12, 0,  12, 13 ],
             [ 16, 13, 15, 18, 20 ]  ]
```

`tabelle[2]` ist `[ 13, 12, 15, 11, 9 ]`.

`tabelle[2][3]` ist `11`.

Die Idee

Name	Blatt 1	Blatt 2	Blatt 3	Blatt 4	Blatt 5
A	20	15	12	8	13
B	12	0	20	20	8
C	13	12	15	11	9
D	18	19	17	18	20
E	11	12	0	12	13
F	16	13	15	18	20

Wir speichern die Zahlenwerte der Tabelle als 2-dimensionales Array.

Jeder Eintrag im Array `tabelle` ist ein Array, das für eine Zeile der Tabelle steht.

```
tabelle = [  [ 20, 15, 12, 8,  13 ],
             [ 12, 0,  20, 20, 8  ],
             [ 13, 12, 15, 11, 9  ],
             [ 18, 19, 17, 18, 20 ],
             [ 11, 12, 0,  12, 13 ],
             [ 16, 13, 15, 18, 20 ]  ]
```

`tabelle[2]` ist `[ 13, 12, 15, 11, 9 ]`.

`tabelle[2][3]` ist `11`.

Die Struktur des Programmes

1. Erzeuge die Tabelle mit den Punkten aus den Übungsaufgaben.
2. Berechne von jeder Zeile der Tabelle den Mittelwert
   und gib diese Mittelwerte aus.
3. Berechne von jeder Spalte der Tabelle den Mittelwert
   und gib diese Mittelwerte aus.

Das Programm tabelle-auswerten.py

```
tabelle = [  [ 20, 15, 12, 8,  13],
             [ 12, 0,  20, 20, 8 ],
             [ 13, 12, 15, 11, 9 ],
             [ 18, 19, 17, 20, 12],
             [ 11, 12, 0,  12, 13],
             [ 16, 13, 15, 18, 20]  ]

# Berechne den Mittelwert jeder Zeile der Tabelle und gib ihn aus.
for zeilennr in range(len(tabelle)):
  m = sum(tabelle[zeilennr])/len(tabelle[zeilennr])
  print( 'Zeile ' + str(zeilennr) + ' hat Mittelwert ' + str(m) + ' .' )

# Berechne den Mittelwert jeder Spalte der Tabelle und gib ihn aus.
# Dabei gehen wir davon aus, dass alle Zeilen gleichviele Spalten haben.
for spaltennr in range(len(tabelle[0])):
  spaltensumme = 0
  for zeilennr in range(len(tabelle)):
    spaltensumme += tabelle[zeilennr][spaltennr]
  print('Spalte ' + str(spaltennr) + ' hat Mittelwert ' + str(spaltensumme/len(tabelle)) + ' .')
```

```
$ python3 tabelle-auswerten.py

Zeile 0 hat Mittelwert 13.6 .
Zeile 1 hat Mittelwert 12.0 .
Zeile 2 hat Mittelwert 12.0 .
Zeile 3 hat Mittelwert 17.2 .
Zeile 4 hat Mittelwert 9.6 .
Zeile 5 hat Mittelwert 16.4 .
Spalte 0 hat Mittelwert 15.0 .
Spalte 1 hat Mittelwert 11.833333333333334 .
Spalte 2 hat Mittelwert 13.166666666666666 .
Spalte 3 hat Mittelwert 14.833333333333334 .
Spalte 4 hat Mittelwert 12.5 .
```

```
tabelle = [  [ 20, 15, 12, 8,  13],
             [ 12, 0,  20, 20, 8 ],
             [ 13, 12, 15, 11, 9 ],
             [ 18, 19, 17, 20, 12],
             [ 11, 12, 0,  12, 13],
             [ 16, 13, 15, 18, 20]  ]
```

Programmier-Auftrag: erzeuge ein 2-dimensionales Array

Erzeuge ein Array mit z Zeilen und s Spalten.
Der Eintrag mit Zeilen-Index *i* und Spalten-Index *j* soll `'i,j'` sein.

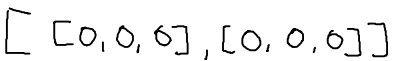
Vorstellung:

Zeilenindizes \ Spaltenindizes	0	1	2	3	4
0	'0,0'	'0,1'	'0,2'	'0,3'	'0,4'
1	'1,0'	'1,1'	'1,2'	'1,3'	'1,4'
2	'2,0'	'2,1'	'2,2'	'2,3'	'2,4'
3	'3,0'	'3,1'	'3,2'	'3,3'	'3,4'

Wie Python das Array mit 4 Zeilen und 5 Spalten ausgeben soll (so ungefähr):

```
[ ['0,0', '0,1', '0,2', '0,3', '0,4'],
  ['1,0', '1,1', '1,2', '1,3', '1,4'],
  ['2,0', '2,1', '2,2', '2,3', '2,4'],
  ['3,0', '3,1', '3,2', '3,3', '3,4'] ]
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

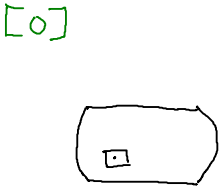
```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

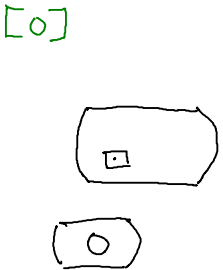
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

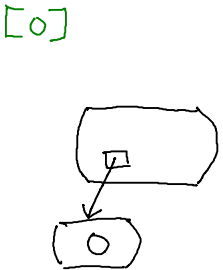
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

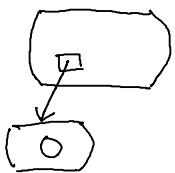


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

[0]*3

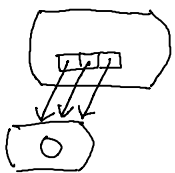


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

[0]*3

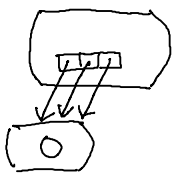


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

b = [0] * 3

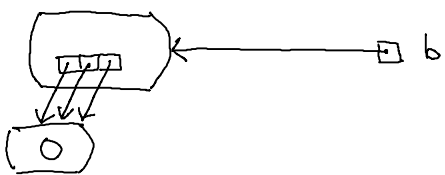


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

b = [0] * 3

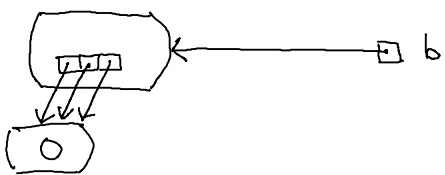


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

b = [0] * 3

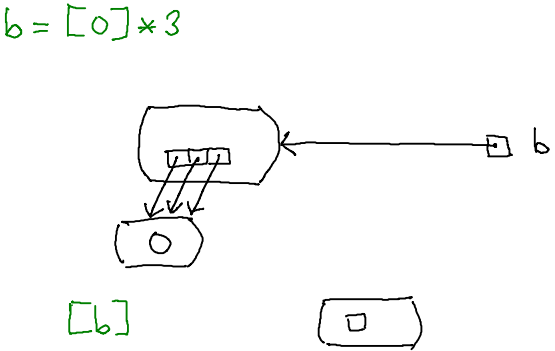


[b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

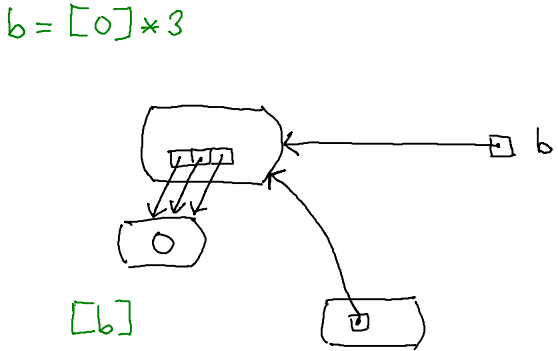
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

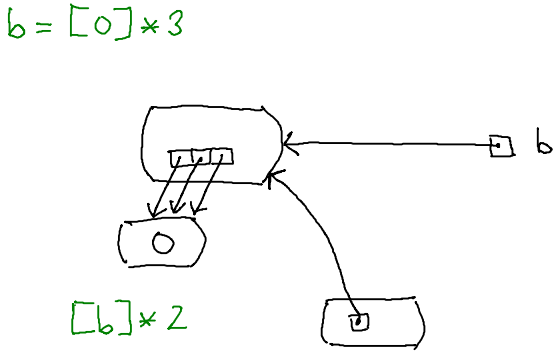
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

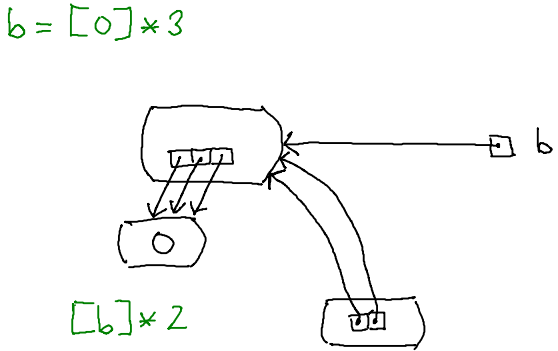
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

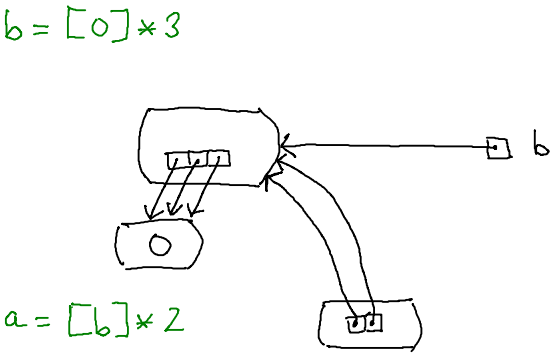
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

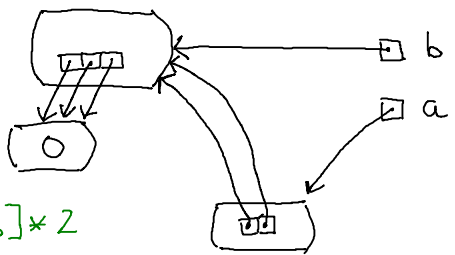


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

b = [0] * 3



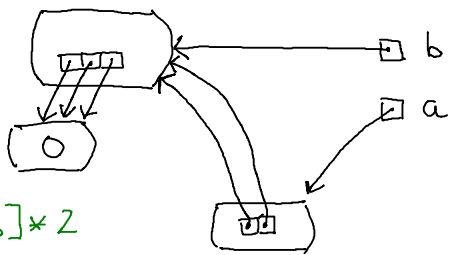
a = [b] * 2

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

b = [0] * 3



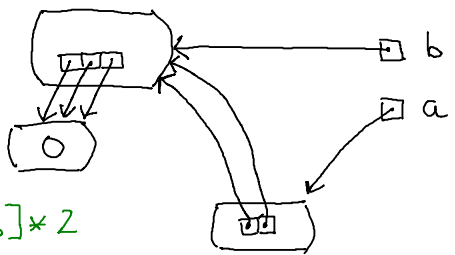
a = [b] * 2

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2

a[1][0] = 5        # Test
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen

b = [0] * 3



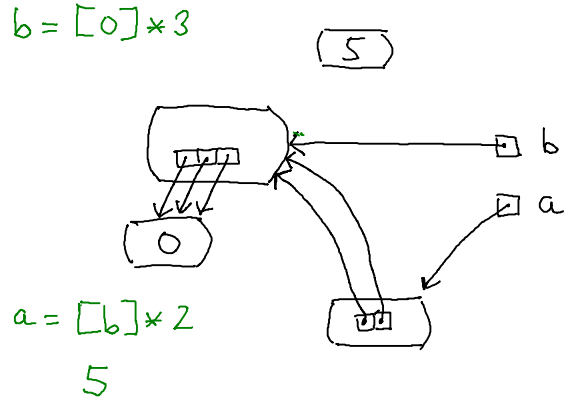
a = [b] * 2

5

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2

a[1][0] = 5        # Test
```

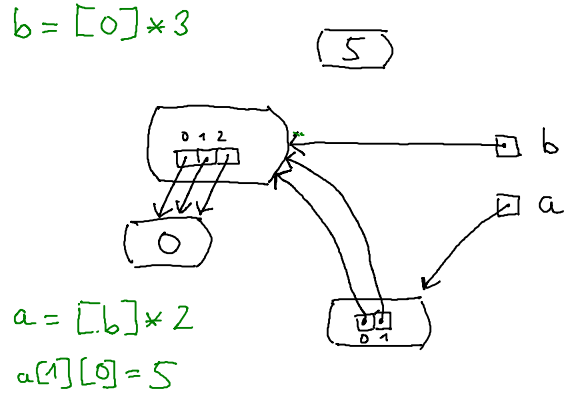
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2

a[1][0] = 5        # Test
```

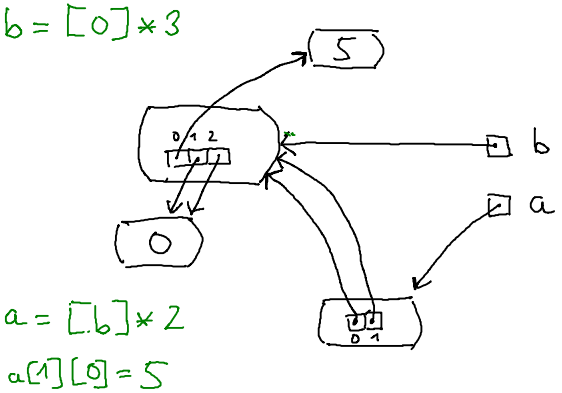
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2

a[1][0] = 5          # Test
```

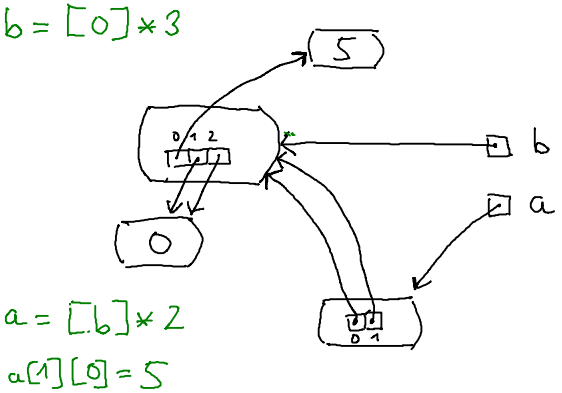
Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2

a[1][0] = 5          # Test
```

Fehlerhafter Versuch, alle Plätze in einem 2d-Array zu erzeugen



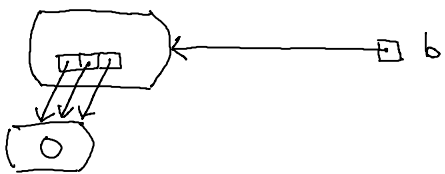
Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b] * 2

a[1][0] = 5          # Test
```

Jedes Array,
das Element des 2d-Arrays a ist,
muss „individuell“ erzeugt werden.

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3

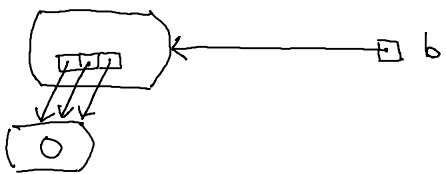


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



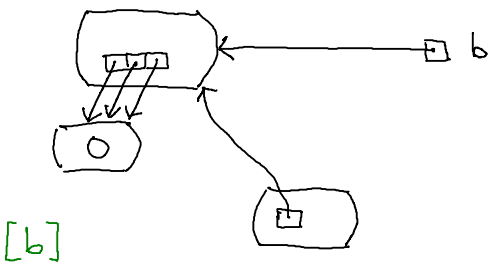
[b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3

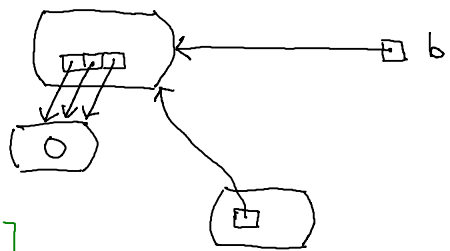


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



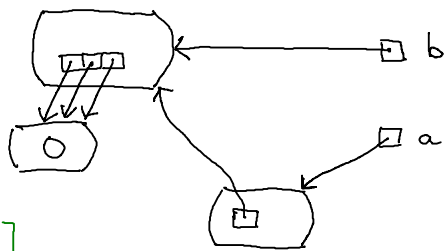
a = [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



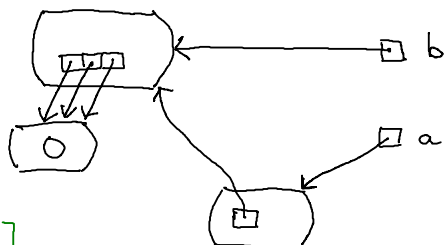
a = [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

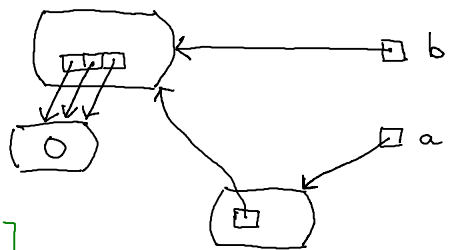
[0] * 3

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3
```

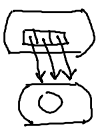
Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

[0] * 3

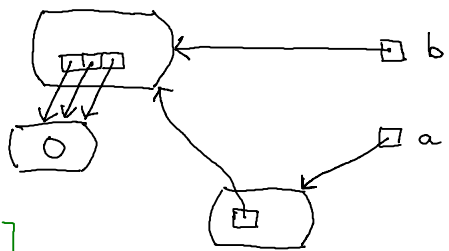


Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

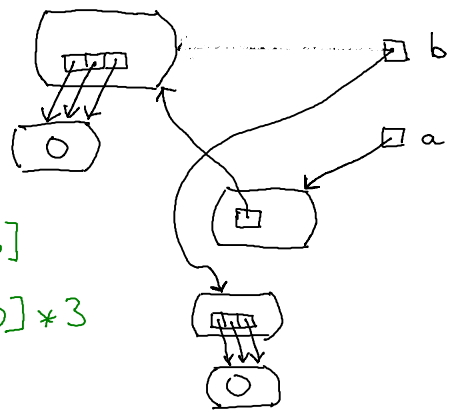
b = [0] * 3

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

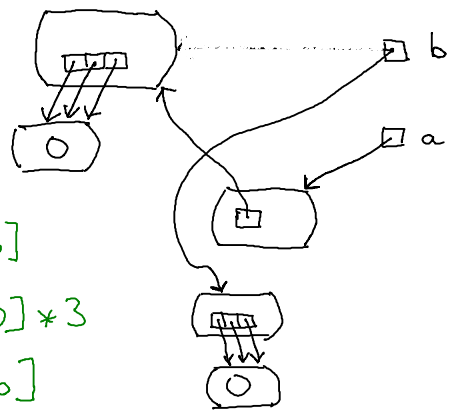
b = [0] * 3

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

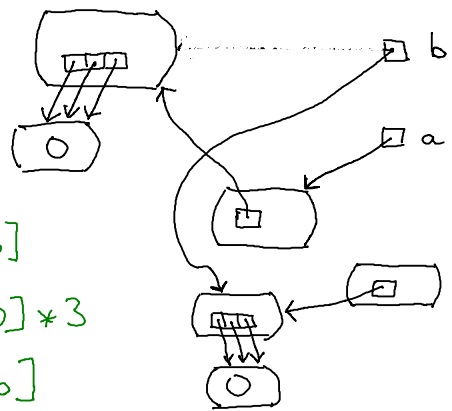
[b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

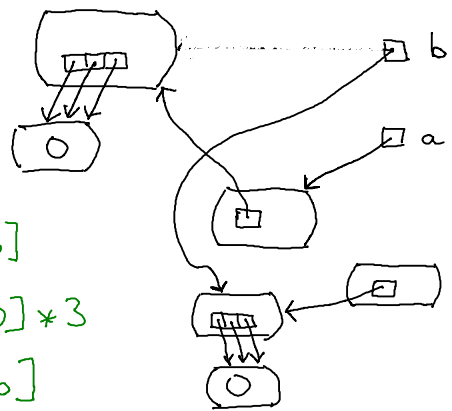
[b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

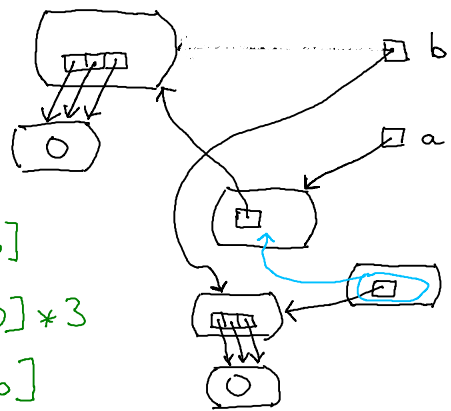
a += [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

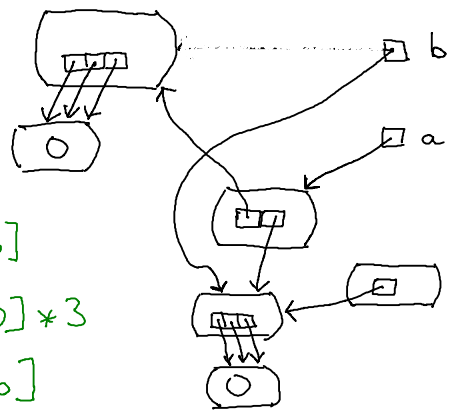
a += [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

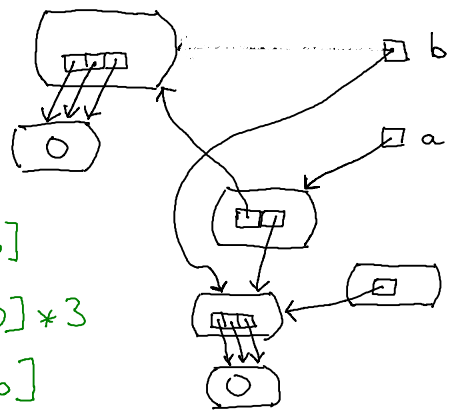
a += [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

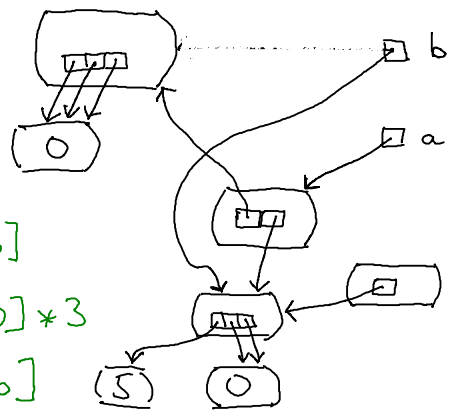
a += [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]

a[1][0] = 5      # Test
```

Beispiel: erzeuge alle Plätze in einem 2d-Array

b = [0] * 3



a = [b]

b = [0] * 3

a += [b]

Erzeuge ein 2d-Array a
mit 2 Zeilen und 3 Spalten,
also mit 2 Elementen,
die 3-elementige Arrays sind.

```
b = [0] * 3

a = [b]

b = [0] * 3

a += [b]

a[1][0] = 5      # Test
```

Die Idee

Programmier-Auftrag:

Erzeuge ein Array mit `z` Zeilen und `s` Spalten.

Der Eintrag mit Zeilen-Index *i* und Spalten-Index *j* soll `'i,j'` sein.

```
[ ['0,0', '0,1', '0,2', '0,3', '0,4'],
  ['1,0', '1,1', '1,2', '1,3', '1,4'],
  ['2,0', '2,1', '2,2', '2,3', '2,4'],
  ['3,0', '3,1', '3,2', '3,3', '3,4'] ]
```

- Lies die Zeilen- und Spaltenanzahl ein.
- Erzeuge das 2d-Array mit allen benötigten Plätzen.
- Trage an jedem Platz den passenden String ein.

Die grobe Programm-Struktur

1. Lies die Zeilen- und Spaltenanzahl ein.
2. Das Ergebnis am Ende ist das Array a, das am Anfang leer ist.
3. Erzeuge alle Plätze im Array a wie folgt:

   Für jede Zeile:

   3.1 Erzeuge ein Array mit einem Platz für jede Spalte.
   3.2 Hänge dieses Array an a an.

4. Gehe durch alle Plätze des 2d-Arrays a und trage den passenden String ein.

Programm array-2d-beispiel.py

```
# array-2d-beispiel.py
#-----------------------------------------------------------------------------------------------------------
# Erzeuge ein 2d-Array a mit z Zeilen aus s Spalten, und schreibe an Stelle a[i][j] den String 'i,j'.
#-----------------------------------------------------------------------------------------------------------
import sys
# Lies die Zeilenzahl z und die Spaltenzahl s von der Kommandozeile.
z = int(sys.argv[1])
s = int(sys.argv[2])
# Zu Beginn ist das Array a leer.
a = []
# Die Zeilen des Arrays werden der Reihe nach erzeugt und an a angehängt.
for i in range(z):
  b = [0]*s                    # Die Zeile hat s Einträge (Spalten).
  a += [ b ]                   # Sie wird an a angehängt.
# Nun sind alle Plätze im Array a erzeugt.
# Sie werden durchlaufen und der passende String wird eingetragen.
for i in range(len(a)):
  for j in range(len(a[i])):
    a[i][j] = '%d,%d' % (i,j)
print(a)
```

Programm array-2d-beispiel.py

```
# array-2d-beispiel.py
#-------------------------------------------------------------------------------------------------------------
# Erzeuge ein 2d-Array a mit z Zeilen aus s Spalten, und schreibe an Stelle a[i][j] den String 'i,j'.
#-------------------------------------------------------------------------------------------------------------
import sys
# Lies die Zeilenzahl z und die Spaltenzahl s von der Kommandozeile.
z = int(sys.argv[1])
s = int(sys.argv[2])
# Zu Beginn ist das Array a leer.
a = []
# Die Zeilen des Arrays werden der Reihe nach erzeugt und an a angehängt.
for i in range(z):
  b = [0]*s                    # Die Zeile hat s Einträge (Spalten).
  a += [ b ]                   # Sie wird an a angehängt.
# Nun sind alle Plätze im Array a erzeugt.
# Sie werden durchlaufen und der passende String wird eingetragen.
for i in range(len(a)):
  for j in range(len(a[i])):
    a[i][j] = '%d,%d' % (i,j)
print(a)
```

```
algo@mm: ~/Python/VL04-2dArrays
mundhenk@home:~/Python/VL04-2dArrays$ python3 array-2d-beispiel.py 2 4
[['0,0', '0,1', '0,2', '0,3'], ['1,0', '1,1', '1,2', '1,3']]
mundhenk@home:~/Python/VL04-2dArrays$
```

Programm array-2d-beispiel.py

```
# array-2d-beispiel.py
#-------------------------------------------------------------------------------------------------------------
# Erzeuge ein 2d-Array a mit z Zeilen aus s Spalten, und schreibe an Stelle a[i][j] den String 'i,j'.
#-------------------------------------------------------------------------------------------------------------
import sys
# Lies die Zeilenzahl z und die Spaltenzahl s von der Kommandozeile.
z = int(sys.argv[1])
s = int(sys.argv[2])
# Zu Beginn ist das Array a leer.
a = []
# Die Zeilen des Arrays werden der Reihe nach erzeugt und an a angehängt.
for i in range(z):
  b = [0]*s                    # Die Zeile hat s Einträge (Spalten).
  a += [ b ]                   # Sie wird an a angehängt.
# Nun sind alle Plätze im Array a erzeugt.
# Sie werden durchlaufen und der passende String wird eingetragen.
for i in range(len(a)):
  for j in range(len(a[i])):
    a[i][j] = '%d,%d' % (i,j)
print(a)
```

```
algo@mm: ~/Python/VL04-2dArrays
mundhenk@home:~/Python/VL04-2dArrays$ python3 array-2d-beispiel.py 2 4
[['0,0', '0,1', '0,2', '0,3'], ['1,0', '1,1', '1,2', '1,3']]
mundhenk@home:~/Python/VL04-2dArrays$ python3 array-2d-beispiel.py 4 2
[['0,0', '0,1'], ['1,0', '1,1'], ['2,0', '2,1'], ['3,0', '3,1']]
mundhenk@home:~/Python/VL04-2dArrays$
```

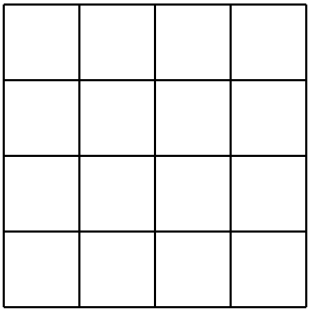
Das Modul `stdarray.py`

Zu den Modulen zum Buch gehört das Modul `stdarray.py`.

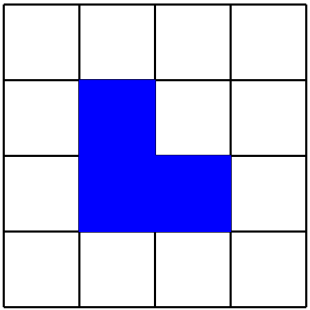
Es enthält Funktionen zum Erzeugen von Arrays.

`stdarray.create1D(n, val)`	Array der Länge `n`, jedes Element hat Wert `val`
`stdarray.create2D(n, m, val)`	`n`-mal-`m` Array, jedes Element hat Wert `val`

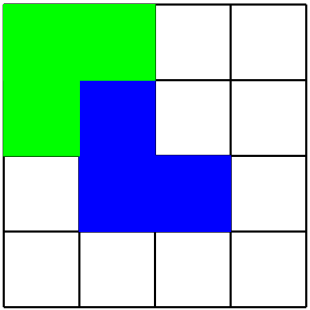
Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



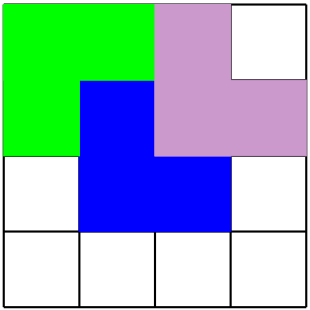
Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



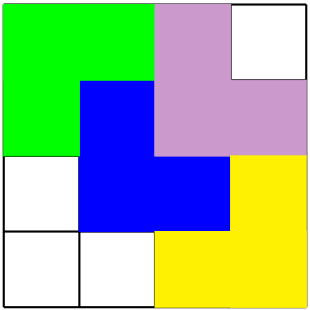
Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



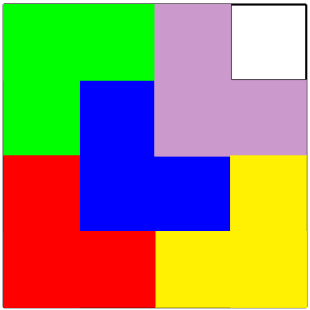
Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



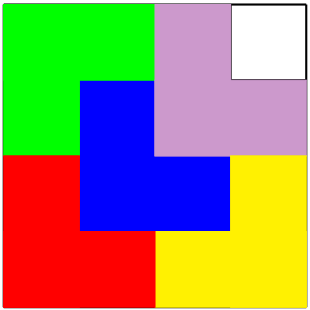
Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



Programmier-Auftrag: pflastere eine Fläche mit L-Steinen



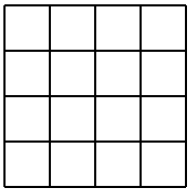
Man weiß, dass man ein Quadrat mit Seitenlänge 2^n so mit L-Steinen pflastern kann, dass nur die rechte obere Ecke frei bleibt (siehe ein paar Vorlesungen später).

Uns reicht es (heute), möglichst einfach möglichst viel zu pflastern ...

Präzisierung der Aufgabe

gegeben: zwei positive ganze Zahlen b und h

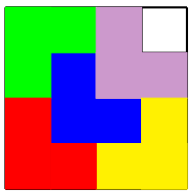
gesucht: eine möglichst lückenlose Pflasterung der Fläche mit Breite b und Höhe h
mit L-Steinen aus 3 Quadraten mit Seitenlänge 1



Präzisierung der Aufgabe

gegeben: zwei positive ganze Zahlen b und h

gesucht: eine möglichst lückenlose Pflasterung der Fläche mit Breite b und Höhe h
mit L-Steinen aus 3 Quadraten mit Seitenlänge 1

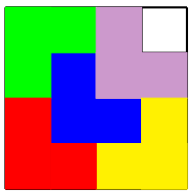


```
X X X *
X X X X
X X X X
X X X X
```

Präzisierung der Aufgabe

gegeben: zwei positive ganze Zahlen b und h

gesucht: eine möglichst lückenlose Pflasterung der Fläche mit Breite b und Höhe h
mit L-Steinen aus 3 Quadraten mit Seitenlänge 1



```
1 1 2 *
1 4 2 2
3 4 4 1
3 3 1 1
```

Die Idee

Da wir (noch) keinen Plan haben, wie man das systematisch machen kann,
pflastern wir die Fläche „zufällig“:

wiederhole ziemlich oft:
wähle zufällig eine Stelle (ein kleines Quadrat) auf der Fläche
wähle zufällig einen L-Stein
falls der L-Stein an die Stelle passt,
dann lege ihn dorthin

Idee für die Daten:
die Fläche ist ein 2-dimensionales Array mit Einträgen
`False` für „die Stelle ist nicht frei“ oder
`True` für „die Stelle ist frei“.

Die Idee

Da wir (noch) keinen Plan haben, wie man das systematisch machen kann,
pflastern wir die Fläche „zufällig“:

wiederhole ziemlich oft:
wähle zufällig eine Stelle (ein kleines Quadrat) auf der Fläche
wähle zufällig einen L-Stein
falls der L-Stein an die Stelle passt,
dann lege ihn dorthin

Idee für die Daten:
die Fläche ist ein 2-dimensionales Array mit Einträgen
`False` für „die Stelle ist nicht frei“ oder
`True` für „die Stelle ist frei“.

Ideen zur Umsetzung von „L-Stein passt an Stelle . . . “

Es gibt L-Steine in 4 Formen.

L-Stein $(1, 1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (1,1) an Stelle [2][1] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen [2][1], [2+1][1] und [2][1+1] frei sein.

Ideen zur Umsetzung von „L-Stein passt an Stelle . . . “

Es gibt L-Steine in 4 Formen.

L-Stein $(1, 1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (1,1) an Stelle [2][1] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen [2][1], [2+1][1] und [2][1+1] frei sein.

L-Stein $(1, -1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (1,-1) an Stelle [2][3] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen [2][3], [2+1][3] und [2][3-1] frei sein.

L-Stein $(1, -1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (1,-1) an Stelle [1][3] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen [1][3], [1+1][3] und [1][3-1] frei sein.

L-Stein $(1, -1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (1,-1) an Stelle [1][3] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen [1][3], [1+1][3] und [1][3-1] frei sein.

L-Stein $(-1,-1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (-1,-1) an Stelle [0][2] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen [0][2], [0-1][2] und [0][2-1] frei sein,

L-Stein $(-1,-1)$ an Stelle [z][s]:

	s-1	s	s+1
z-1	[z-1][s-1]	[z-1][s]	[z-1][s+1]
z	[z][s-1]	[z][s]	[z][s+1]
z+1	[z+1][s-1]	[z+1][s]	[z+1][s+1]

Passt L-Stein (-1,-1) an Stelle [1][3] ?

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

Dazu müssen die Stellen
[0][2], [0-1][2] und [0][2-1]
frei sein,
aber die Stelle [0-1][2] (also [-1][2]) gibt es garnicht.

Umsetzung von „L-Stein passt an Stelle . . . “

- Ein L-Stein passt an Stelle $[z][s]$, falls
  - der Stein die Form  hat,
    $z+1$ und $s+1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z+1][s]$ und $[z][s+1]$ frei sind,
  - der Stein die Form hat,
    $z+1$ und $s-1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z+1][s]$ und $[z][s-1]$ frei sind,
  - der Stein die Form hat,
    $z-1$ und $s-1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z-1][s]$ und $[z][s-1]$ frei sind, oder
  - der Stein die Form hat,
    $z-1$ und $s+1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z-1][s]$ und $[z][s+1]$ frei sind.

Umsetzung von „L-Stein passt an Stelle . . . “

- Ein L-Stein passt an Stelle $[z][s]$, falls
  - der Stein die Form  hat,
    $z+1$ und $s+1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z+1][s]$ und $[z][s+1]$ frei sind,
  - der Stein die Form  hat,
    $z+1$ und $s-1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z+1][s]$ und $[z][s-1]$ frei sind,
  - der Stein die Form  hat,
    $z-1$ und $s-1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z-1][s]$ und $[z][s-1]$ frei sind, oder
  - der Stein die Form hat,
    $z-1$ und $s+1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z-1][s]$ und $[z][s+1]$ frei sind.

Umsetzung von „L-Stein passt an Stelle . . . “

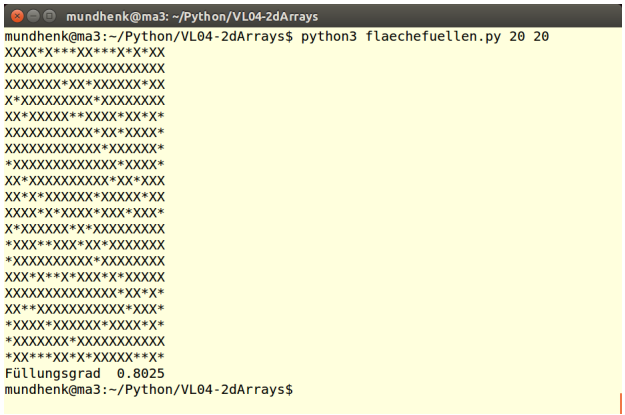
- Ein L-Stein passt an Stelle $[z][s]$, falls
  - der Stein die Form  hat,
    $z+1$ und $s+1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z+1][s]$ und $[z][s+1]$ frei sind,
  - der Stein die Form  hat,
    $z+1$ und $s-1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z+1][s]$ und $[z][s-1]$ frei sind,
  - der Stein die Form  hat,
    $z-1$ und $s-1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z-1][s]$ und $[z][s-1]$ frei sind, oder
  - der Stein die Form  hat,
    $z-1$ und $s+1$ zulässige Indizes sind,
    und die Stellen $[z][s]$, $[z-1][s]$ und $[z][s+1]$ frei sind.

Die grobe Struktur des Programmes

1. Lies Höhe `h` und Breite `b` der Fläche ein.
   Erzeuge ein 2d-Array mit `h` Zeilen und `b` Spalten.
   Alle Einträge im Array sind `True` („die Stelle ist frei“).
2. Wiederhole häufig:
   - **2.1** Wähle zufällig eine Stelle auf der Fläche.
   - **2.2** Nimm einen L-Stein mit einer zufällig gewählten Form.
   - **2.3** Wenn der L-Stein an die Stelle passt, dann lege ihn dort hin.
3. Gib die belegten Stellen der Fläche und den Füllungsgrad aus.

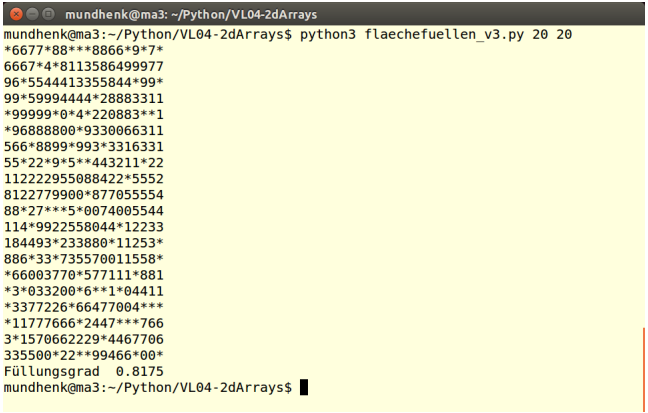
```
# flaechefuellen.py
import sys, random
# Lies die Breite und die Höhe der Fläche ein.
breite = int(sys.argv[1])
hoehe  = int(sys.argv[2])
# Erzeuge ein 2d-Array frei mit hoehe Zeilen und breite Spalten. Alle Einträge im Array sind True.
frei = [ True ] * hoehe
for i in range(len(frei)):  frei[i] = [ True ] * breite
# Wiederhole oft: wähle zufällig eine Stelle und eine Form, und versuche, einen Stein der Form an die Stelle zu legen.
for i in range((breite*hoehe)**2):
  z,s = random.randrange(hoehe), random.randrange(breite)   # die Stelle in der Fläche
  form = random.randrange(4)                                # die Form des L-Steins
  # Falls ein L-Stein der gewählten Form an die gewählte Stelle passt, dann lege ihn dort hin.
  if form==0 and z+1<hoehe and s+1<breite and frei[z][s] and frei[z+1][s] and frei[z][s+1]:
          frei[z][s] = frei[z+1][s] = frei[z][s+1] = False
  if form==1 and z+1<hoehe and 0<=s-1 and frei[z][s] and frei[z+1][s] and frei[z][s-1]:
          frei[z][s] = frei[z+1][s] = frei[z][s-1] = False
  if form==2 and 0<=z-1 and 0<=s-1 and frei[z][s] and frei[z-1][s] and frei[z][s-1]:
          frei[z][s] = frei[z-1][s] = frei[z][s-1] = False
  if form==3 and 0<=z-1 and s+1<hoehe and frei[z][s] and frei[z-1][s] and frei[z][s+1]:
          frei[z][s] = frei[z+1][s] = frei[z][s+1] = False
# Gib die belegten Stellen der Fläche und den Füllungsgrad aus.
belegteStellen = 0                # Zähler für die belegten Stellen.
for z in range(len(frei)):        # Die Fläche und ihre Belegung wird "zeilenweise" ausgegeben.
  for s in range(len(frei[z])):
    if frei[z][s]: print('*', end='')
    else:          print('X', end='') ; belegteStellen += 1
  print()
print('Füllungsgrad ', belegteStellen/(breite*hoehe))  # Der Füllungsgrad wird ausgegeben.
```

Die Ausgabe des Programmes ist nicht wirklich informativ.



Man kann beim Verlegen eines Steines aber auch dessen Stelle und Form ausgeben.
Dann kann man die Pflasterung nachvollziehen.

Das Programm lässt sich einfach zu einer informativen Ausgabe ändern.



Für jeden Stein wird eine 1stellige Zahl ausgegeben.
Das kann man in der Regel als Bauplan für seine Pflasterung nehmen.

Eine kompaktere Umsetzung von „L-Stein passt an Stelle . . . “

- Es gibt folgende 4 Formen von L-Steinen: (1,1), (1,-1), (-1,1) und (-1,-1).
- Ein L-Stein der Form (a, b) passt an Stelle $[z][s]$, falls
  - $z + a$ ein zulässiger Zeilenindex ist und
  - $s + b$ ein zulässiger Spaltenindex ist, und
  - die drei Stellen $[z][s]$, $[z + a][s]$ und $[z][s + b]$ frei sind.

Das Programm flaechefuellen_v2.py

```
import sys, random
# Lies die Breite und die Höhe der Fläche ein.
breite = int(sys.argv[1])
hoehe  = int(sys.argv[2])
# Erzeuge ein 2d-Array frei mit hoehe Zeilen und breite Spalten. Alle Einträge im Array sind True.
frei = [ True ] * hoehe
for i in range(len(frei)):
   frei[i] = [ True ] * breite
# Die Variable form enthält die 4 Formen von L-Steinen.
form = ( (1,1), (1,-1), (-1,1), (-1,-1) )
# Pflastere die Fläche zufällig.
for i in range((breite*hoehe)**2):                          # Wiederhole oft genug:
  z,s = random.randrange(hoehe), random.randrange(breite)   #   Wähle zufällig eine Stelle in der Fläche
  dz,ds = form[random.randrange(4)]                         #     und die Form des L-Steins.
  if 0<=z+dz<hoehe and 0<=s+ds<breite:                      #   Wenn der L-Stein nicht über den Rand ragt
    if frei[z][s] and frei[z+dz][s] and frei[z][s+ds]:      #     und die Stellen in der Fläche frei für ihn sind,
        frei[z][s] = frei[z+dz][s] = frei[z][s+ds] = False  #   dann lege ihn an die Stelle.
# Gib die belegten Stellen der Fläche und den Füllungsgrad aus.
belegteStellen = 0              # Zähler für die belegten Stellen.
for z in range(len(frei)):      # Die Fläche und ihre Belegung wird "zeilenweise" ausgegeben.
  for s in range(len(frei[z])):
    if frei[z][s]: print('*', end='')
    else:          print('X', end='') ; belegteStellen += 1
  print()
```

Bemerkungen zu Kurzschreibweisen im Programm

Damit das Programm auf eine Folie passt, habe ich ein paar Kurzschreibweisen verwendet.
Bei der Verwendung solcher Kurzschreibweisen muss man darauf achten, dass das Programm lesbar bleibt.

```
#   Wähle zufällig eine Stelle in der Fläche.
z,s = random.randrange(hoehe), random.randrange(breite)
```

```
#   Wähle zufällig eine Stelle in der Fläche.
z = random.randrange(hoehe)
s = random.randrange(breite)
```

```
# form = ( (1,1), (1,-1), (-1,1), (-1,-1) )
dz,ds = form[random.randrange(4)]
```

```
# form = ( (1,1), (1,-1), (-1,1), (-1,-1) )
r = random.randrange(4)
dz = form[r][0]
ds = form[r][1]
```

```
if 0 <= z+dz < hoehe and 0 <= s+ds < breite:
```

```
if 0<=z+dz and z+dz<hoehe and 0<=s+ds and s+ds<breite:
```

```
frei[z][s] = frei[z+dz][s] = frei[z][s+ds] = False
```

```
frei[z][s]    = False
frei[z+dz][s] = False
frei[z][s+ds] = False
```

Bemerkungen zu Kurzschreibweisen im Programm

Damit das Programm auf eine Folie passt, habe ich ein paar Kurzschreibweisen verwendet.
Bei der Verwendung solcher Kurzschreibweisen muss man darauf achten, dass das Programm lesbar bleibt.

```
#   Wähle zufällig eine Stelle in der Fläche.
z,s = random.randrange(hoehe), random.randrange(breite)
```

```
#   Wähle zufällig eine Stelle in der Fläche.
z = random.randrange(hoehe)
s = random.randrange(breite)
```

```
# form = ( (1,1), (1,-1), (-1,1), (-1,-1) )
dz,ds = form[random.randrange(4)]
```

```
# form = ( (1,1), (1,-1), (-1,1), (-1,-1) )
r = random.randrange(4)
dz = form[r][0]
ds = form[r][1]
```

```
if 0 <= z+dz < hoehe and 0 <= s+ds < breite:
```

```
if 0<=z+dz and z+dz<hoehe and 0<=s+ds and s+ds<breite:
```

```
if frei[z][s]: print('*', end='')
else:          print('X', end='') ; belegteStellen += 1
```

```
if frei[z][s]:
   print('*', end='')
else:
   print('X', end='')
   belegteStellen += 1
```

Zusammenfassung

Wir kennen den Datentyp `list` (Array) jetzt genauer,

können 2-dimensionale Arrays erzeugen und auf ihre Elemente zugreifen,

und haben einen Algorithmus gesehen,
der „ganz zufällig“ ein Pflasterungsproblem löst.

Es gibt bessere Algorithmen für dieses Problem,
aber keinen, bei dem man sich als Programmiererin und Programmierer
so wenig Gedanken machen muss.