



1. Tuples

• • •

- 2. Box & Pointer
- 3. List



• A <u>mutable</u> sequence of elements

```
lst = [1,2,3]
lst[2] = 0 # lst = (1,2,0)
```

- Is a <u>reference</u> type
- Element could be any type
- Square brackets e.g. [1, 2]

```
lst = [0]
lst = [0] * 4
lst = list((1,2,3))
lst = list(i for i in range(10))
```

#### **Important**

```
Code tracing is important!!
```

Box & Pointer is a tool to trace reference type object!!

#### Recap!!

```
Primitive Type: (int, str, float, bool, none)
- fundamental data structure
```

#### that <u>predefined</u>

- SAME identity!!

```
a = "same"
```

#### Reference Type:

- Look alike *\( + \)* ⇒ Same Identity
- Same Identity ⇒ Look alike

```
lst1 = [1,2]
```

$$lst2 = [1,2]$$

lst1 == lst2 # True

lst1 is lst2 # False



# • List Indexing & Slicing

#### PythonTips!

Indexing and Slicing create an **entirely new object.** 

#### List as Iterator

```
lst = [1, 2, 3]
for ele in lst:
    print(ele)
>>> 1
>>> 2
>>> 3
```

### PythonTips!

```
for idx in range(len(lst)):
    print(i)
    print(lst[i])

for idx, ele in enumerate(lst):
    print(idx)
    print(ele)

COMMON MISTAKE!!
```

COMMON MISTAKE!!
NEVER MODIFY YOUR ITERATING LIST!!!

#### List Addition

#### **Concatenation**

Although list is mutable,

Concatenation always create an entirely new list when trying to "update" it by concatenation

As a result, O(n) time & space

# • List Method : append, extend, insert

```
lst.append(ele/iter)
lst.extend(iter)
lst.insert(insert_pos, ele/iter)
```

#### PythonTips!

Take note that append, extend, insert UPDATE the list (IN PLACE)! It return None!

For Order of Growth, refer to <a href="Python Website">Python Website</a>.

# List Method : copy

# lst.copy()

```
lst = [1,2,[10,11]]
lst2 = lst.copy() # lst2 = [1,2,[10,11]]

lst is lst2 # False
lst == lst2 # True

lst[2] is lst2[2] # True
```

lst[0] = 10 # lst = [10,2,[10,11]]; lst2 = [1,2,[10,11]]

lst[2][0] = 1000 # lst = [10, 2,[1000,11]]; lst2 = [1,2[1000,11]]

# PythonTips!

Always draw BOX & POINTER DIAGRAM!!

Always remember lst.copy() is SHALLOW copy!!

It only copy the first layer with the rule:-

- i) if primitive, copy
- ii) if reference, point



lst.reverse() # lst = [5,4,3,2,1]

• List Method : sort & reverse

```
lst.sort(*key=lambda x:x, *reverse=False)
lst.reverse()

lst = [1,3,2,4,5,]
lst.sort() # lst = [1,2,3,4,5]
```

#### Important!

sort & reverse similar to append, extend,
insert, it modify the existing lst (In
Place)

#### PythonTips!

Take note that when using sort, the element must be comparable with each other!

```
lst = [1,2,[10]]
lst.sort() #ERROR
```

But for reverse, work for anything!

# • List Method : others

```
lst.clear()
lst.pop(*pos=-1)
lst.remove(ele)
lst.count(ele)
lst.index(ele)
```

# PythonTips!

For more info, refer to <a href="PythonWebsite">PythonWebsite</a>
As always Google is your best friend.

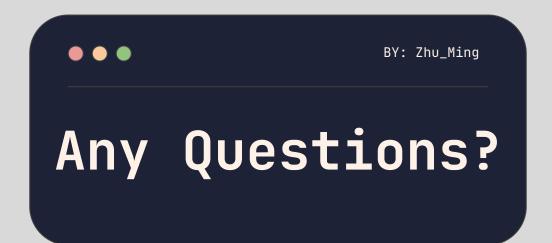
Helpful function on List

```
len(lst)
ele in lst
min, max, map, filter, reversed, sorted
```

#### PETips!

```
sorted(iter, *key=lambda x:x , *reverse=False) is useful!!

lst = [(1,2), (3,2), (2,1)]
sorted(lst, key=lambda tup: tup[0], reverse=True)
>>> [(3,2), (2,1), (1,2)]
```





Ben's function at\_least\_n takes in a list of integers and an integer n and returns the original list with all the integers smaller than n removed.

```
def at_least_n(lst, n):
    for i in range(len(lst)):
        if lst[i] < n:
            lst.remove(lst[i])
    return lst</pre>
def at_least_n(lst, n):
    for i in lst:
        if i < n:
            lst.remove(i)
    return lst
```

Is it correct?? Why and Why Not??
PythonTutor

NOOO! Moral of the story is

NEVER mutate a list while iterating through it with a for loop



Ben's function at\_least\_n takes in a list of integers and an integer n and returns the original list with all the integers smaller than n removed.

```
def at_least_n(lst, n):
    idx = 0
    while i < len(lst)):
        if lst[i] < n:
            lst.pop(i)
        else:
            i += 1
    return lst</pre>
def at_least_n(lst, n):
        spare = lst.copy()
    for i in spare:
            if i < n:
            lst.remove(i)
    return lst</pre>
```



Ben's function at\_least\_n takes in a list of integers and an integer n and returns the original list with all the integers smaller than n removed.

If you want a new list (Effect-free)

```
def at_least_n(lst, n):
                                              def at_least_n(lst, n):
    idx = 0
                                                   spare = lst.copy()
   newlst = []
                                                  newlst = []
    while i < len(lst)):</pre>
                                                  for i in spare:
        if lst[i] < n:</pre>
                                                       if i < n:
            newlst.append(i)
                                                           newlst.append(i)
        else:
                                                   return newlst
            i += 1
    return newlst
def at_least_n(lst, n):
    return list(filter(lambda ele: ele > n, lst))
```



Write a function transpose which takes in a matrix and transposes it, returning a new matrix. PETips!

```
def transpose(matrix):
    result = []
    for row_idx in range(len(lst)):
        for col_idx in range(len(matrix[row_idx])):
            if len(result) <= col_idx:</pre>
                result.append([matrix[row_idx][col_idx]])
            else:
                result[col_idx].append(matrix[row_idx][col_idx])
    return result
def transpose(matrix):
    return list([row[i] for row in matrix] for i in range(len(matrix[0])))
def transpose(matrix):
    return list(list(row) for row in zip(*mat))
    // return list(map(list, zip(*mat)))
```

# tup = ((1,1), (2,2), (3,3))zip(tup) >>> ((1,2,3),(1,2,3) Optional Arguments svntax → \*args def f(\*args): print(f"first: {arqs}") print(\*arqs) f(1,2,3)

>>> first: (1,2,3)

>>> 1 2 3



Now re-implement transpose2 such that it returns the original matrix instead.

```
def transpose2(matrix):
    copy = transpose(matrix)
    matrix.clear()
    matrix.extend(copy)
    return matrix
```



Write a function **row\_sum** which takes in a matrix and returns a list, where the i-th element is the sum of elements in the i-th row of the matrix.



Write a function **col\_sum** which takes in a matrix and returns a list, where the i-th element is the sum of elements in the i-th column of the matrix.

```
def col_sum(matrix):
    result = [0] * len(matrix[0]) # initialise result to be [0, 0, ... 0]
    for row in matrix:
        for col_idx in len(row):
            result[col_idx] += row[col_idx]
    return result
def col_sum(matrix):
    result = []
    for row in matrix:
        result.append(sum(map(lambda row: row[col_idx], matrix)))
    return result
def col_sum(matrix):
                                         def col_sum(matrix):
    trans_matrix = transpose(matrix)
                                             return list(sum(row) for row in transpose(mat))
    result = row_sum(trans_matrix)
                                             // return row_sum(transpose(mat))
    return result
```



Sorting algorithms on [5, 7, 4, 9, 8, 5, 6, 3]

Insertion sort  $O(n^**2)$ , stable Selection sort  $O(n^**2)$ , unstable Bubble sort  $O(n^**2)$ , stable Merge sort  $O(n \log n)$ , stable

#### VisualGo

For more information, CS2040 Data Structure & Algorithms



```
Given a list of students (name, letter grade, score) ...

Write a function mode_score that takes a list of students and returns a list of the mode scores

def mode_score(students):
    # get all score
    all_score = list(map(lambda row: row[2], students))

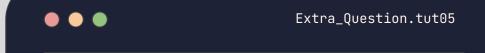
# get the maximum frequency
    mode = max(map(lambda score: all_score.count(score), all_score))

# filter score by its mode, then slides it
    return sorted(filter(lambda score: all_score.count(score) == mode, all_score))[::mode]
```



```
Given a list of students (name, letter grade, score) ...
Write a function top k that takes a list of students and an
integer k and returns a list of k students with the highest scores
in alphabetical order.
 def top_k(students, k):
    # sort by name
    students.sort()
    # sort by score
    students = sorted(students, key=lambda row: row[2], reverse=True)
    # get k_th score
    kth_highest = students[k-1][2]
    # get student who have kth_highest or above
    return list(filter(lambda row: row[2] >= kth_highest, students))
```





# EXTRA Practices

# $\bullet \bullet \bullet$

# (EXTRA) b will never forgot

# Question 1

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



#### OUTPUT:

```
(1,2,3,[1,2,3])
(1,2,3,[])
[1]
(1,2,3,[])
```

