LISTS

Topic

- Lists

Function – Scope

Ex: 01

INPUT	**OUTPUT**

`def ABC ():`
`  sum = 10`
`  for x in range (10):`
`  sum = 0`
`  print (sum)`
`ABC ()`
	0

Ex: 02

INPUT	**OUTPUT**

`sum = 0 # global scope`
`def ABC ():`
`  sum = 5`
`  print ("inner sum", sum)`
 	
`ABC ()`
`print ("outer sum", sum)`
	inner sum 5
	outer sum 0

Global scope: Any variable outside a function is called to be in global scope.

Local scope: Anything inside a function is called local scope.

When we are accessing the sum inside the function. We will get the sum from the local scope, else outside the function we will get from global scope.

Any variable declared outside the function name will have global scope. Python will search for sum in local scope but, if it is not able to find it, then it will search in the global scope.

Ex:03

INPUT:	**OUTPUT:**

`#Example 3`
`sum = 0 # global scope`
`def ABC ():`
`    sum = 10`
`    for i in range (1,10):`
`        sum = 5`
`    print ("inner sum:", sum)`
`sum = 10`
`print ("outer sum:", sum)`
`ABC ()`
	outer sum: 10
	inner sum: 5

Data Structures:

A data structure is a collection of data type 'values' which are stored and organized in such a way that it allows for efficient access and modification.

In simple words, “how do you store data”.

List is a predefined data structure. They are like small boxes.

0	1	2	3	4	5	6

Whenever you run a program or application, that gets stored in RAM. It can be anything, when you write a = 5, it means that you are storing this variable in RAM at some particular location, with the value 5.

When you say a+=6, it will fetch the value from the RAM update 6 and will put it back in the RAM.

Whenever we declare or store something in a variable, the variable is assigned to a unique address in RAM and value of the variable is stored at that position.

- When you say a = 5 can be stored in any random memory block.

Lists

- It is actually a chain of data.
- Count of list start from 0
- You can store anything in list (int, str, float)

- The numbering in list is called "***index***"

Ex: a =, [5, 12, "raj", 15, 16]

Size of A = 5 and value at the 0^{th} index is 5.

Lists is a type of data structure, where all the values are present next to each other (contiguous) and it has indexes which starts from 0 and go to the (size of the list -1).

In python we denote list with [ ] or list( )

Declare a list:

fruits = ["apple", "banana", "papaya"]

>>> fruits = ["apple", "banana", "papaya"]

>>> fruits

['apple', 'banana', 'papaya']

>>> type (fruits)

<class 'list'>

API – Application Programming Interface

(methods/functions/tools available when we use lists)

1. To get size of list – len

len (fruits): will tell you the size of the list. → 3

2. Accessing the item of the list

fruits [1] → banana

fruits [0] → apple

fruits [2] → papaya

fruits [3] → list index out of range

Ex: 04

INPUT	**OUTPUT:**

`fruits = ["apple", "banana", "papaya"]`
 	
`print ("length of the list is", len(fruits))`
`print ("element at index 2", fruits[2])`
	length of the list is 3
	
	element at index 2 papaya

The list is stored in contiguous section of the memory, which means, one after the other in increasing order.

3. Slices in Lists

A = [2, 5, 7, 8, 15, 20, 30]

A [2:5] = 7, 8, 15

A [4:6] = 15, 20

A [0:4] = 0, 1, 2, 3

INPUT:

>>> A = [''apple'', 5, 7.8, 100, 220, 'HJ']

>>> A [1:3]

[5, 7.8] → this is called a slice or a sub-list

When we as A [start:end] → A[start], A[start+1], A[start+2]……A[end-1]

A = [5,20,30,40,60]

A [3:5] = A [3], A [4] = 40, 60

Similar to step in range:

A [1:5:2] → 1, 3

4. Negative Indexing:

-6	-5	-4	-3	-2	-1
0	1	2	3	4	5
[1	2	3	4	5	6]

>>> A

['apple', 5, 7.8, 100, 220, 'HJ']

>>> A [-2]

220

>>>

It takes the value of the second element from last. In the same way if we use A[-3], it will show us the third element from the last.

>>> A [2:]

[7.8, 100, 220, 'HJ']

A [: idx] -> will print from start till idx - 1

A [idx:] -> will print from idx till end

A [:] -> will print whole list

A [0:-1] -> 0 till second last element

Q1) names = [“Rahul”, “Shyam”, “Sadaf”, “Akash”]

INPUT:

Method 01:

```
names = ["Rahul", "Shyam", "Sadaf", "Akash"]

for idx in range (0, len(names)):
    print ("Hello", names[idx])
```

Method 02:

```
names = ["Rahul", "Shyam", "Sadaf", "Akash"]

for name in names:
    print ("Hello", name)
```

OUTPUT:

Hello Rahul

Hello Shyam

Hello Sadaf

Hello Akash

If we want an output as following

1. Hello Rahul

2. Hello Shyam

3. Hello Sadaf

4. Hello Akash

INPUT:

```
names = ["Rahul", "Shyam", "Sadaf", "Akash"]

cnt = 1
for name in names:
    print (cnt, ".", "Hello", name)
    cnt += 1
```

>>> A = [1,2,3,4,5,6,7,8,9,10]

>>> A [:-1]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> A = A[:-1]

>>> A [:-1]

[1, 2, 3, 4, 5, 6, 7, 8]

>>> A = A[:-1]

>>> A [:-1]

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

>>> A = A[:-1]

>>> A [:-1]

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

>>> A = A [:-1]

>>> A [:-1]

[1, 2, 3, 4, 5]

>>> A = A [:-1]

>>> A [:-1]

[1, 2, 3, 4]

When we use the come A [:-1] , it will give us a slice. Then A is being updated as A[:-1]. So, the value of A from [1,2,3,4,5,6,7,8,9,10] will become [1,2,3,4,5,6,7,8,9]

Based on this understanding,

INPUT:

```
names = ["Rahul", "Shyam", "Sadaf", "Akash"]
cnt = 1
while len(names) > 0:
    print("Hello", names[0])
    names = names[1:]
    cnt += 1
```

OUTPUT:

Hello Rahul

Hello Shyam

Hello Sadaf

Hello Akash