***LISTS***

List is a sequence of values but they may or may not have a uniform value.

*Example*

factors = [1,2,5,10]

name = [“John”, “Robert”, “Albert”]

We can also have lists which might contain numbers, Boolean and a string

*Example*

mixed = [ 3, True, “John”]

List also has position [ 0, 1, 2, ……. n-1 ] for a list from [0,1,…… n ]

To extract values of a given position, slice like str

name [2] = “Albert”

mixed [1] = True

mixed [1:3] = [ True, “John”]

Length of list is given by function ***len( )***

*Example*

***len(mixed) = 3***

***Lists and Strings***

In a string if we take value at single position or we take substring of length we get the same results

h = “help”

h[0] = = h[0:1] == “h”

This is not same in the case of lists.

For lists, a *single position return a value,* a *slice returns as list*

factors = [1,2,5,10]

factors[0] = 1

factors[ 0:1] = **[ 1 ]**

***Nested List***

Lists can contain lists inside a list

*Example*

***nested = [ [2,[20]], 4, [“help”]]***

nested[0] = ***[2,[20]]***

nested[1] = 4

nested[2] = [‘help’]

nested[2][0][3] is “p”

nested[0][1:2] is [[20]] // *When we take slice of a list we get a list*

***Updating Lists***

Unlike strings, lists can be updated

***nested = [ [2,[20]], 4, [“help”]]***

nested[1] = 7

nested is now [[2,[20]], 7, [“help”]]

nested[0][1][0] = 10

nested is now [[2,[10], 7, [“help”]]

***This concludes that lists are mutable***

***Understanding Mutable vs Immutable***

What happens when we assign name?

x = 10

y = x

x = 20

*Will the value of y change to 20?*

*Let’s try this!*

**No!** Value of y didn't changed when we changed the value of x.

We should understand that when we assign a value to value of another name so are simply copying the value & making a fresh copy of it.

For immutable values of type int, float, bool, str updating one value doesn't affect the copy.

For mutable values, assignment does not make a fresh copy

list 1 = [1,3,5,7]

list 2 = list 1

list 1[2] = 9

now list 1 = [1,3,**9**,7]

What is list 2 now ?

List1 and List2 are two names for a same list

***Copying a List***

A slice creates a new sub-list from an old list

Recall

l[ :k] is [0:k]

l[k: ] is l[k : len( l )]

Omitting both end points gives a *full slice*

l [ : ] = = l[0:len ( l )]

## ***Digression on Equality***

Consider the following assignment

list1 = [1,3,5,7]

list2 = [1,3,5,7]

list3 = list2

All three lists are equal but there is a difference list1 and list2 are two different list with the same values whereas list3 is a copy of list2 or in other words it is different name for list2.

## ***Concatenating Lists***

Use **"+"** concatenation

*Example*

list1 = [1,3,5,7]

list2 = [2,4,6,8]

list3 = list1 + list2

"+" always produces a new list

List3 = [1,3,5,7,2,4,6,8]

List3 = list3 + [10]

List3 = [1,3,5,7,2,4,6,8, 10]

## *Accepting an Input from User*

***input( ) function*** is used to accept an input from the user. If the input function is called, the program flow will be stopped until the user has given an input and has ended the input with the return key.

The input of the user will be interpreted. If the user e.g. puts in an integer value, the input function returns this integer value. If the user on the other hand inputs a list, the function will return a list.

*Example*

```
name = input("What's your name? ")
print("Nice to meet you " + name + "!")
age = input("Your age? ")
print("So, you are already " + str(age) +
" years old, " + name + "!")
```

*Note: Value of age is stored as a **string***

**Exercise**

Difference between input ( ) and raw_input ( )

## ***Control Flow***

- Many a times we need to vary computational steps as value changes.
- Control flow determines the order in which the statement are to be executed

## ***Type of Control Flow***

- Conditional Execution
- Repeated Execution: Loops
- Function Definitions

## ***Conditional Execution***

*IF CONDITION*

If is used to make a decision depending upon a particular condition.

*Example*

*int will convert the age into integer and will save the value in age variable*

```
age = int(input("Please enter your age: "))
if age >= 18:
    print("Adult")
else:
    print("Not Adult")
```

## ***ELIF Condition***

By using elif we can check multiple conditions in our python code

*Example: Grading System*

```
marks = int(input("Enter Your Marks: "))
if marks >= 95:
    print("Grade - A+")
elif marks >= 90:
    print("Grade - A")
elif marks >= 80:
    print("Grade - B+")
elif marks >= 70:
    print("Grade - B")
elif marks <= 60:
    print  ("Grade-F")
```

## ***FOR LOOPS***

For loops can iterate over a sequence of numbers using the "range" and "xrange" functions.

Task: Read about xrange function

```
for x in range(5):
    print(x)

# Prints out 3,4,5
for x in range(3, 6):
    print(x)

# Prints out 3,5,7
for x in range(3, 8, 2):
    print(x)
```

*Example*

```
list_one = [1,2,3,4,5,6,7,8,9]
for eachnumber in list_one:
    print(eachnumber)
```

## ***WHILE LOOPS***

While loops repeat as long as a certain boolean condition is met.

```
# Prints out 0,1,2,3,4

count = 0
while count < 5:
    print(count)
    count += 1  # This is the same as
count = count + 1
```

## ***Infinite Loop***

```
while True:
    print("HaHa you stuck in an Infinite
Loop")
```

```
Press Ctrl+C to get rid of infinite loop
```

***Functions***

Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time.

```
def my_function():
    print("Hello From My Function!")

# Calling a Function
my_function()
```

Functions may also receive arguments (variables passed from the user to the function)

```
def my_function_with_args(username, greeting):
    print("Hello, %s , From My Function!, I
wish you %s"%(username, greeting))

my_function_with_args("Saurabh", "Best of
Luck")
```

*Example-2*

```
def addition(num1, num2):
    answer = num1 + num2
    print("Num1 is ", num1)
    print("Num2 is", num2)
    print("Sum is", answer)

addition(10,20)
```