LIST

List 1

```
x=5
my_list1=[2,6,8,"python",6]
my_lst=[ ]
my_lst1=[1,2.5,3.5,"Hello",6]
bool(my_lst) ⇒ False
bool(my_lst1) ⇒ True
```

↳ Slicing

```
my_list=[3,2.4,"python",5.6]
my_list[0] ⇒ 3
my_list[-1]=5.6
type(my_list) # <class 'list'>
my_list[3][1]=#y.
my_list[:] # [3,2.4,'python',5.6]
my_list[0:] #  "
my_list[1:4] ⇒ 2.4,"python",5.6
```

Lists are mutable, unlike string immutable

↳ "append", "clear", "copy", "count", "extend", "index", "insert", "pop", "remove", "reverse", "sort"

my_list.append(56)
# [3,4,5,6,7,8,56]
my_list.insert(1,99)
[3,99,4,5,6,7,8,56]

extend

my_list = [3,4,5,6,7,8]
my_new_list = [9,10]
my_list.append(my_new_list)
[3,4,5,6,7,8,[9,10]]
my_list = [3,4,5,6,7,8]
my_list.extend(my_new_list)
[3,4,5,6,7,8,9,10] // list with normal dots

pop / remove

my_list.remove(8)
[3,4,5,6,7,9]
my_list.remove(11) # ValueError
[3,4,5,6,7,8,9] // last index ✓
my_list.pop()
[3,4,5,6,7,8]
pop(6) # IndexError
↳ my_list = [3,4,5,6,7,8,9]
my_list.pop(4) # index ✓
index
[3,4,5,6,8,9]

List 2

Reverse/Sort

```
my_list=[3,4,5,6,7,8,9]
my_list.reverse()
[9,8,7,6,5,4,3]
// my_list.sort()
[3,4,5,6,7,8,9] //Ascending
my_list.reverse() // Descending
[9,8,7,6,5,4,3]
// my_list=[3,4,5,6,7,8,9]
my_list.sort(reverse=True)
// [9,8,7,6,5,4,3]
```

⇒ Ascending at a time

ABDUL:

↳ List :- iterable

```
L4=list(3,5,7,9)  ⇒ [3,5,7,9]
L5=list('abcde')  ⇒ ['a','b','c','d','e']
L6=[]
```

↳ `L1=[6,5,4,2,3,2]`
`for i in L1:`
`pt(i,end=" ")`

↳ `L1=[7,3.2,"john",True,5+6j)`
`L1[3]=5`
`7,3.2,"john",5,5+6j`
`L1[2][3]='9'` ✗ TypeError

Mutable

↳ L1 = [2, 4, 6, 8, 10, 12]

L2 = [ ]

L2 [2] = 16

L1.append (25)

[2, 4, 16, 8, 10, 12, 25]

↳ L1.remove (10)

pt(L1)

[2, 4, 16, 8, 12, 25]

↳ ordered collection of heterogenous elements

↳ It's mutable

↳ Can have duplicates

Index_Slice

↳ List is mutable

Read/write

Read :

Indexing

Slicing

↳ Read Indexing

L1 = [3, 6, 9, 12, 15, 18, 21]

L1[4] = 15

L1[-3] = 15

L1[::] → [3, 6, 9, 12, 15, 18, 21] # start, stop, step

L1[:] # Begin, End.

Abdul

List 3

↳ L1 = [3, 6, 9, 12, 15, 18, 21]

L1[4] = 15

L1[-3] = 15

↳ L1[:] } [3, 6, 9, 12, 15, 18, 21]

L1[:]

↳ L1[2:]

[9, 12, 15, 18, 21]

L1[:6]

[3, 6, 9, 12, 15, 18]

↳ L1[2:6]

[9, 12, 15, 18]

L1[-5:-2] -ve index

forward print

[9, 12, 15]

↳ [::2] → [3, 9, 15, 21]

[::3] → [3, 12, 21]

[4::-1] → [15, 12, 9, 6, 3]

Reverse

[4:0:-1] → [15, 12, 9, 6]

[-3:-7:-1] → [15, 12, 9, 6]

↳

L1 = [1, 2, 3, 4, 5]

L1[1] = 10

L1[3] = [10, 11]

L1 = [1, 10, 3, [10, 11], 5]

↳ L1 = [1, 2, 3, 4, 5]

L1[0:0] = [10]

[10, 1, 2, 3, 4, 5]

↳ L1[3:3] = 10

[10, 1, 2, 10, 3, 4, 5]

L1[9:9] = [10]

[1, 2, 3, 4, 5, 10]

↳ L1 = [1, 2, 3, 4, 5]

L1[3:3] = [10, 11, 12]

[1, 2, 3, 10, 11, 12, 4, 5]

// as many will value we can give

↳ L1 = [1, 2, 3, 4, 5]

L1[1:4] = [10, 11]

[1, 10, 11, 5]

Step

L1 = [1, 2, 3, 4, 5]

L1[::-1] = [10, 11, 12, 13, 14]

[14, 13, 12, 11, 10]

↳ L1 = [1, 2, 3, 4, 5]

L1[3:0:-1] = [12, 13, 14]

[1, 14, 13, 12, 5]

// Concat / Repetition

List 4

↳ L1 = [1, 2, 3]
L2 = [8, 9, 10]
L1 + L2 = [1, 2, 3, 8, 9, 10]

↳ L1 + 4 ✗
L1 + [4] ✓

↳ L1 = [1, 2, 3]
L1.extend ([4, 5, 6])
[1, 2, 3, 4, 5, 6]

// Repetation

L1 = [1, 2, 3]
L1 = L1 * 3 = [1, 2, 3, 1, 2, 3, 1, 2, 3]

// Membership

a. L1 = [1, 2, 3, 4, 5]
3 in L1 → True

b. L = [[1, 2], [3, 4], 5]
3 in L1 → False
[3, 4] in L1 → True

c. for x in L1:
print(x, end = " ")

List Comparision

L1 = [1,2,3]
L2 = [1,2,3]
L3 = [3,2,1]

L1 == L2 → True
L1 < L3 → True

L1 = [1,2,3]
L2 = [1,2,3,4]
L3 = [1,2,1]

L1 < L2 → True
L1 < L3 → false

L1 = [2]
L3 = [1,2,1]
L1 > L3 → True

List Travel

```
L1 = [5,6,7,8]
for x in L1:
    pt(x, end=" ")
```

```
for i in range(len
    pt(L1[i],
```

```
i = 0
while (i < len(L1)):
    pt(L1[i], en
    i = i + 1
```

Lise 5

// append :- Insert at end

↳ L1 = [1,2,3,4,5]
L1.append(6)
[1,2,3,4,5,6]

↳ L1.append(7,8) X
TypeError

↳ L1 = [ ]
L1.append(10) [10]
L1.append("python") [10, "python"]
L1.append([1,2]) ⇒ (10, "python", [1,2])

↳ L1 = [1,2,3,4,5]
L1[5:5] = 10
[1,2,3,4,5,10]

// extend

L1 = [1,2,3,4]
L1.extend([5,6,7])
[1,2,3,4,5,6,7]
L1.extend('python')
[1,2,3,4,5,6,7,'p','y','t','h','o','n']
L1.extend(range(5,8))

// Insert

```
L1=[1,2,3,4]
L1.insert(0,50)
[50,1,2,3,4]
L1.insert(70,"python")
[50,1,2,3,4,'python']
```

// L1=[1,2,3,4]
L1[2:2]=[55]
[1,2,55,3,4]

// Copy

```
L1=[1,2,3,4]
L2=L1.copy()
L2[0]=6
L1=[1,2,3,4]
L2=[6,2,3,4]
```

Remove
- pop(index)
- remove(element)
- clear()
- del

```
L1=[1,2,3,4,5,6]
L1.pop() #6
L1=[1,2,3,4,5,6]
L1.pop(2) = #3
```

```
L1=[]
L1.pop()
#IndexError
```

List 6

// Remove

L1=[1,2,3,4,5,3,3]

L1.remove(3)

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

L1.remove(3)

[1,2,4,5,3]

// clear()

a. L1=[1,2,3,4,5]

L1.clear()

[]

b. L1=[1,2,3,4,5,6]

del L1[3]

// [1,2,3,5,6]

c. L1=[1,2,3,4,5,6]

del L1[1:4]

[1,5,6]

d. L1=[1,2,3,4,5,6]

del L1

pt(L1) → NameError : "L1" not defined

Index-Reverse-Sort

L1=[10,20,30,40,10,20,30,20]

L1.index(20) #1

L1.index(20,2) #5

(20,2,6) #5

Count

```
L1 = [10,20,30,40,10,30,20,40]
L1.count(20)   #2
```

Reverse

```
L1 = [10,20,30,40,50,60,70]
L1.reverse()
[70,60,50,40,30,20,10]
```

Sort

```
L1 = [70,10,60,20,50,30,40]
L1.sort()  → increasing order
[10 . . 70]
L1 = [70,10,60,20,50,30,40]
L1.sort(reverse=True)
// decreasing
[70,60,50,40,30,20,10]
```

↳ `L1 = ["coat","python","block","cat"]`
`L1.sort()`
b cat c p

↳ `L1.sort(key=len)`
cat coat block 'python'

↳ `L1 = ["apple","Bat","cat","DOG"]`
`L1.sort()`
`["Bat","DOG","apple","Cat"]`

`L1.sort(key=str.lower)`
`["apple","Bat","Cat","DOG"]`

List comprehensions

L1 = list(iterable)

L = [exp for item in iterable]

for x in range(1,3):
    print(x)

↳ L1 = [x for x in range(1,5)]
[1,2,3,4]

↳ L2 = [x**2 for x in range(1,5)]

L3 = [x.casefold() for x in "Python"]
['p','y','t','h','o','n']

L4 = [int(x) for x in "12345"]

L5 = [x for x in "ab*cd7e" if x.isalpha()]
['a','b','c','d','e']

Nested Lists

L3 = [[1,2], 3, 4, 5]
L4 = [[1,2], [3,4], [5,6]]
L6 = [[1,2], [3,4], [5,6]]
L6[0] = [1,2]
L6[2][0] = 5

L1 = [[1,2,3,4], [5,6,7,8], [9,10,11,12]]
L1[0][2] = #3
L1[1][2] = 7
L1[2][3] = 12

Wages

```
hours=input("    ")
wages=int(input("    "))
hours=hours.split()
work-hours=[int(x) for x in hours]
total_hours=sum(work_hours)

if total_hours<=40:
    total_wages=total_hours * wages
else:
    over=total_hours-40
    total_wages=(total_hours *wages)
                +(over * wages *1.5)
```

output:

8 8 8 8 8

12

480

Remove dups

```
L1=[
L2=[]
for i in L1:
    if i not L2:
        L2.append(i)

pt(L2)
```

Split odd / even

List 6

Split odd/even

```
L1=[1,2,3,4,5,6,8,9]
L_odd=[]
L_even=[]
odd=[x for x in L1 if x%2==0]
even=[x for x in L1 if x%2!=0]
```

palindrom

```
lst=[5,4,3,3,4,5]
rev=lst[::-1]
if lst==rev:
    pt("palindrom")
else:
    pt("not palindrom")
```

Rotate:-

```
L1=[1,2,3,4,5,6]
n=int(input(""))
L2=L1[n:]+L1[:n]
```

Shuffle:-

```
import random as rd          → #0.05
rd.random()
rd.randint(1,100) #94
rd.randint(1,100) # 94
rd.randrange(1,102) #1
for i in range(5):
    rd.randint(1,100)
```

42 30 68 42 16

```
rd.seed(10)
for i in range(5):
    rd.randint(1,100)
```

74 5 55 62 74 : Run1

:

Run4

```
L1=[1,2,3,4,5,6,7]
rd.shuffle(L1)
pt(L1)
```

#[2, 4, 6, 3, 7, 5, 1]

List permutation

```
import itertools as it
lst=["A","B","C","D"]
perms=it.permutations(lst,r=2)
for i in perms:
    pt(i,end="  ")
combi=it.combinations(lst,r=2)
combi=list(combi)
for i in combi:
    pt(i)
```

```
# product=it.product(lst, repeat=2)
product=list(product)
for i in product:
    pt(i,end="  ")
```

('A','A') ('A','B') ('A','C')

```
import statistics st
lst = [6,8,4,9,3,10,8,11,5,7,8,4,2]

mean = st.mean(lst)
median = st.median(lst)
mode = st.mode(lst)

pt(mean)
#6.538
pt(median)
#7
pt(mode)
#8
```

List of

Longest

```
lists = [[1,2,3],[1,1,1,1,1],[2,2,3,3]]
pt(max(lists, key=len))
```