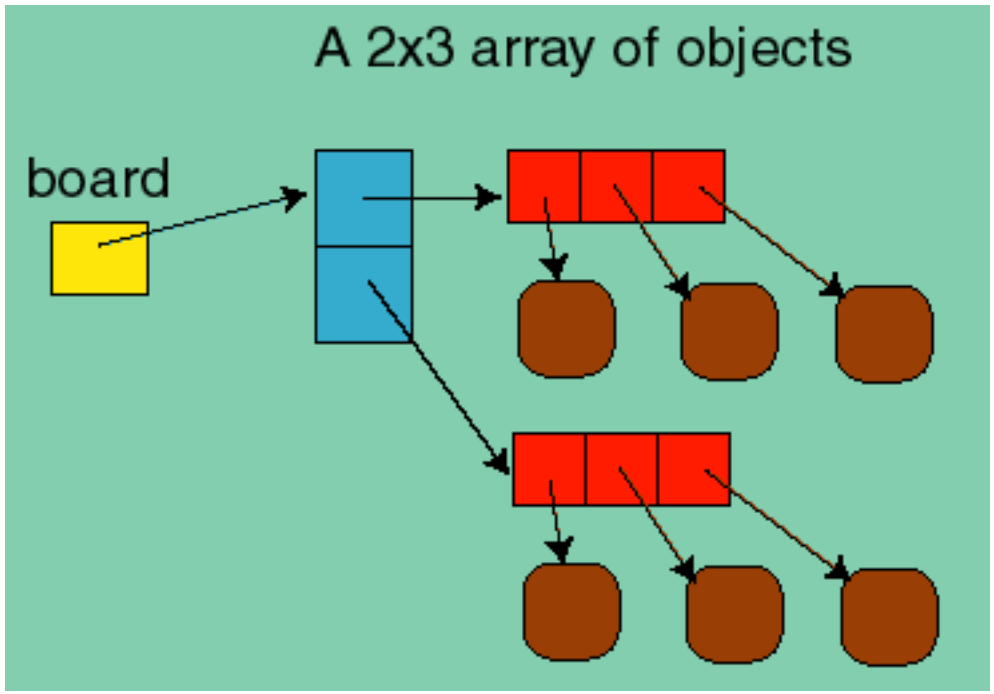
15-112
Fundamentals of Programming

Week 3 - Lecture 1:
"2-dimensional" lists



May 31, 2016

"2d lists"

A list can contain any type of object.

```
a = [1, "hello", False]
```

Can also contain lists.

```
a = [[1, 3, 5], [6], [1, 5]]          # A list of lists

print(len(a))        3
```

a[0] is a reference to the first list [1, 3, 5]

a[1] is a reference to the second list [6]

a[2] is a reference to the third list [1, 5]

a[0][0] is a reference to the first element of the first list [1, 3, 5]

a[2][1] is a reference to the second element of the third list [1, 5]

Example: Print all the elements

```
a = [[1, 3, 5], [6], [1, 5]]
```

```
a = [ [1, 3, 5],
      [6],
      [1, 5]
    ]
```

Looping through the elements one by one.

```
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j])
```

```
1
3
5
6
1
5
```

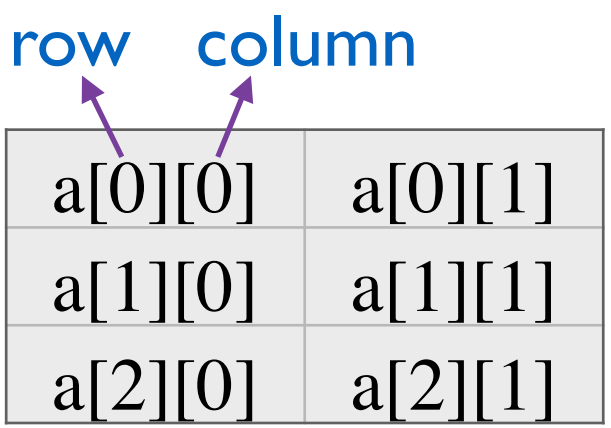
rectangular "2d list"

Most "2d lists" we deal with will have same length sublists.

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

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

Really like a table (or matrix)



2d list examples

A chess board: 8 lists of length 8 each (or 8 by 8 table)

Each entry either contains a chess piece or is empty.

An image: a 2d list of points/pixels

Each entry contains the color of the point.

A database: e.g. a list of users and various information about the users

Cool. Seems easy enough. Can we go home?

Unfortunately, no. 

Tricky thing about 2d lists

1d list: references to **immutable** objects.

Aliases of elements not a problem.

2d list: references to **mutable** objects.

We must be careful about aliases of elements !!

"Weird" Example 1

```
a = [1, 2, 3]
b = copy.copy(a)
b[0] = 0
print(a)        [1, 2, 3]
print(b)        [0, 2, 3]
```

```
a = [[1, 2, 3], [4, 5, 6]]
b = copy.copy(a)
b[0][0] = 0
print(a)        [ [0, 2, 3], [4, 5, 6] ]
print(b)        [ [0, 2, 3], [4, 5, 6] ]
```

“Weird” Example 2

```
a = [ [0]*2 ]*3
print(a)
```

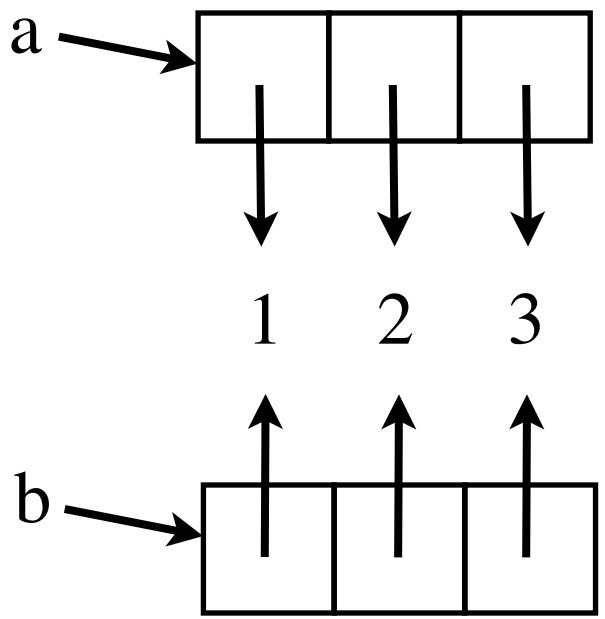
[[0, 0], [0, 0], [0, 0]]

```
a[0][0] = 9

print(a)
```

[[9, 0], [9, 0], [9, 0]]

Understanding Example 1

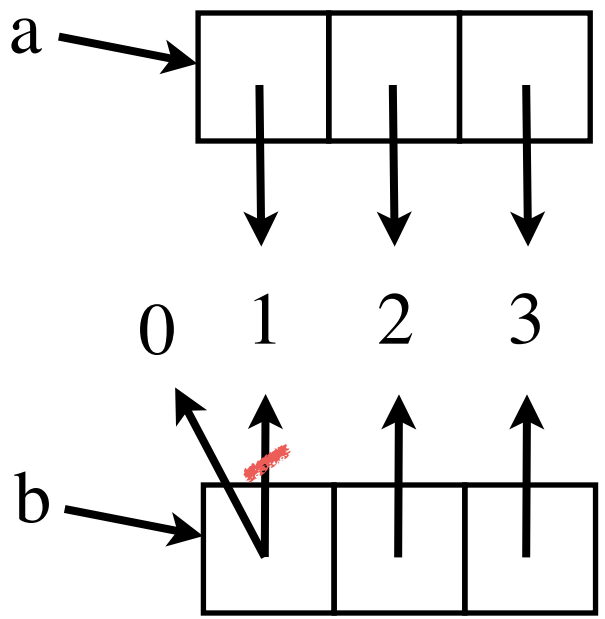
```
a = [1, 2, 3]

b = copy.copy(a)

b[0] = 0

print(a[0])

print(b[0])
```



Making a copy of the references.

Understanding Example 1

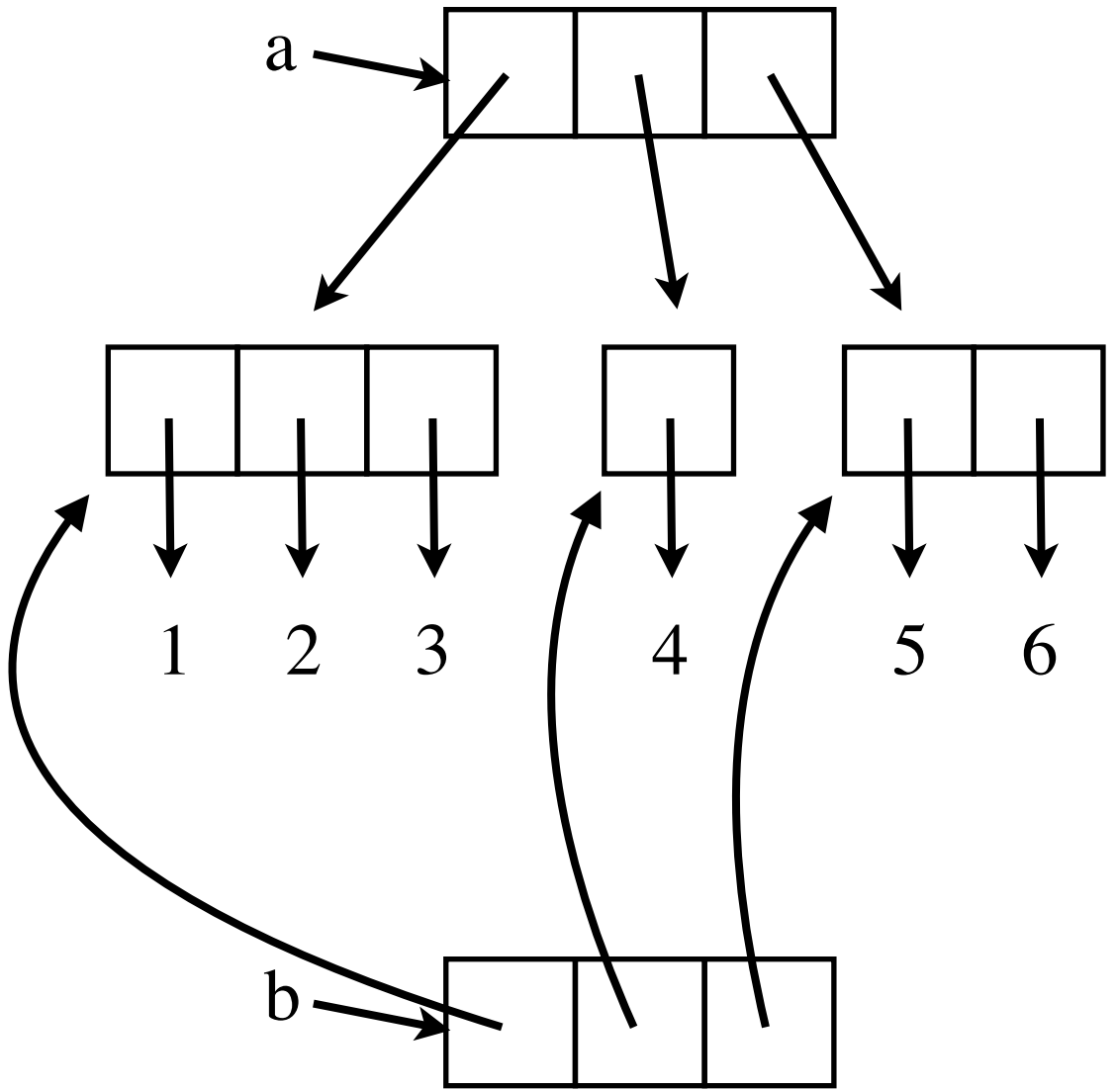
```
a = [1, 2, 3]

b = copy.copy(a)

b[0] = 0

print(a[0])

print(b[0])
```



Making a copy of the references.

Understanding Example 1

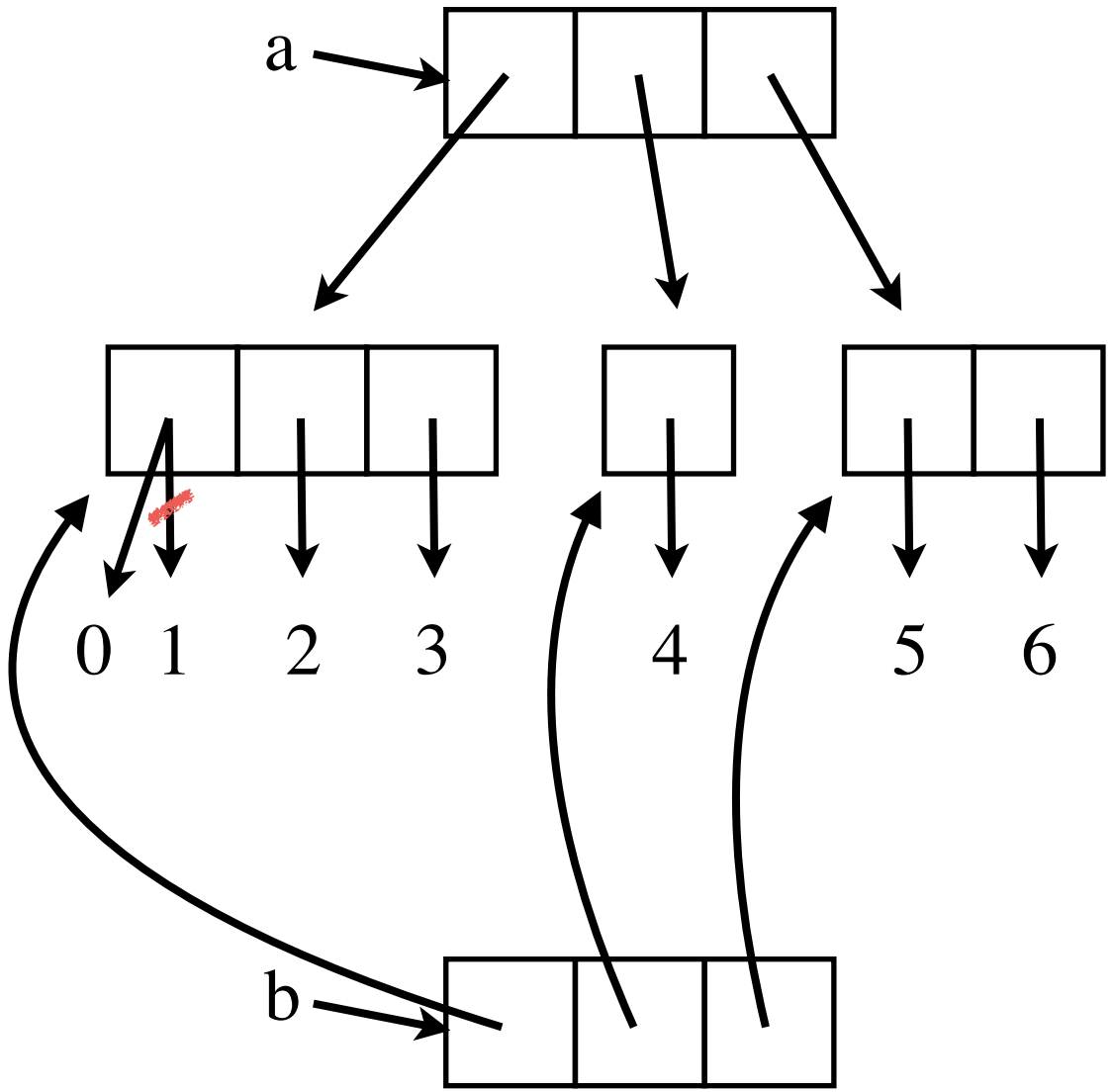
```
a = [[1, 2, 3], [4], [5, 6]]

b = copy.copy(a)

b[0][0] = 0

print(a[0][0])

print(b[0][0])
```



Understanding Example 1

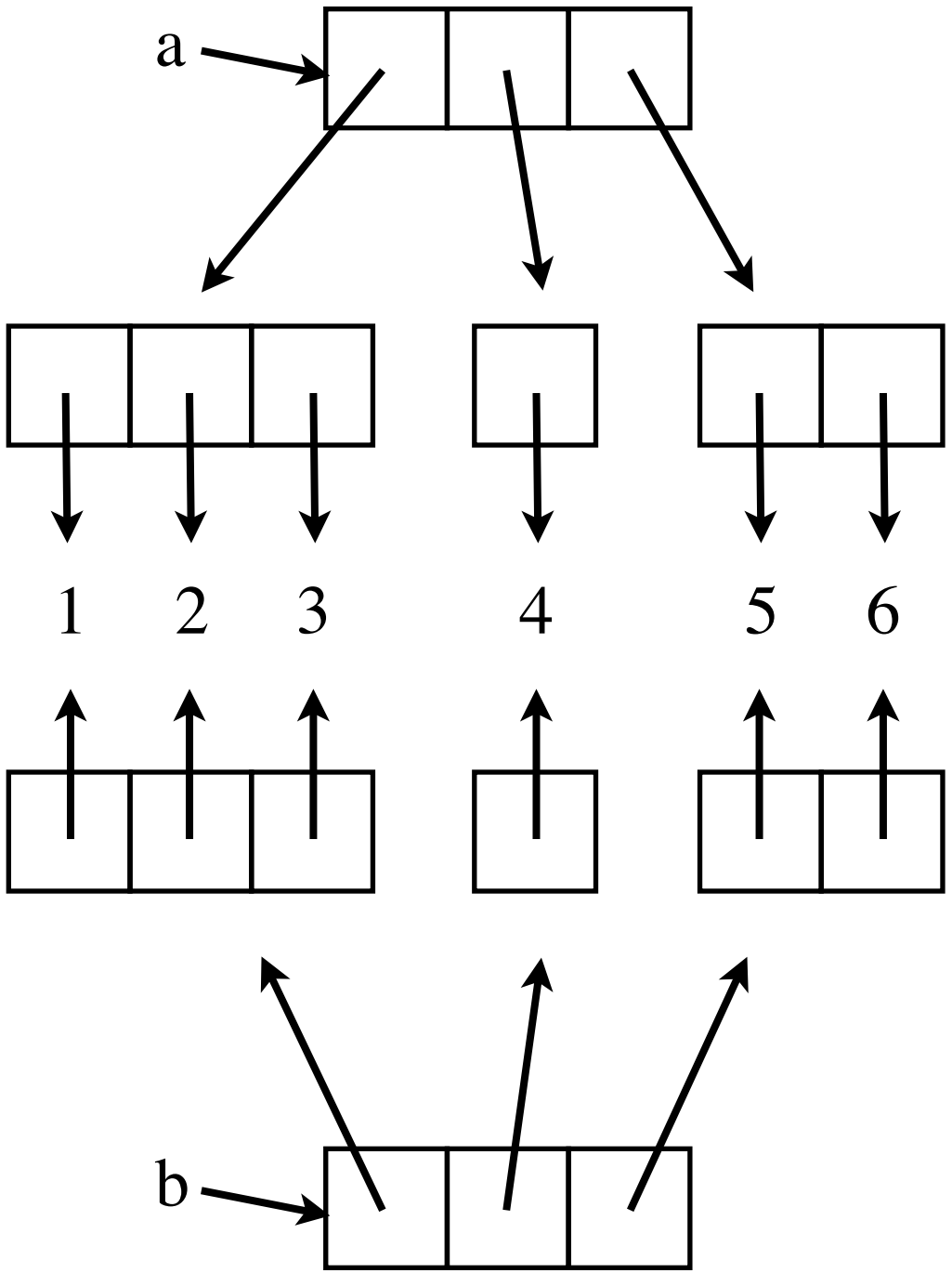
```
a = [[1, 2, 3], [4], [5, 6]]

b = copy.copy(a)

b[0][0] = 0

print(a[0][0])

print(b[0][0])
```

Shallow copy



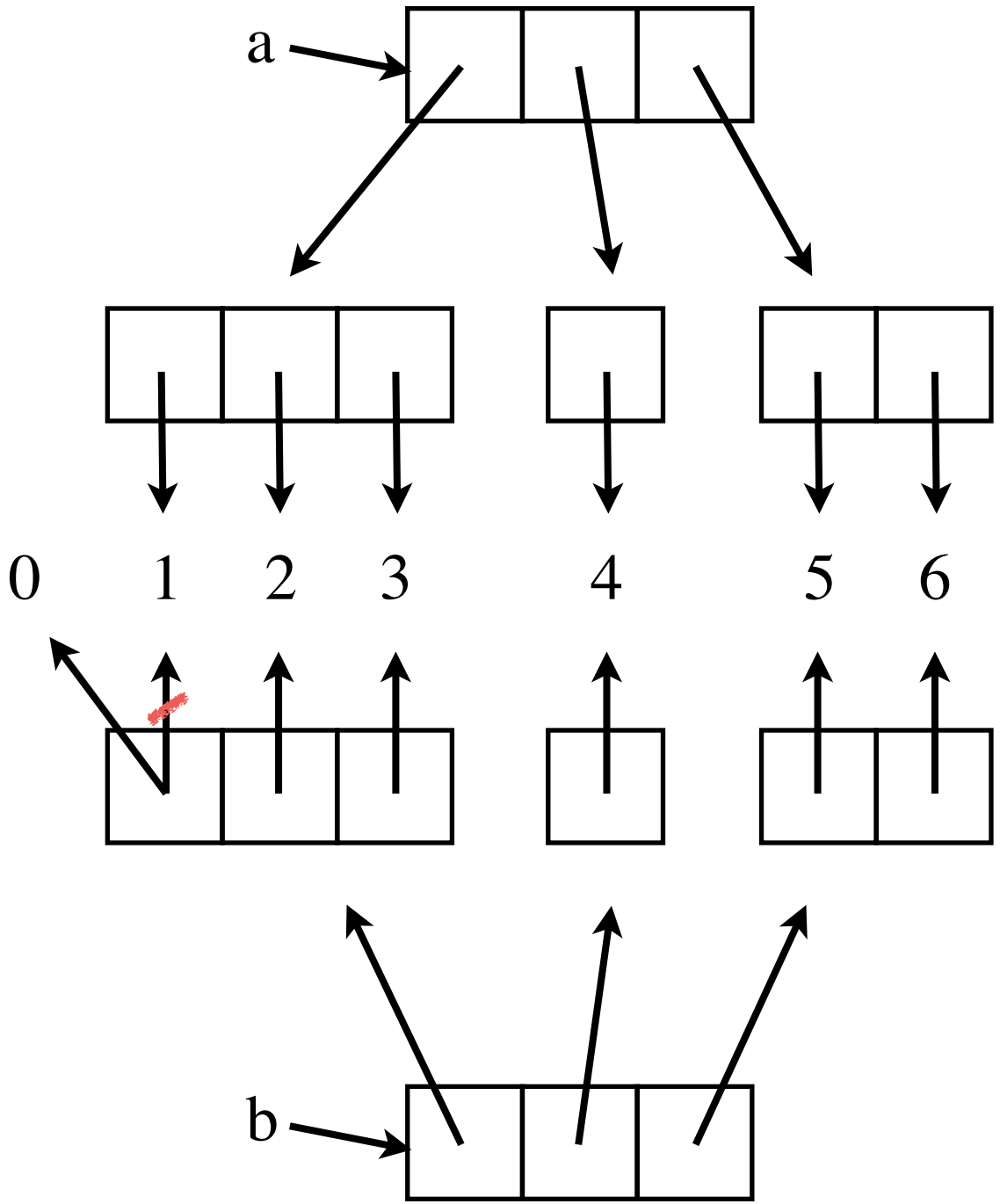
Understanding Example 1

```
a = [[1, 2, 3], [4], [5, 6]]

b = copy.deepcopy(a)

b[0][0] = 0

print(a[0][0])

print(b[0][0])
```



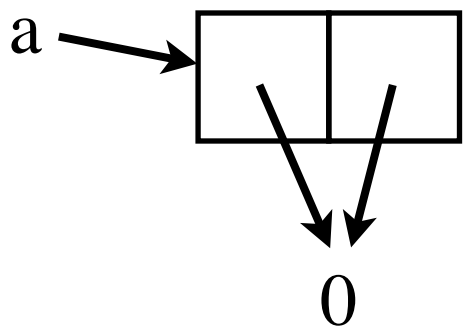
Understanding Example 1

```
a = [[1, 2, 3], [4], [5, 6]]

b = copy.deepcopy(a)

b[0][0] = 0

print(a[0][0])

print(b[0][0])
```



Understanding Example 2

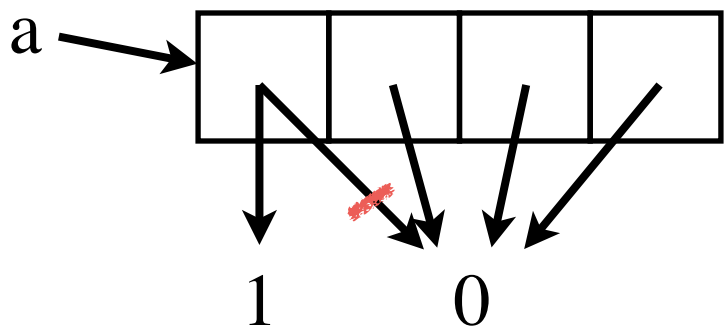
```
a = [0]*2
```



Understanding Example 2

a = [0]*4

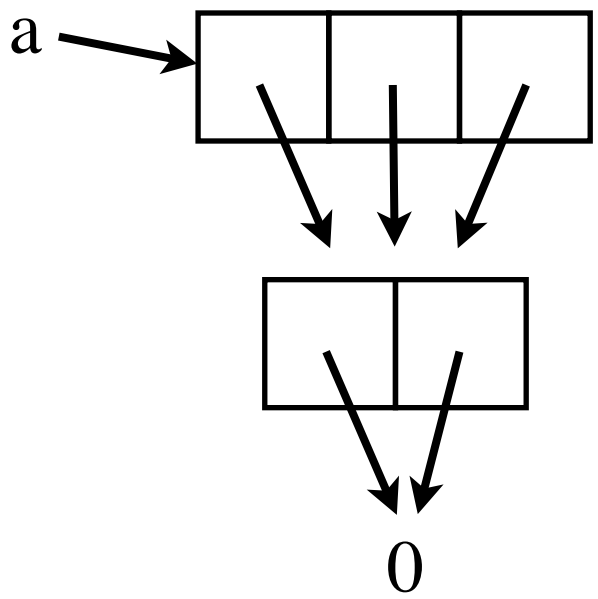
a[0] = 1



Understanding Example 2

 # Create a 3 by 2 list

a = [[0]*2]*3



Understanding Example 2

 # Create a 3 by 2 list

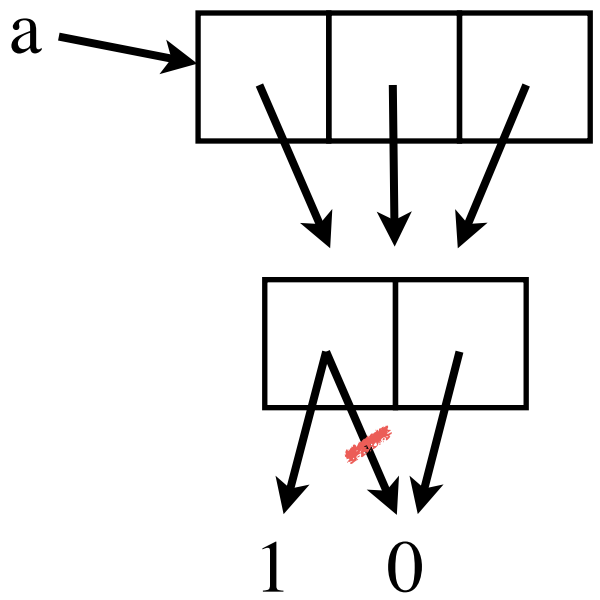
a = [[0]*2]*3

[[0, 0], [0, 0], [0, 0]]

a[0][0] = 1

print(a)

[[1, 0], [1, 0], [1, 0]]



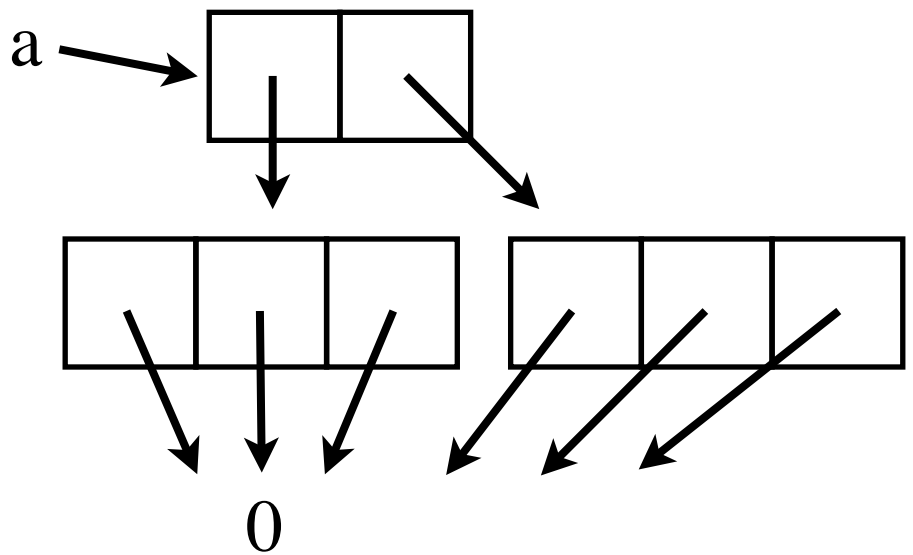
a[0], a[1], and a[2] are aliases !

* makes a shallow copy !

Creating a *rows* by *cols* 2d list

```
rows = 2
cols = 3
a = [ ]
for row in range(rows):
    a += [[0]*cols]
```

```
a += [[0, 0, 0]]
a += [[0, 0, 0]]
```



Creating a *rows* by *cols* 2d list

Define a function for this task.

```
def make2dList(rows, cols):
    a = []
    for row in range(rows):
        a += [[0]*cols]
    return a
```

One more important thing

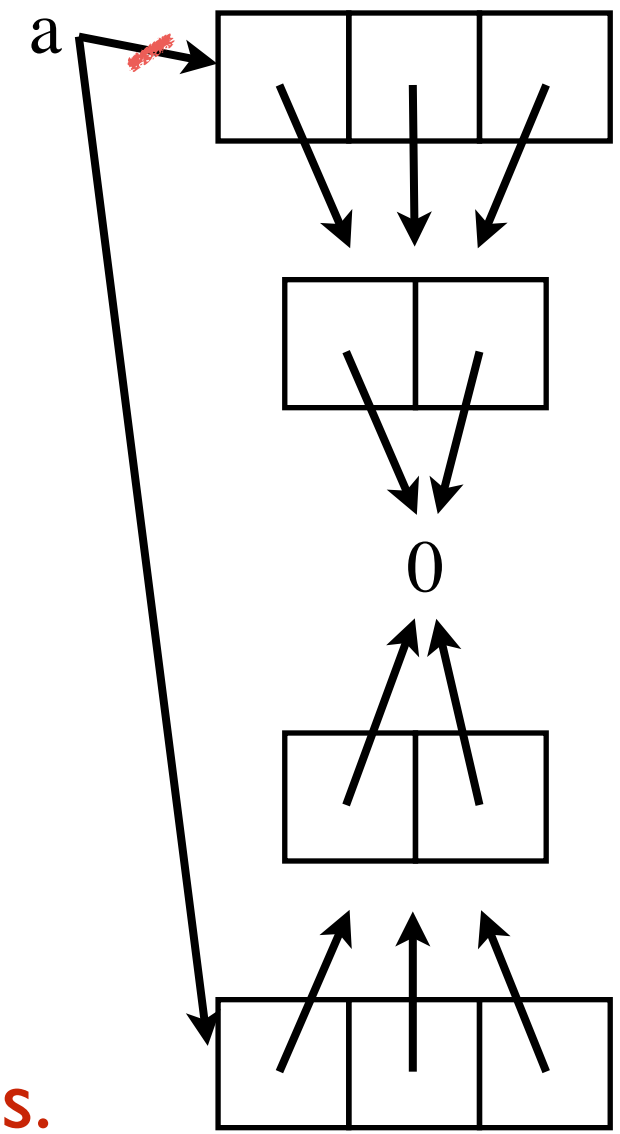
```
# Create a 3 by 2 list
a = [ [0]*2 ]*3
```

Trying to break aliasing with deepcopy:

```
a = copy.deepcopy(a)
```

deepcopy preserves alias structure !!

see myDeepCopy in the notes.



Rules

Use * only on the first level (with immutable elements)

- creates aliases

Never use copy with 2d lists.

- creates aliases
- ok to use with 1d lists since elements are immutable.

Remember: deepcopy does not break alias structure within the list.

3d Lists

2d list:

```
a = [ [1, 3, 5],
      [6],
      [1, 5]
    ]
```

Printing elements of 2d lists:

```
for i in range(len(a)):
    for j in range(len(a[i])):
        print("a[%d][%d] = %d" % (i, j, a[i][j]))
```

```
a[0][0] = 1
a[0][1] = 3
a[0][2] = 5
a[1][0] = 6
a[2][0] = 1
a[2][1] = 5
```

3d Lists

```
a1 = [ [ 1, 2 ],
       [ 3, 4 ] ]

a2 = [ [ 5, 6, 7 ],
       [ 8, 9 ]    ]

a3 = [ [ 10 ] ]
```

3d list:

```
a = [ a1, a2, a3 ]
```

4d list:

```
a = [ a, a ]
```

3d Lists

```
a = [   [ [ 1, 2 ],
          [ 3, 4 ] ],
                    [ [ 5, 6, 7 ],
                      [ 8, 9 ] ],
                                  [ [ 10 ] ]
    ]
```

Printing elements of 3d lists:

```
for i in range(len(a)):
    for j in range(len(a[i])):
        for k in range(len(a[i][j])):
            print("a[%d][%d][%d] = %d" % (i, j, k, a[i][j][k]))
```

Example Problem: Word Search

HEATINGOIL
KEROSENE
AGAMATE
KEROULTRA
TANKER
DELIVERY
RUNOUT
TATEOIL
FILLUP
LITRES
DRIVER
FREEDOM
ACCOUNT
ORDER
CENTRALHEATING
CUSTOMERSERVICE
BOILER
PUMP
GASOIL
DIESEL
ADBLUE
ANTIWAX
LUBRICANTS
PARAFFIN
ENGINEOIL
GREASE
BUNDEDTANK
APOLLO
MONITOR
SALESTEAM

Example Problem: Word Search

```
def testWordSearch():
    board = [ [ 'd', 'o', 'g' ],
              [ 't', 'a', 'c' ],
              [ 'o', 'a', 't' ],
              [ 'u', 'r', 'k' ],
            ]
    print(wordSearch(board, "dog"))     # ('dog', (0, 0), 'right')
    print(wordSearch(board, "cat"))     # ('cat', (1, 2), 'left')
    print(wordSearch(board, "tad"))     # ('tad', (2, 2), 'up-left')
    print(wordSearch(board, "cow"))     # None
```

Example Problem: Word Search

```
def wordSearch(board, word):
    # ...
```

Algorithm: wordSearch(board, word)

- go through each **cell** of the board one by one:
    - check if word appears starting at that **cell**

Example Problem: Word Search

```
def wordSearch(board, word):
    # ...
```

Algorithm: wordSearch(board, word)

- go through each cell of the board one by one:
  - check if word appears starting at that cell

needs to be broken down further

Example Problem: Word Search

```
def wordSearchFromCell(board, word, startRow, startCol):
    # ...
```

Algorithm: wordSearchFromCell(board, word, startRow, startCol)

- go through each **direction** one by one:
    - check if word appears in that **direction** starting at the given cell

Example Problem: Word Search

```
def wordSearchFromCell(board, word, startRow, startCol):
    # ...
```

Algorithm: wordSearchFromCell(board, word, startRow, startCol)

- go through each **direction** one by one:
  - check if word appears in that **direction** starting at the given cell

needs to be broken down further

it is important how you represent **direction**.

let's see an elegant way of doing it…