Topics:

- Lists
- Tuples
- String
- Addition
- Dictionary

Let's suppose:

A = ["apple", 5, 2, 3.3, "banana"]

A [::-1] = ["banana", 3.3, 2, 5, "aaple"] # reverse of the list

A = ["A", "B", "C", "D"] → how to revrse without slicing or revrse.

API

5. **append** → this will add the new element at the end.

INPUT:	**OUTPUT:**

`A = list ()`
`print (A)`
`A.append(5)`
`print (A)`
`A.append(6)`
`print (A)`
`A.append("Rahul")`
`print (A)`
	[ ]
	[5]
	[5, 6]
	[5, 6, 'Rahul']

6. insert → this insert an element before an element.

INPUT:	**OUTPUT**:
`A.insert (1, "Shyam")` `print (A)`	[5, 6, 'Rahul'] [5, 'Shyam', 6, 'Rahul']

A.pop () → removes the last value in the list. It is just as A[:-1].

Tuples:

Tuples are list whose values you cannot change once you have constructed. You cannot delete or insert with tuples. They are immutable (this which we cannot change).

- Mutable: things which you can change.

Tuples are represented by tuples or ()

A = (“apple”, “banana”, “ham”)

Print(type(A))

For I in A:

    Print (i)

OUTPUT:

tuple

You can get a reverse of it as well:

INPUT:	**OUTPUT**

`A = ("apple", "banana", "ham")`
`A = A [::-1]`
`print (A)`
	('ham', 'banana', 'apple')

You can assign one tuple to another, but you cannot change the values in a tuple.

INPUT:	**OUTPUT:**

`A = ("apple", "banana", "ham")`
`B = ["apple", "banana", "grapes"]`
`print (B)`
`B [1] = "mango"`
`print (B)`
	['apple', 'banana', 'grapes']
	['apple', 'mango', 'grapes']

Why Tuple?

It will protect our data, what ever you have declared is locked and nobody can change it. It is also used for constants. Further in OOPS.

You can type cast between a list and a tuple. HOW?

Ex: 01

INPUT:	**OUTPUT:**

`A = (1, 2, 3, 4, 5)`
`B = list(A)`
 	
`Print ("A is a ", type(A))`
`Print ("B is a ", type(B))`
	A is a <class 'tuple'>
	B is a <class 'list'>`

Once you have changed the tuple to a list, you can make changes to the list. But as a tuple, we cannot make any changes in it. And once you have made changes, then you can change it back to tuple.

- Whenever you are declaring A = [ ] then it is a list, but when you declare A = ( ), then it is a tuple. The remaining operations can be done with [ ].

MCQ's

1.

```
def Check (x):
    res = list ()
    for I in range (x):
        res.append (I)
    return res
Check (5)
```

2.

```
def Check (x):
    res = list ()
    for I in range (x):
        res.append (x)
    return res
Check (5)
```

3.

```
A = [1, 2]
A = tuple (A)
for x in range (A):
    B = []
    B.append (x + 5)
print (B)
```

4.

```
A = [60, 50, 40, 30, 20,
10]
A = tuple (A)
print (A [4:])
```

5.

```
A = (1, 2, 3, 4)
B = list(A)
B [0] = 100
B will be now?
```

‘Plus’ operator:

If you add two strings you will get joining part.

Ex: 2+3 = 5

“a” + “b” = ab

[5,6] + [7,8] = [5,6,7,8]

We can join two list using ‘+’ operator.

INPUT:	**OUTPUT:**

`A = [1,2,3,4,5]`
`B = [6,7,9]`
`C = A + B`
`print (C)`
	[1, 2, 3, 4, 5, 6, 7, 9]

Now, list can have integers, string and float. List can also have list.

In A = [1, 2, 4, 5, 6, “ABC”, [100, 200]]

So, A [6] is [100,200]

A [6][1] will be 100

A [6][2] will be 200

To add anything in the list,

A [6].append (300)

Then A = [1, 2, 4, 5, 6, “ABC”, [100, 200, 300]]

Problems:

1. given a string S and an integer n repeat each character of string S n times.

Eg:

INPUT: “add” n = 3

OUTPUT: aaaddddd

INPUT: “ham” n = 3

OUTPUT: “hhhaaammm”

A = [ABCD] for n = 3

AAABBBCCCDDD

for c in A: gives us every character of A

res = “” # it is empty.

re += c*n # I’m adding c*n value to my res

return res

INPUT	**OUTPUT:**

`def makeString(s, n):`
`  res = ""`
`  for c in s:`
`  res += c*n`
`  return res`
 	
`print (makeString("ABC", 5))`
	AAAAABBBBB

INPUT: “hello people this is me”
OUTPUT: “me is this people hello”

Understanding.

“Hello people this is me”

This is string and I can iterate it from index zero till the last character. Lets supposed my word is empty.

Word = “” → empty

But now I’m adding each letter in the string to my **word**. H, e, l, l, o. the moment I get to the space ( ) I’ll add the **word** *hello* to my string and make my **word** empty again.

Now the index is at p and we will continue the same. The moment I get to the space, I’ll add the **word** *people* to the list and make my **word** empty.

Once we are done with this, add whatever is at the end of the word to the list. With this understanding, let’s code it.

We need a **word** variable (word = “”). And I can say *for c in str*:

Word = “”

For c in str:

If c == “”

l.append (word)

word = ““

else:

word += c

INPUT:

```
def split_into_list(words):
    word = ""
    res = list()
    for c in words:
        if c == " ":
            res.append(word)
            word = ""
        else:
            word += c
    res.append(word)
    return res

def join_list_to_str(l):
    ans = ""
    for r in l:
        ans += r + " "
    return ans

res = split_into_list("hey boy this is a class")
res = res[::-1]
print(join_list_to_str(res))
```

OUPUT:

class a is this boy hey