2-10-2024

Training Day- 12

All variables in Python are of reference type. When we assign one variable to another variable in Python, the address of RHS variable is assigned to LHS variable. Generally the quality of ref type variable is that we can change the value of one variable the other variable should also get changed, but this effect is generally not directly visible in python why? Because Python supports dynamic memory allocation, whenever we assign a new value to a variable in python, it is stored at a new address.

How variables are created in Python: By assigning the value.
If assign int value, variable will become of int type
If assign str value, variable will become of str type
All variables are stored in RAM, the moment we stop our program, all the location get free ie values of variables are washed.

Python: Dynamic memory allocation:

How variables in created: By assigning the value.
Every time we assign a new value in Python, new addresses will
be allocated.
a=5
Steps:
1: Will check what is the data type of 5 and how much memory is needed to store 5, say 10 bytes are needed
2. That much memory (say 10 bytes) will be searched in RAM and will be blocked, say
1000 address onwards 10 bytes are blocked (say 1000 to 1009 locations are blocked to store 5)
3. 5 Will be stored at that location ie at say 1000 address onwards.
4. a will start referring to that location ie 1000 address.
or other locations.

```
b=a     #b address 1000, why all variables in python are of ref type
a=7     #a address 2000
print(a is b)   False
```

Function:

```
def function_name(arguments):
        set of statements
        return argument (Optional)
```

Formal and actual parameters can have same or different names

```
#BLL
# def len(a):
#     Block of code to calculate length
#     return length
# #PL
# s="CETPA"
# n=len(s)
# print(n)
```

Class is a collection of variables and functions(methods)
How to call a function, created inside class:
Syntax:
obj_name.method_name(arguments)

"""

```
# #New Program
# s="Cetpa"
# r=s.upper()     #Upper function is created inside str class
# print(r)


# #New Program
# L1=["ab","cd"]
# L2=L1.upper()       #AttributeError: 'list' object has no attribute 'upper'
# print(L2)
```

LOOPS IN PYTHON: Loops are the only hurdle for new programmers.

Loop: is used to execute block of code repeatedly.
To execute set of statements again and again.

In Python we have two types of loops:

- **for**
- **while**

- **For loop:** Few people say, for loop is of 2 types but I say, its of only 1 type but we can use same for loop in different ways.

Syntax:
for element in iterator:
   set of statements to execute repeatedly
How above loop works: Everytime loop runs, one element from left to right is retrieved from iterator and set of statements execute
How many times loop will run by default: ie no of elements present
                    in iterator

How loop works: Firstly loop statement run one time, then loop inner statements run one time, then again loop run one time and loop inner statements one time and so on.

iterators: str, list, tuple, dict, set, frozenset, range
"""

```
# #New Program
# L=[10,20,30,40]
# for e in L:
#     print(e)
#     print("ABC")
#     print(95)
```

Now few people who doesn't know the background of range class, they say, for loop is of 2 types, one directly used on iterator and other used on range class.

```
for element in iterator:
    set of statements

for i in range(arguments):
    set of statements
```

range class requires at least one argument, which is called the upper bound.
Further syntax:

```
range(a,n,s)    : a lower bound, n upper bound, s step size
range(a,n)      : a lower bound, n upper bound, default s = 1
range(n)        : n upper bound, default lower bound=0, default s = 1
```

```
# for e in L:
#     if(e%2==0):
#         print(e*e)


# #New Program
# s="10 20 30 40 50"
# L=s.split()
# print(L)
# L.append(60)
# print(L)
```

While Loop: Similar to for loop with different syntax
Python while loop is similar to C Lang while loop with minute syntax difference.
Syntax:

```
index initialize
while(condition):       #If condition is True, loop will run, else loop will stop
    set of statements
    increment or decrement index
```

```
for i in range(n):  #lower bound=0, upper bound=n, step size=1
    print(i)        #range(0,n,1)
```

while in place of for loop above, will be as follows

```
i=0     #i=lower bound
```

```
while(i<n):     #while(i<upper bound):
    set of statement
    i+=1        #i+=step_size
```

for loop:
1. for loop will work directly on any iterator.
2. for loop is having simple or less line of syntax

while loop:
1. used to execute infinite loop
2. Can also work on float lower, upper bounds or step size

……….New Program……….

```
# #BLL
# import math
# def add(a,b):
#     return a+b
# def sub(a,b):
#     return a-b
# def mul(a,b):
#     return a*b
# def div(a,b):
#     return a/b
# def pow(a,b):
#     return a**b
#
# #PL
# print("Welcome to my Calculator")
# while(1):
#     no1=int(input("Enter First No:"))
#     no2=int(input("Enter Second No:"))
#     choice=input("Enter Any operation +,-,*,/,pow,log,exit:")
#     if(choice=="+"):
#         res=add(no1,no2)
#         print("Result:",res)
#     elif(choice=="-"):
#         res = sub(no1, no2)
#         print("Result:", res)
#     elif(choice=="*"):
#         res = mul(no1, no2)
#         print("Result:", res)
#     elif(choice=="/"):
#         res = div(no1, no2)
#         print("Result:", res)
#     elif(choice=="pow"):
#         res = pow(no1, no2)
#         print("Result:", res)
#     elif(choice=="log"):
#         res = math.log(no1,no2)
#         print("Result:", res)
#    elif(choice=="exit"):
#        print("Thanks for using Harshit's Calci")
#        break
#    else:
#        print("Incorrect Choice")
```