w14-Lab2

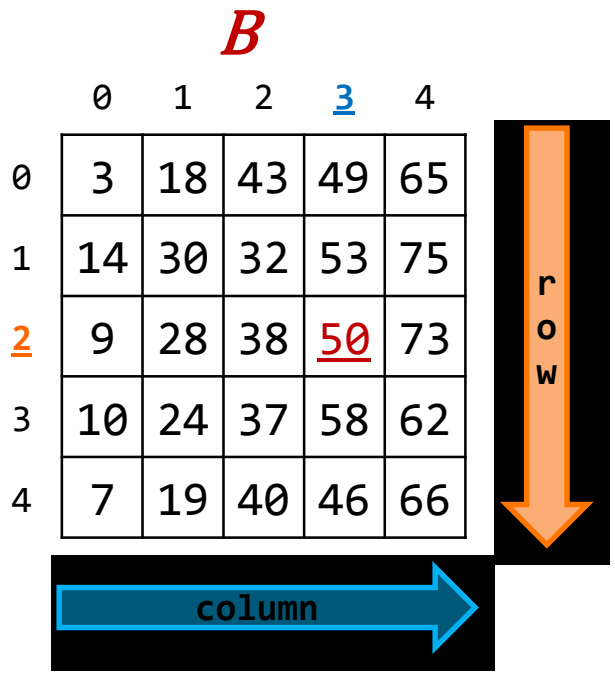
n-Dimensional Lists

for 204111

Kittipitch Kuptavanich

Two-Dimensional Lists

- ในหลาย ๆ กรณี การแทนข้อมูลที่ใช้ในรูปแบบตาราง (Matrix, 2-dimensional Array หรือ 2-dimensional List) ทำให้ทำงานได้มีประสิทธิภาพมากขึ้น
- เราสามารถเข้าถึงข้อมูลในแต่ละช่อง (Cell) ได้ด้วยการใช้เครื่องหมาย Subscript ระบุ Row และ Column ในรูป



[row][column]

```
>>> B[2][3]
50
```

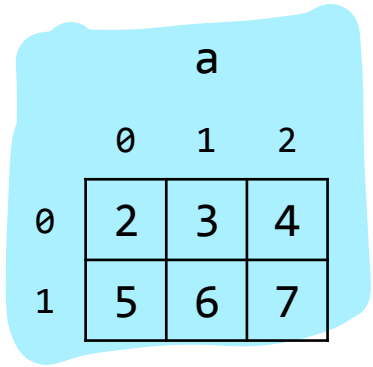
Creating 2D Lists

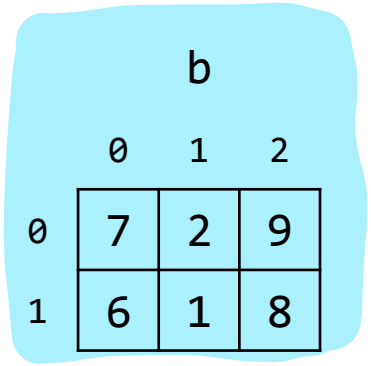
Static Allocation

- คือการสร้าง List แบบกำหนดค่า

```
02 a = [[2, 3, 4],
03      [5, 6, 7]]
04
05 b = [[7, 2, 9], [6, 1, 8]]
```

```
>>> print(a)
[[2, 3, 4], [5, 6, 7]]

>>> print(b)
[[7, 2, 9], [6, 1, 8]]
```





http://www.cs.cmu.edu/~./15110/lectures/lec15-Arrays.pdf

Creating 2D Lists [2]

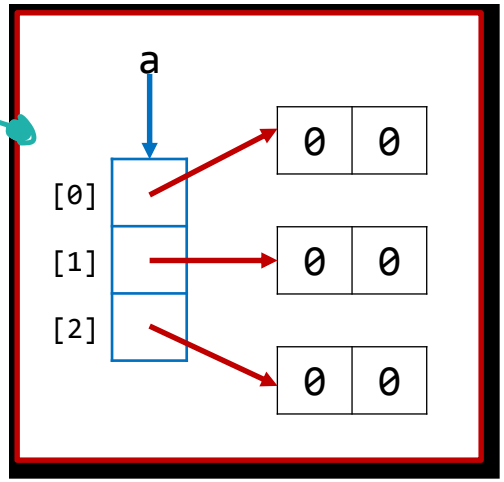
Dynamic Allocation: ต้องการสร้าง **Zero Matrix ขนาด 3 × 2**

- **Right**: ใช้ **Loop** เพื่อเพิ่มทีละแถว

ลองสอบ

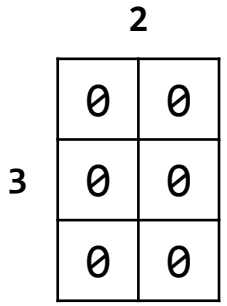
```
>>> rows = 3
>>> cols = 2

>>> a=[]
>>> for row in range(rows):
...     a += [[0] * cols]
>>> a
[[0, 0], [0, 0], [0, 0]]

>>> a[0][0] = 42
>>> a
[[42, 0], [0, 0], [0, 0]]
```

```
01 def make_2d_list(rows, cols):
02     a = []
03     for row in range(rows):
04         a += [[0] * cols]
05     return a
```



- สร้างเป็นฟังก์ชัน `make_2d_list()`

Creating 2D Lists [3]



Dynamic Allocation

- **Wrong**: การใช้ * เป็นการสร้าง **Shallow Copy**

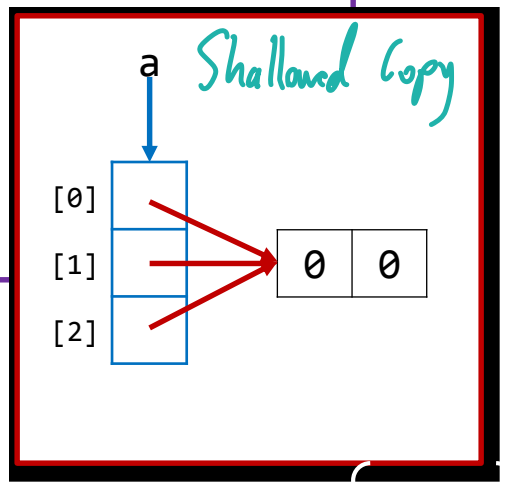
```
>>> rows = 3
>>> cols = 2

>>> a = [[0] * cols] * rows
>>> a
[[0, 0], [0, 0], [0, 0]]

>>> a[0][0] = 42
>>> a
[[42, 0], [42, 0], [42, 0]]
```

Atomic ดังนั้นสามารถทำ Shallow Copy

= [0, 0]

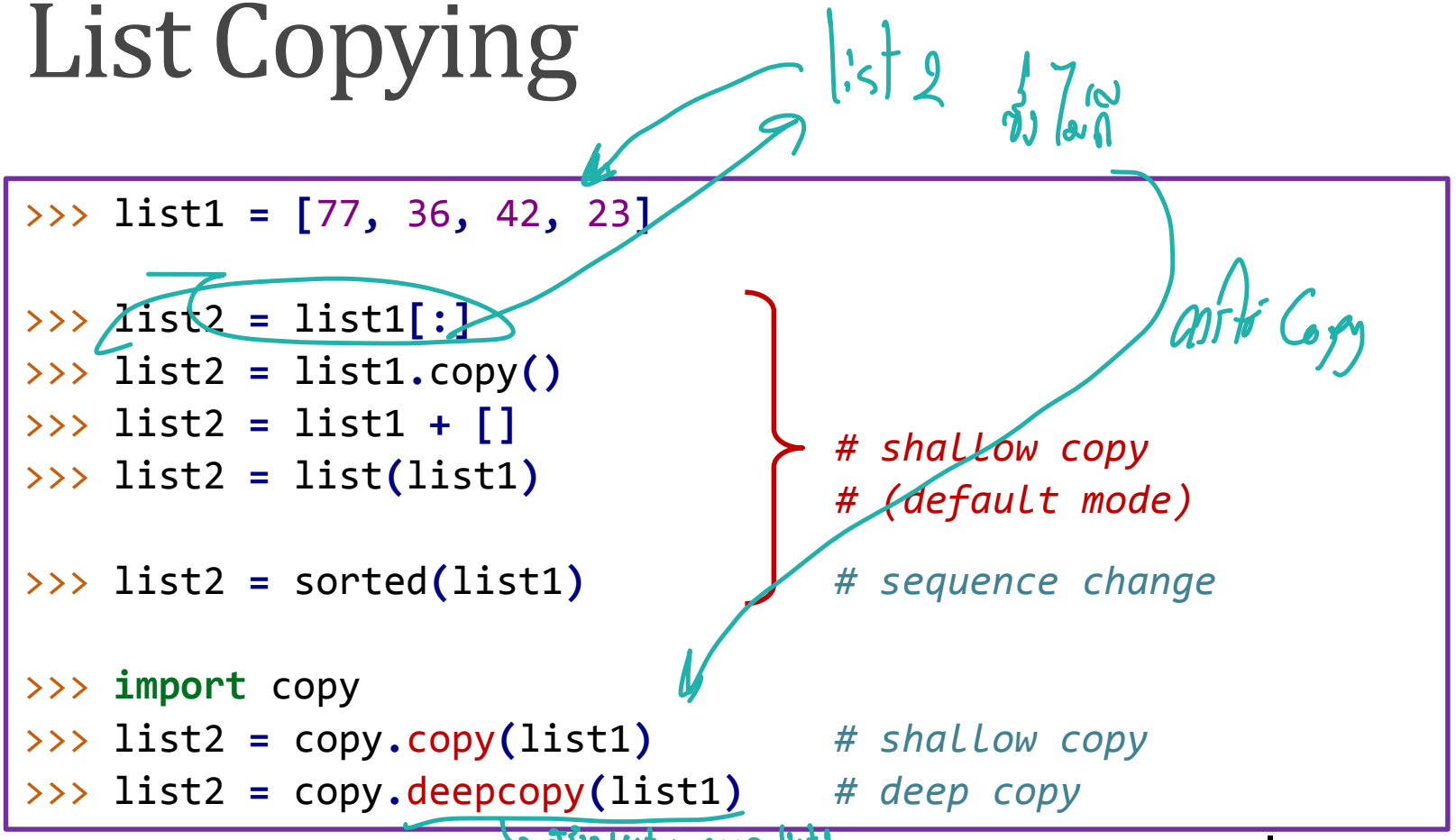
→ไม่ได้ทำใน for loop

Shallow copying
1D List: WRONG



https://docs.python.org/3/library/copy.html

List Copying

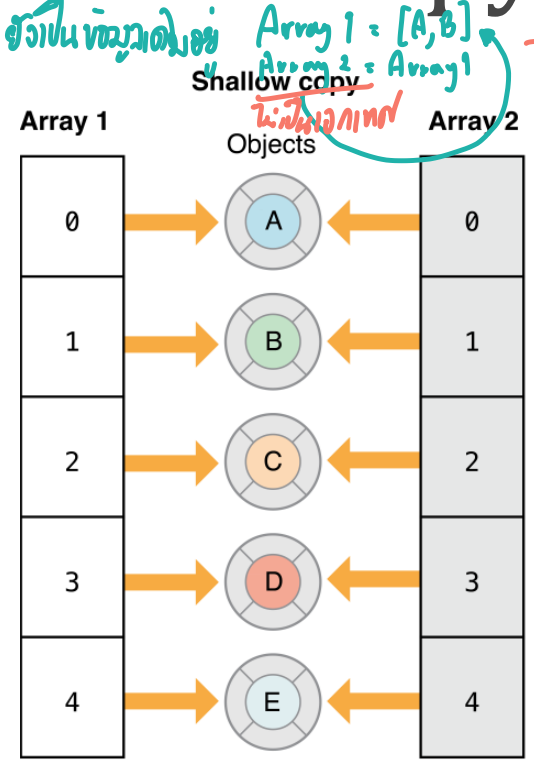


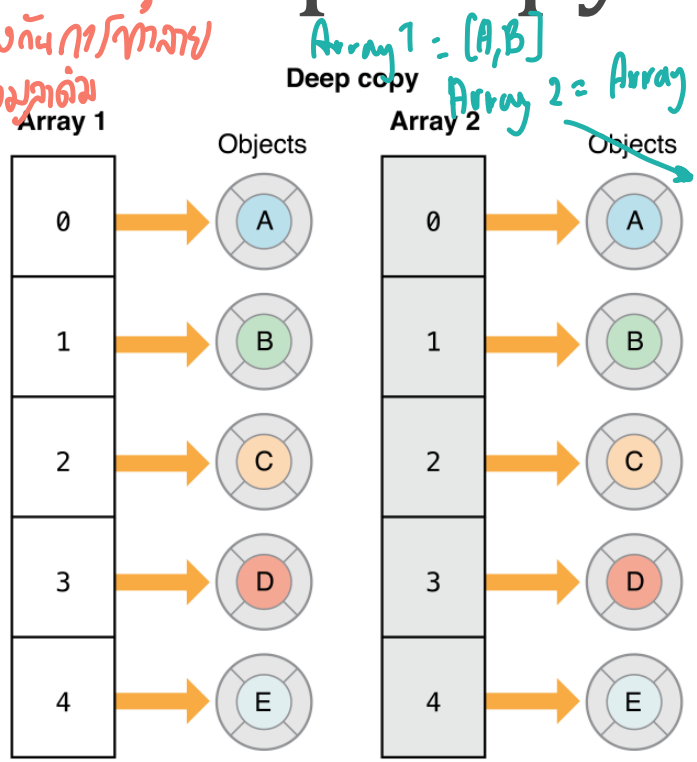
```
>>> list1 = [77, 36, 42, 23]

>>> list2 = list1[:]
>>> list2 = list1.copy()
>>> list2 = list1 + []
>>> list2 = list(list1)

>>> list2 = sorted(list1)

>>> import copy
>>> list2 = copy.copy(list1)
>>> list2 = copy.deepcopy(list1)
```

```
# shallow copy
# (default mode)

# sequence change

# shallow copy
# deep copy
```


- **Deep Copy และ Shallow Copy แตกต่างกันเฉพาะในกรณีที่ Element ของ List เป็น Compound Object เช่น List (หรือ Class)**
  - **Deep Copy จะสร้าง Element ใหม่ แล้วสร้าง Reference ชี้ไป**
  - **Shallow Copy แค่สร้าง Reference ใหม่ แล้วชี้ไปที่ Element เดิม**

Shallow Copy vs Deep Copy





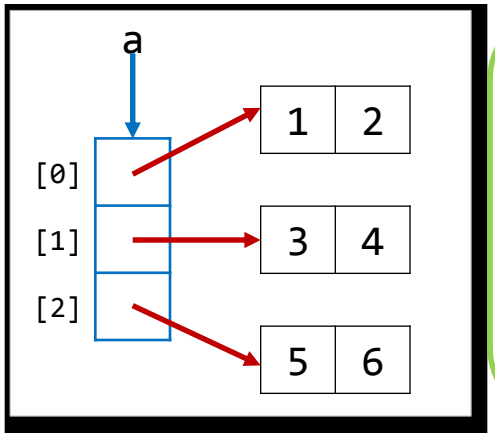

```
>>> a = [1, [2, 3]]
>>> b = copy.copy(a)
>>> b[0] is a[0]      #atomic
True
>>> b[1] is a[1]
True
```

```
>>> a = [1, [2, 3]]
>>> b = copy.deepcopy(a)
>>> b[0] is a[0]      #atomic
True
>>> b[1] is a[1]
False
```

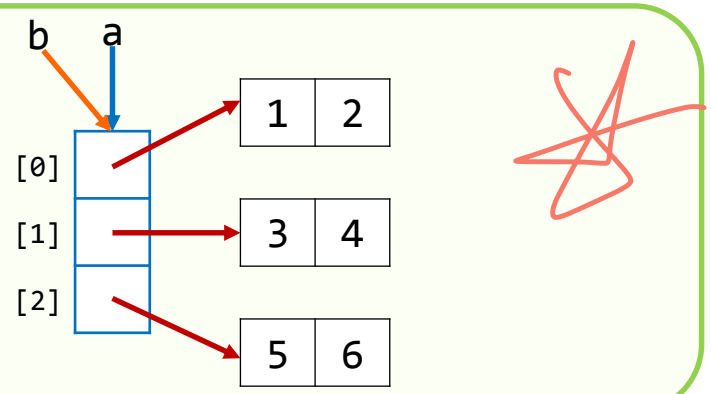
https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Collections/Articles/Copying.html

List Aliasing and Copying

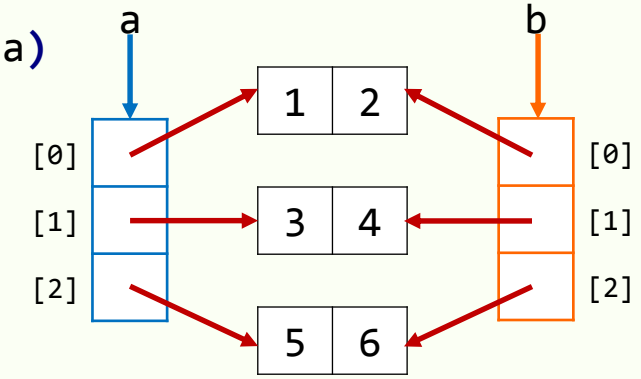
```
>>> a =  [[1, 2],
          [3, 4],
          [5, 6]]
```



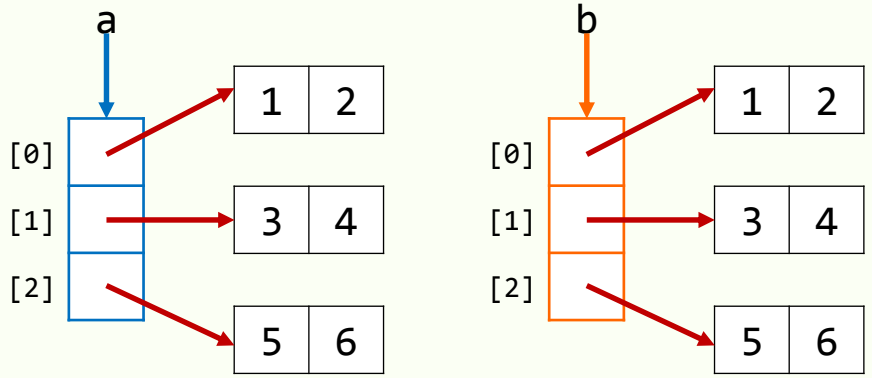

```
>>> b = a
```




```
>>> b = copy.copy(a)
```




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



Looping over 2D Lists

```
01 # Create an "arbitrary" 2d List
02 a = [[2, 3, 5], [1, 4, 7]]
03 print("Before: a =\n", a)
04
05 # Now find its dimensions
06 rows = len(a)
07 cols = len(a[0])
08
09 # And now loop over every element
10 # and add one to each
11 for row in range(rows):
12     for col in range(cols):
13         a[row][col] += 1
14
15 # Finally, print the results
16 print("\nAfter:  a =\n", a)
```

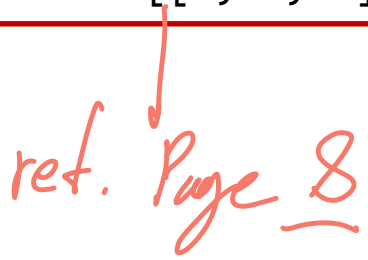
```
>>>
Before: a =
[[2, 3, 5], [1, 4, 7]]

After:  a =
[[3, 4, 6], [2, 5, 8]]
```

Copying 2D List

list_copy.py

```
01 import copy
02
03 # Create a 2d list
04 a = [[1, 2, 3], [4, 5, 6]]
05
06 # Try to copy it
07 b = copy.copy(a)
08 c = copy.deepcopy(a)
09
10 print("Before")
11 print("   a =", a)
12 print("   b =", b)
13 print("   c =", c)
14
15 a[0][0] = 9
16 print("\nAfter a[0][0] = 9")
17 print("   a =", a)
18 print("   b =", b)
19 print("   c =", c)
```

```
$ python list_copy.py
Before
   a = [[1, 2, 3], [4, 5, 6]]
   b = [[1, 2, 3], [4, 5, 6]]
   c = [[1, 2, 3], [4, 5, 6]]

After a[0][0] = 9
   a = [[9, 2, 3], [4, 5, 6]]
   b = [[9, 2, 3], [4, 5, 6]]
   c = [[1, 2, 3], [4, 5, 6]]
```

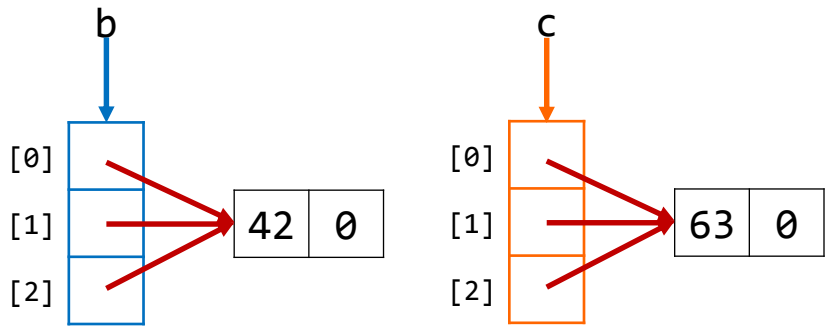


Copying 2D List [2]

```
01 import copy
02
03 a = [[0] * 2] * 3
04 a[0][0] = 42
05 print("a = \n", a)
06
07 b = [[0] * 2] * 3
08 c = copy.deepcopy(b)
09 b[0][0] = 42
10 c[0][0] = 63
11 print("\nb = \n", b)
12 print("\nb = \n", c)
```

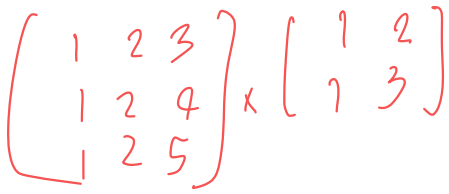
ref. Page 5

```
a =
 [[42, 0], [42, 0], [42, 0]]

b =
 [[42, 0], [42, 0], [42, 0]]

c =
 [[63, 0], [63, 0], [63, 0]]
```



- ในกรณีที่ใช้ Deep Copy ถ้า Original เป็น Shallow Copy ก็จะได้ผลลัพธ์เหมือน Original

Accessing Rows and Columns



- ต้องการเข้าถึงข้อมูลทั้ง **Row** ในคราวเดียว

```
01 # Accessing a whole row
02 # alias (not a copy!); cheap (no new list created)
03 a = [[1, 2, 3], [4, 5, 6]]
04 row = 1
05 rowList = a[row]
06 print(rowList)                    # [4, 5, 6]
```

- ต้องการเข้าถึงข้อมูลทั้ง **Column** ในคราวเดียว

```
09 # Accessing a whole column
10 # copy (not an alias!); expensive (new list created)
11 a = [[1, 2, 3], [4, 5, 6]]
12 col = 1
13 colList = []
14 for i in range(len(a)):
15     colList += [a[i][col]]
16 print(colList)                    # [2, 5]
```

Non-Rectangular 2D Lists

- **List 2 มิติไม่จำเป็นต้องมีลักษณะเป็นสี่เหลี่ยมผืนผ้า**
  - **แต่ละแถวไม่จำเป็นต้องมีจำนวน Element เท่ากัน**

```
01 # 2d lists do not have to be rectangular
02 a = [[1, 2, 3],
03      [4, 5],
04      [6],
05      [7, 8, 9, 10]]
06
07 rows = len(a)
08 for row in range(rows):
09     cols = len(a[row])
10     print("Row", row, "has", cols, "columns: ", end="")
11     for col in range(cols):
12         print(a[row][col], end=" ")
13     print()
```

2D ข้างในมี 2

```
>>>
Row 0 has 3 columns: 1 2 3
Row 1 has 2 columns: 4 5
Row 2 has 1 columns: 6
Row 3 has 4 columns: 7 8 9 10
```

เช็คจำนวน Column ทุกครั้งเมื่อขึ้น Row ใหม่

Nested Lists and `map()`

- เราสามารถใช้ฟังก์ชัน `map()` (รวมถึง `filter()` และ `reduce()`) กับ **nested list** ได้ เช่นเดียวกันกับ **1d list** ปกติ

```
>>> scores = [
    ['John', 50, 45],
    ['Jane', 48, 36],
    ['Dave', 39, 49],
]
#  could as well be a list of tuples, nested tuples, etc.

>>> avg = map(lambda x: [x[0], sum(x[1:])/(len(x)-1)], scores)
>>> print(list(avg))
[['John', 47.5], ['Jane', 42.0], ['Dave', 44.0]]
```

Flatten Lists Using reduce()

```
>>> a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> from functools import reduce
>>> reduce(lambda x, y: x + y, a)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

รวม list * ลดโครงสร้างลง 2 เหลือ 1 อัน

3D Lists

ขยายจะไม่ออก

- โดยแท้ที่จริงแล้ว List 2 มิติ ใน Python คือ Nested List (List ซ้อน List) ดังนั้น เราสามารถสร้าง List 3 มิติ หรือ 4 มิติ และ อื่น ๆ ได้อย่างไม่จำกัดรูปร่างและขนาด

```
02 a = [[[1, 2],
03       [3, 4]],
04      [[5, 6, 7],
05       [8, 9]],
06      [[10]]]
07
08 for i in range(len(a)):
09     for j in range(len(a[i])):
10         for k in range(len(a[i][j])):
11             print("a[%d][%d][%d] = %d" % (i, j, k, a[i][j][k]))
```

```
>>>
a[0][0][0] = 1
a[0][0][1] = 2
a[0][1][0] = 3
a[0][1][1] = 4
a[1][0][0] = 5
a[1][0][1] = 6
a[1][0][2] = 7
a[1][1][0] = 8
a[1][1][1] = 9
a[2][0][0] = 10
```

unzip Operation with 2D Lists

ไม่มีคำสั่ง unzip ต้องใช้วิธีนี้

```
>>> a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>>
>>> # notice the transpose effect
>>> b = list(zip(*a))
>>> b
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>>
>>> # another round of transpose
>>> c = list(zip(*b))
>>> c
[(1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

LIST COMPREHENSION

List Comprehensions

สร้าง list ด้วยวิธีใหม่ บางอย่าง

- **List Comprehensions เป็น Concept หนึ่งใน Python ในการสร้าง List ซึ่งโดยมากมักเป็นการสร้าง List จาก Element ของ List อื่น ๆ (หรือ Iterable Data Type ชนิดอื่น ๆ)**
- **พิจารณา การสร้าง List**

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...

>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- **เราสามารถสร้าง List ที่เหมือนกันโดยใช้**

```
>>> squares = [x ** 2 for x in range(10)]
```

List Comprehensions [2]

- **List Comprehension ประกอบด้วย Square Brackets [ ] ที่มี Expression `for` ข้างใน โดยสามารถมีมากกว่า 1 `for` Expression หรือมี `if` Expression ได้**
- **ผลลัพธ์ที่ได้จะเป็น List ที่เกิดจากการ evaluate ตัว Expression ภายใน Brackets [ ]**

```
>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

- **Expression ด้านบนสร้าง List ของ Tuple ที่ประกอบด้วย Element จาก 2 List จับคู่กัน**
  - **เว้นกรณีที่ Element จาก 2 List เท่ากัน**

List Comprehensions [3]

```
[(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]

# Expression นี้มีการทำงานเหมือน:

>>> combs = []
>>> for x in [1, 2, 3]:
...     for y in [3, 1, 4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

List Comprehensions [4]

```
>>> vec = [-4, -2, 0, 2, 4]

>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]

>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]

>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]

>>> # call a method on each element
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
```

List Comprehensions [5]

```
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

>>> # the tuple must be parenthesized
>>> [x, x**2 for x in range(6)]
  File "<stdin>", line 1, in ?
    [x, x**2 for x in range(6)]
               ^
SyntaxError: invalid syntax

>>> # flatten a list using a list comprehension with two 'for'
>>> vec = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> [num for row in vec for num in row]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

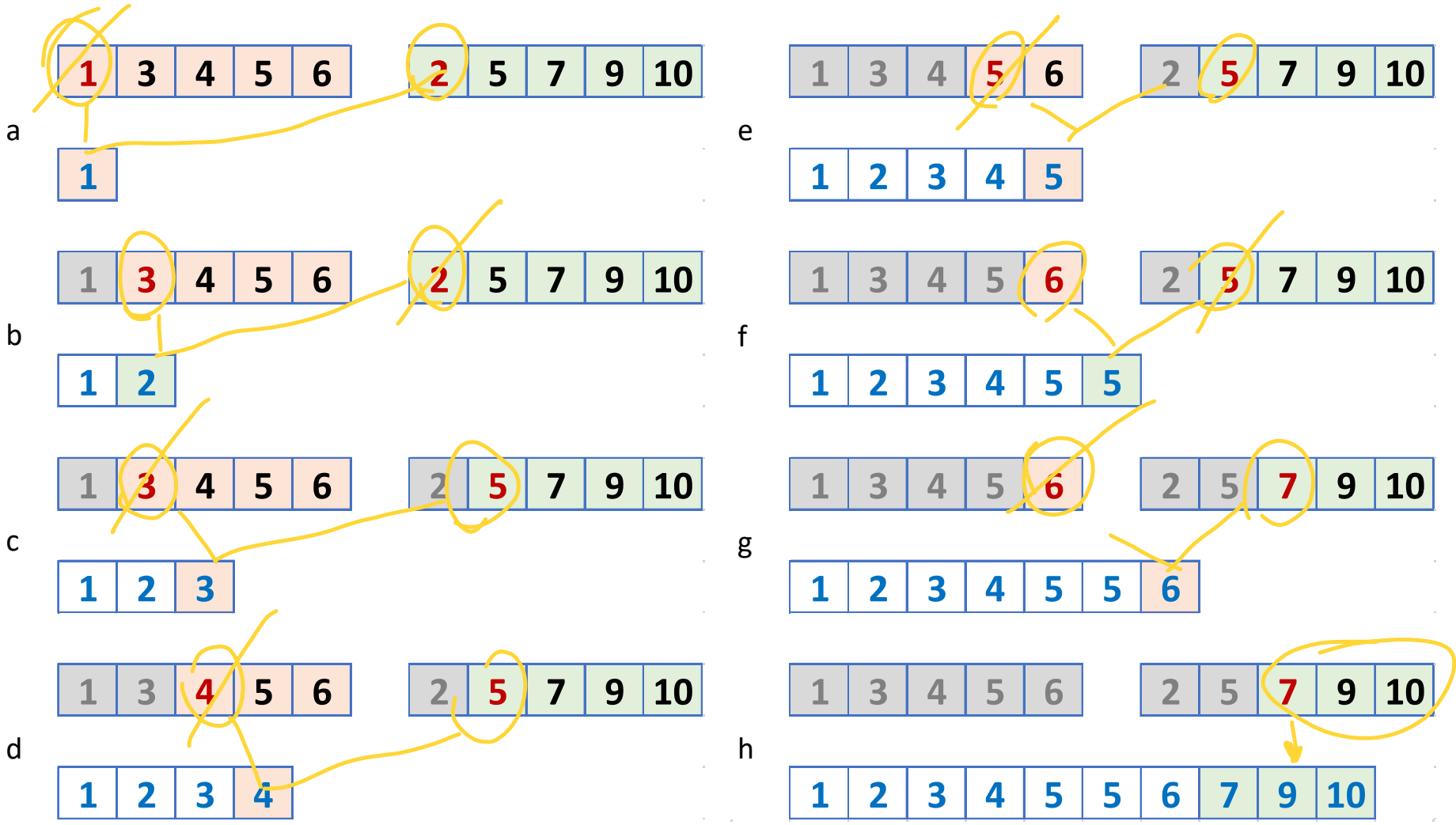
ใส่เข้า tuple

`set` and `dict` Comprehension

```
>>> {x for x in string.ascii_lowercase[:8]}
{'f', 'd', 'e', 'h', 'b', 'c', 'g', 'a'}

>>> {x:[] for x in string.ascii_lowercase[:8]}
{'a': [], 'b': [], 'c': [], 'd': [], 'e': [], 'f': [], 'g':
[], 'h': []}

>>> # creating dict of words with vowels as keys
>>> ll = ['raise', 'touch', 'money', 'roate']
>>> vowels = 'aeiou'

>>> # filter and dict comprehension (v = vowel, w = word)
>>> {v:list(filter(lambda w: v in w, ll)) for v in vowels}
{'a': ['raise', 'roate'], 'e': ['raise', 'money', 'roate'],
'i': ['raise'], 'o': ['touch', 'money', 'roate'],
'u': ['touch']}
```

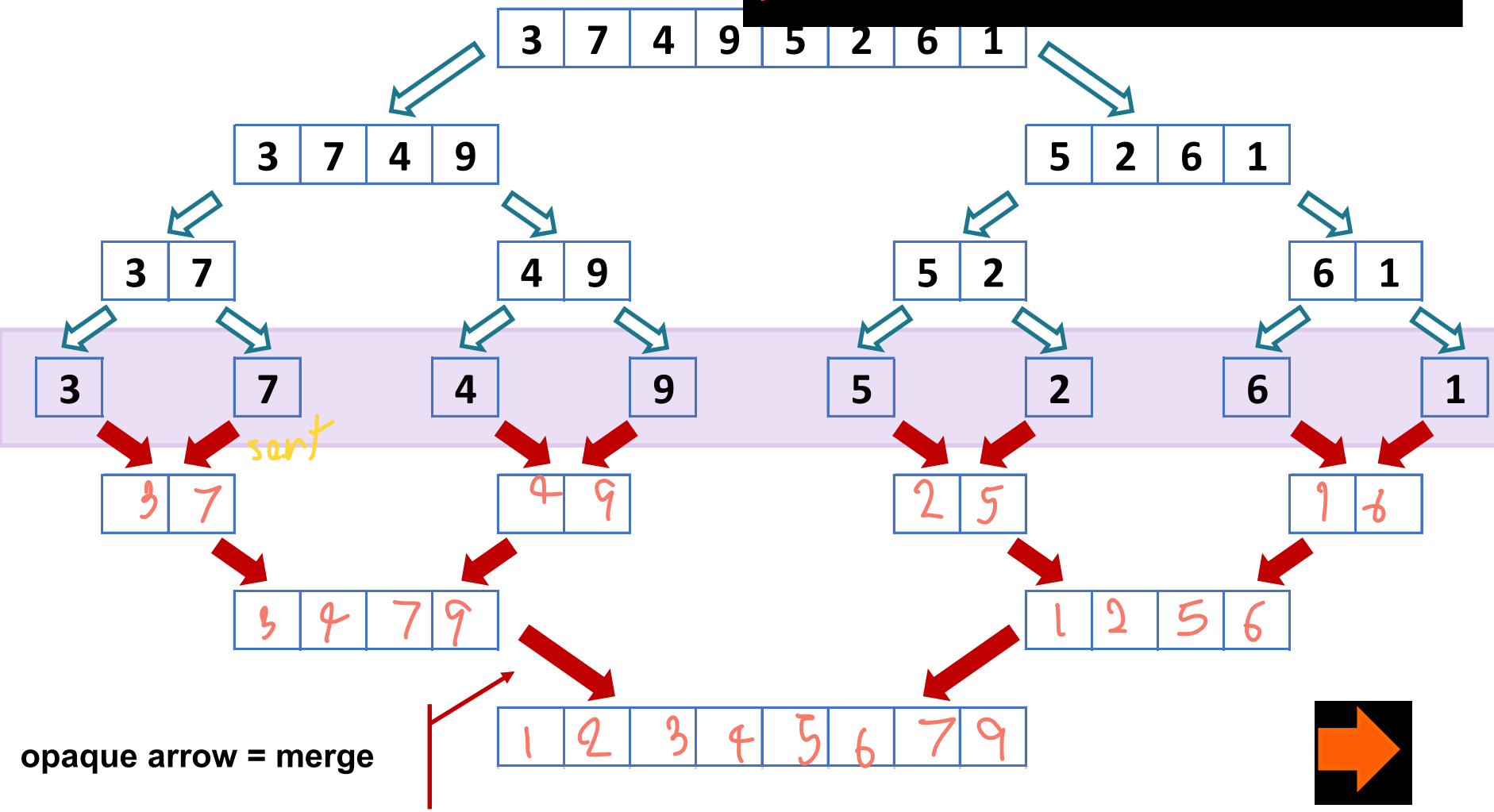
LIST APPLICATION – MERGE SORT

Merge Algorithm - Recap

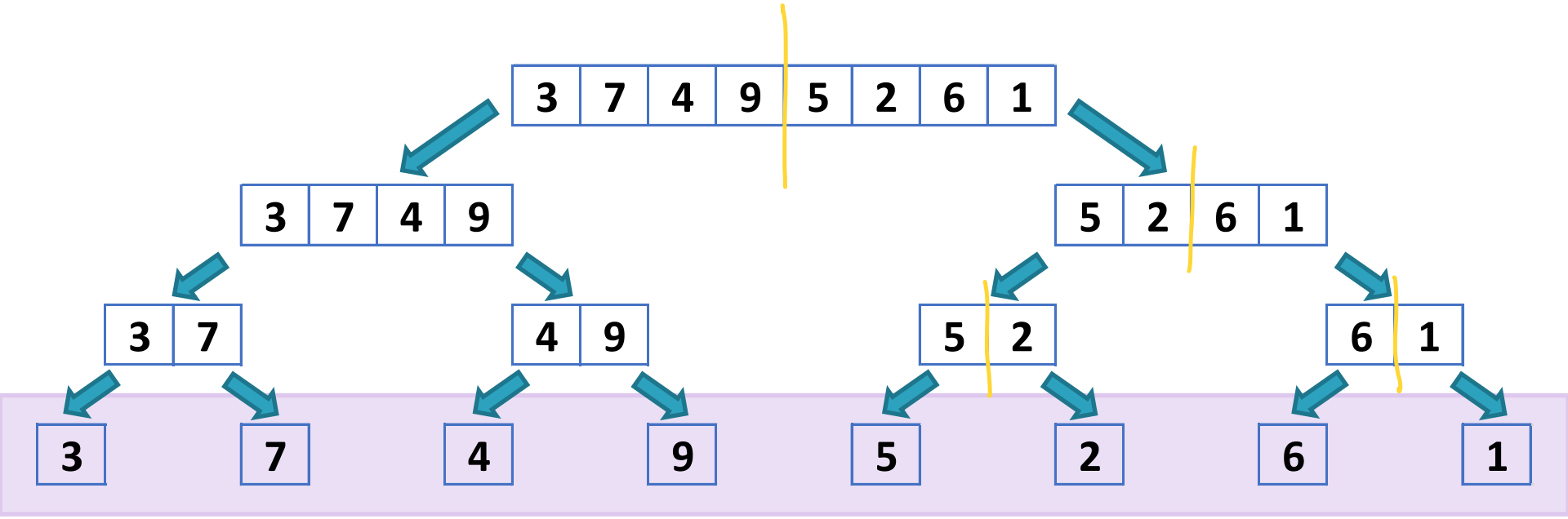


Merge Sort

กรณีหาร 2 ไม่ลงตัว
ให้เอาเศษไว้ทางขวา



w12 Assgmt – simplified_m_sort()




```
>>> list_a = [3, 7, 4, 9, 5, 2, 6, 1]
>>> list_a
[3, 7, 4, 9, 5, 2, 6, 1]
>>> list_b = list(map(lambda x: [x], list_a))
>>> #list comprehension: list_b = [[x] for x in list_a]
>>> list_b
[[3], [7], [4], [9], [5], [2], [6], [1]]
```

References

- **https://docs.python.org/3/howto/functional.html?highlight=lambda**
- **http://www.cs.cmu.edu/~./15110/lectures/lec15-Arrays.pdf**
- **https://docs.python.org/3/library/copy.html**
- **http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-2d-lists.html**