Data structures

1. Counter

Functions

- clear
- copy
- get
- items
- keys
- values
- update
- most_common
- pop
- popitem

```
L = ['Mark', 'Jonny', 'David', 'Mark', 'Jonny',
     'Mark', 'James', 'Mattew']

from collections import counter
C = Counter(L)

C
↳
C['Mark']
C.get('Mark')
```

↳ c.keys()

↳ c.values()

c.update({'Ajay':4})

Soted over

↳ c

↳ c.elements()

for i in c.elements():

print(i)

↳ c.pop('Ajay')

c

↳ c.popitem()

most occurring one

↳ c.most_common(1)

↳ c.update({'Ajay':4})

(2)

c.most_common(2)

↳ clear()
copy()

```
from collections import Counter
L = ['Mark', 'Johny', 'David', 'Mark', 'Johny',
     'Mark', 'James', 'Matthew']

c = Counter(L)
pt(c)
    # Counter({'Mark': 3, 'Johny': 2, 'David': 1,
               'James': 1, 'Matteew': 1})

pt(c['Mark']) #3
pt(c.get('Mark')) #3
pt(c.keys()) # dict_keys(['Mark', 'Johny',
               'David', 'James', 'Matte
c.values() # dict_values([3, 2, 1, 1, 1])

for i in c.elements():
    pt(i, end=' ') # Mark Mark Mark
                   Johny

c.update({'Ajay': 4}) #
pt(c) # Counter({'Ajay': 4, 'Mark': 3
c.pop('Ajay')
c.most_common(1) #[('Mark', 3)]
c.update({'Ajay': 4}) #{'Ajay': 4}
```

↳ Deque

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

↑ left

↑ right

insert
delete

queue → 1 2, 3, 4, 5 ← Del (FIFO)

Stack → 1, 2, 3, 4, 5 (LIFO)

Ins ↑

Insert · Delete

right

↳ append
↳ appendleft
↳ count
↳ extendleft
↳ index

↳ Insert
↳ pop
↳ popleft
↳ remove
↳ reverse
↳ rotate

from collections import deque

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

q = deque(L)

↳ q

q.append(6)

↳

q.appendleft(10)

↳

q.pop() ⇒ last element

q.popleft()

→ q.extend([10,20,30]) right

q.extendleft([11,22,33,44])

↳ q.rotate(2)

↳ q.reverse()

```
from collections import deque

# append appendleft
# clear, copy
# extend, extendleft
# index, insert
# pop, popleft, remove, reverse, rotate

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

q = deque(L)

q.append(50) # [1,2,3,4,5,50]

q.appendleft(10)

pt(q)  # deque([10,1,2,2,3,4,50])

q.pop()
# [10,1,2,3,4,5]

q.popleft
# [1,2,3,4,5]

q.extend([50,60,70])
[1,2,3,4,5,50,60,70]

q.extendleft([10,20,30])
[30,20,10,1,2,3,4,5,50,60,70]
```

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

q = deque(L)

q.Rotate(1)

[5,1,2,3,4]

q.Rotate(2)

[3,4,5,1,2]

q.Reverse()

[2,1,5,4,3]

q.insert(2,300)

[2,1,300,5,4,3]

q.index(300) # 2.

↳ Array

arrayName = array.array(type, [array items])

L = [10, 'Hello', 12.59] → Heterogenious → List

~~Homogenious~~ → Array

Code	Type	size	Ch
	int	1	
b	unsigned	1	
B	signed	2	
i	unsigned	2	
I	signed long	4	
l	unsigned long	4	
L	signed long long	8	
q	unsigned long long	8	
Q		4	
f	float	8	
d	Double float		

↳ impot array
ar1 = array.array ('i', [10,20,30])

ar
S1 = b 'abcdef'
ar2 = array.array ('b', S1)

↳ ar2[0]

ar2[1:3]

b'[98,99]

↳ ar2.append(103)
ar2.count(103)
ar2.extend([])
ar2.frombytes()
ar2.index(102)
ar2.remove()
ar2.typecode()

import array
ar1 = array.array('i', [10,20,30])
pt(ar1) # array('i', [10,20,30,40])
S = b'abcde'

Heap

H = [10, 20, 50, 40, 60, 30, 70]

heapq

priority queue

smallest number → highest priority

↳ heapify
↳ heappush
heappop
↳ heapreplace
↳ nlargest
↳ nsmallest

↳ import heapq
H = [ ]
↳ heapq.heappush(H, 20)
↳ heapq.heappush(H, 50)
H
↳ heapq.heappush(H, 10)
[20, 50]
[H, 10]
(10, 50, 20)
40
30
60
Tree heap
H = [10, 30, 20, 50, 40, 60]
↳ heapq.heappop(H)
[20, 30, 60, 50, 40]

↳ heapq.heappop(H)

[30,40,60,50]

↳ H = [50,30,60,40,70,20,10]

heapq.heapify(H)

H

[10,30,20,40,70,50,60] ↳ HeapTree

↳ heapq.nlargest(2,H)

[70,60]

.nsmallest(3,H)

[10,20,30]

heapq

import

h = [ ]

↳ heapq.heappush(h,20)

h 20

↳ heapq.heappush(h,10)

[10,20]

.heappush(h,50)

↳ (10,20,50) (h,30)

(10,20,50,30) (h,40)

(h,60)

↳ [10,20,50,30,40,60]

↳ [10, 20, 50, 30, 40, 60]

heapq.heappop(h)

10

h = [20, 30, 50, 60, 40]

. heappop(h) 20 [30, 40, 50, 60]

. heappop(h)

h = 30 [40, 60, 50]

↳ h = [50, 30, 60, 40, 70, 20, 10]

heapq.heapify(h)

h

[10, 30, 20, 40, 70, 50, 60]

↳ heapq.nlargest(2, h)

[70, 60]

↳ heapq.nsmallest(2, h)

[10, 20]

b = [10, 20, 20, 30, 50, 60, 70, 90]

↳ insort( )
↳ insort_left( )
↳ insort_right( )

↳ bisect
↳ bisect_left( )
↳ bisect_right( )

↳ import bisect
L = [10, 20, 20, 30, 40, 50, 60, 70, 90]

↳ bisect.insort(L, 25)
↳ bisect.insort_left(L, 90)

↳ id(L[9]
id(L[10])

↳ bisect.bisect(L, 5) — 0
bisect.bisect_right(L, 20) — 3
_left (L, 20) → 1

import bisect

L = [10, 20, 20, 30, 50, 60, 70, 90, 90]

↳ bisect.insort(L, 25)

[10, 20, 20, 25, 30, 50, 60, 70, 90, 90]

bisect.insort(L, 20)

10 20 20 20 25 30, 50 60 70909

insort_left(L, 30)

10 20 20 20 25 30 30 50 60 709090

insort_right(L, 30)

10 20 20 20 25 30 30 30 50 60 709090

↳ bisect.bisect(L, 20) #4

bisect_left(L, 30) #5

bisect_right(L, 30) #8

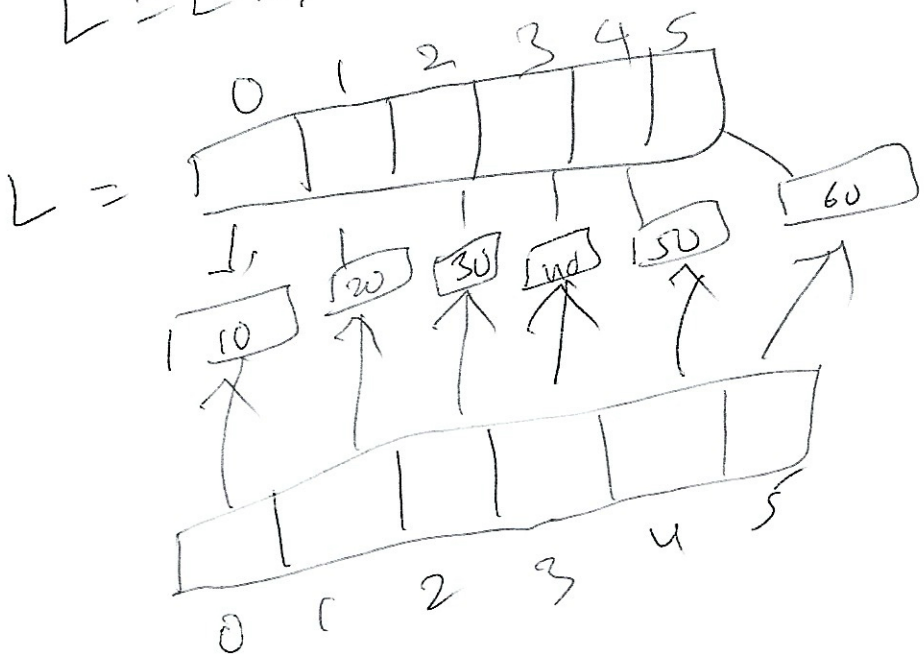
copy

Duplicating objects

copy.copy(x)

copy.deepcopy(x)

L = [10, 20, 30, 40, 50, 60]



import copy

L = [10, 20, 30, 40, 50]

L1 = copy.copy(L)

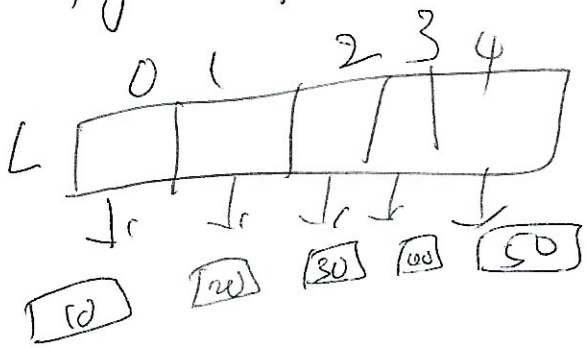
id(L) → id(L[0])

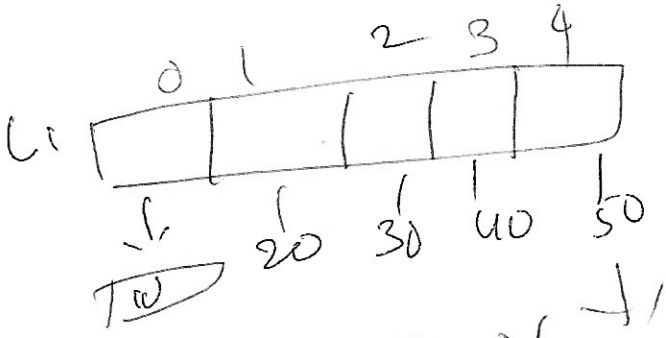
id(L1) → id(L1[0])

✓

✗

copy.deepcopy()





int
float
str
} deepcopy does not work with int, float, str objects

↳ class Person:
    def __init__(self, name):
        self.name = name

L = [Person('John'), Person('Tim'), Person('Jim')]

L1 = copy.deepcopy(L)

id(L[0])

id[L1[0]]

Three Most Common words

''' python is an esy prog.

python syntax

is esy learning '''

{ re.findall('\w+')

Counter }

```
import re
from collections import Counter
words = re.findall('\w+', text)
Count = Counter(words)
pt(Count.most_common(3))

word = re.findall('\w+', text)
Count = Counter(word)
Count.most_common(3)
```

Inventory

inventory = {'apple': 50, 'mango': 100, 'banana': 120 'orange': 70}

order = {'apple': 10, 'mango': 12, 'banana': 15, 'orange': 5}

40 88 105 65

update_inventory(order)

inventory = Counter(...)

order = Counter(...)

def update_inventory(order):

from collections import Counter

inventory = Counter(apple=50, mango=100, banana=120, orange=70)

def updateInventory(order):

inventory.subtract(order)

order = Counter(apple=10, mango=15, banana=5, orange=5)

updateInventory(order)

pt(Inventory)

```
from collections import deque

customers = deque()

def walk_in(cust):
    customers.append(cust)

def serviced():
    x = customers.popleft()
    pt('{x}, left :)

walk_in('John')
walk_in('Rose')
walk_in('NNR')

serviced()
serviced()
serviced()
pt(customers)
```

Circular Queue

```
from collections import deque

students = [

st = deque(students)

def serve():
    pt('serve is order:', st)
    st.rotate(-1)

pt('Bf')
serve()
pt('Lh')
serve()
pt('DR')
serve()
```

def generate_bill(cart)

(11)

⇒ from collections import Counter

prices = {'soap': 50, 'Toothpaste': 25, 'shampoo': 45,
'Toothbrush': 15.99}

def generate_bill(cart):
pf ('product price qty sub-total')
Total = 0
for i, q in cart.items():
pf(f'{i:10}:{prices[i]:5} x {q:2}
= {prices[i] * cart[i]}')
Total += cart[i] * prices[i]
return Total

cart = Counter(soap=5, Toothpaste=1, shampoo=2,
Toothbrush=3)

pf(cart)
Total = generate_bill(cart)
pf(Total)

First dup

nums = [10, 20, 13, 14, 15, 13, 17, 10, 20, 13]

array

set()

```
def find-dups(nums):
    =
    =
```

return -1

```
array = array([ ])
pt(find-duplicate(arr))
```

```
import array
set1 = set()
def find-dups(arr):
    for i in arr:
        if i not in set1:
            set1.add(i)
        else: return i
    else: pt(set1)
        return -1
```

```
arr = array.array
      ('i', [10 ...])
pt(find-dups(arr))
```

Find missing returned numbers

[1, 8, 10]

actual_sum = 1+
given_sum = 1+

10=55
10=46

def missing_num(arr)
==
==
return ?

arr = array('i', [ ])
pt(missing_no(arr))

import array

```
def missing_num(given_list):
    strt = given_list[0]
    end = given_list[len(ar)-1]
    actual_list = list(range(strt, end+1))
    sum1 = sum(actual_list)
    sum2 = sum(given_list)
    return sum1 - sum2

ar = array.array('i', [1, 8,10])
print(missing_num(ar))
```

Find a pair of integers with highest product

i/p = [2,4,6,8,3,7,9] [8,9] 8*9 = 72

[0,-1,-3,-5,-8,2,4] = [-8,-5]

```
import array

def max_product(ar):
    x = ar[0]
    y = ar[1]
    for i in range(0, len(ar)):
        for j in range(i+1, len(ar)):
            if (ar[i]*ar[j]) > (x*y):
                x = ar[i]
                y = ar[j]

    pt(x,y)
    return x,y

a = array.array('i', [
pt(max_product(a))
```

```
import heapq
def heap_sort(elements):
    heapq.heapify(elements)
    sorted_list = []
    for i in range(len(elements)):
        x = heapq.heappop(elements)
        sorted_list.append(x)
    return sorted_list

elements = [12, 14, 8, 7, 3, -5, 6, 2]
sorted_list = heap_sort(elements)
print('sorted list:', sorted_list)
```

K^{th} largest number

List = [10, 29, 64, 90, 82, 74, 33]

K^{th} →
1st → 90
2nd → 82
3rd → 64

heapq
pop → smallest

1st → smallest
2nd → smallest
3rd → smallest

↳ [-10, -29, -64, -90, -82, -74, -33]
-90, -82
+ → 90 82

def Kth largest( )
=
=
=

List = [
Kth Lgest (List)

```
import heapq
def kth_largest(ele, k):
    sorted_list = []
    for e in ele:
        heapq.heappush(sorted_list, -e)
    for i in range(k-1):
        heapq.heappop(sorted_list)
    return heapq.heappop(sorted_list)
element = [10, 34, 64, 90, 83, 89, 70, 63]
print("kth largest(element))
```

-10 -34, -64, -90, -83

→ [-90, -89, -83, -70, -64,]

```
import bisect

def Insertion_sort(A):
    sort = []
    for i in A:
        bisect.insort(sort, i)
        pt(sort)
    return sort

A = [24, 10, 34, 56, 7, 8, 9, 0, 19]
pt(Insertion_sort(A))
```