1-10-2024

Training Day- 11

FUNCTIONS: A function is a reusable block of code that performs a specific task. Functions allow you to break your program into smaller, modular pieces, making the code easier to read, maintain, and reuse.

Defining a Function
In Python, functions are defined using the def keyword.
Syntax:
python
Copy code

```
def function_name(parameters):
    """Optional docstring"""
    # Function body
    return value  # Optional
```

* **FUNCTION_NAME:** The name of the function, following Python naming conventions.
* **PARAMETERS**: Input values passed to the function (optional).
* **RETURN:** Returns a value from the function (optional).

PL (Presentation Layer): Responsible for user interaction
BLL (Business Logic Layer): Responsible for writing the business logic

Camel Nomenclature: These are guidelines designed by Developers community to take logical identifier names.

1. Function Name should start with small letter and rest all trailing words should start with capital letters
Example:
add_Customer() or addNewCustomer()

2. Class Name should be in title case: First letter of all words should start with Capital Letter
MyCustomer   or M_Customer

Above 2 guidelines are as per camel nomenclature
rest other guidelines for identifier names:

GUIDELINES:

1. Try to use underscore in between words
2. Variable names should not have capital letter
3. Identifiers names should be logical names ie they should represent the purpose of identifier.

Ex: Variable names
cus_id, cus_age, cus_name, cus_mob, cus_email, cus_add, cus_city, cus_dob, cus_state, cus_gender......

FUNCTIONS:

The formal parameters and actual parameters names can be same or can be different as per the interest or requirement.

Whenever we call a function in any programming langauge, the actual parameters are assigned to formal parameters.

Now if we want a program for calculator then try to make it using 2 layers.
"""

> BLL(Business logic layer)

```
# def add(a,b):       #a,b formal parameters
#     r=a+b
#     return r
# def sub(a,b):
#     return a-b
```

> PL(presentation layer)

```
# a,b,c,d=1,2,3,4     #This statement can be a part of PL or BLL
# r1=add(a,b)
# r2=sub(c,d)
# print(r1,r2)
```

"""
In Python return statement is optional.
The functions which return nothing in the program then they return None value.
"""

```
# #New Program
# def func1(a,b):
#     r=a+b
#
# r=func1(5,7)
# print(r)

# #New Program
# s=input("Enter Your Name")

# #New Program
# s=print("CETPA")
# print(s)

# #New Program
# def add(a,b):
#     r=a+b
#     return r
# u,v=5,7
# s=add(u,v)      #a=u, b=v
# print(s)
"""u and v are actual variables in above program.
a and b are formal variable in above program
"""
```

# #New Program: Not a better approach to create a function

```
# def add():
#     r=a+b
#     return r
# a,b=5,7
# s=add()
# print(s)
```

# #New Program: Better approach

```
# def add(a,b):       #a,b,r are called local variables
#     r=a+b
#     return r
# a,b=5,7
# s=add(a,b)      #a,b,s are called global variables
# print(s)
```

"""
We can access global variables inside functions directly but
we can't access local variables outside functions.

…New Program….

```
# def func1():
#     a=5         #Local Variable
#     print(a)
# func1()
# print(a)        #NameError: name 'a' is not defined
```

…New Program….

```
# def func1():
#  print(a)        #a global variable
# a=5
# func1()
# print(a)            #a global variable
```

"""

If there are two variables with same name, one inside function and one outside function then inside function local variable will be accessed and outside function global variable will be accessed. In following program, why global variable is not modified, because in python, how variables are created? By assigning the value. If we have a variable outside function (Global variable) and we try to access it inside function then it is accessible, but the moment we try to modify the global variable inside function, what happens, a new local variable is created. Still if we want to modify global variable inside function.

…..New Progra,……

```
# def func1():
#     a=5           #
#     print(a)
# a=7
# func1()
# print(a)
```

….New Program….

```
# def func1():
#     global a
#     a=5           #
#     print(a)
# a=7
# print(a)
# func1()
# print(a)
```

…..New Program…..

```
# def func1():
#     a=5           #local variable
#     print(a)
#
# func1()
# print(a)        #NameError: name 'a' is not defined
```

àà..New Programàà.

```
# def func1():
#     global a
#     a=5           #global variable
#     print(a)
# func1()
# print(a)
```

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")
```

3-10-2024
Training Day - 13

```
# #New Program
# user_input = input("Enter the number with separated space:")    #"10 20 30 40 50"
# user_list = user_input.split() # convert into list, ["10","20"...
# print(user_list)
# for i in range(len(user_list)):     #i=0,1,2,3,4
# # convert each element into integer
#     user_list[i]= int(user_list[i])     #user_list[i]: "10"
# print("No are:",user_list)


# r =0
# for x in range(len(user_list)+1): #i=1, 2...7
# if(x==5):
# continue
# t=x**2 #t=1sq, 2sq # r=r+t #r=0+1sq=1sq, r=1sq+2sq, 1sq+2sq+...7sq


# #New Program
# L1=[10,20,30]       #index 0,1,2
# L2=[100,200,300]    #index 0,1,2
# L3=[30,40,50]       #index 0,1,2
# for i in range(len(L1)):      #range(3), i=0,1,2
#     print(L1[i]+L2[i]+L3[i])

# #New Program
# id_list=[10,20,30]                     #index=0,1,2
# name_list=["Vikas","Anil","Amit"]       #index=0,1,2
# age_list=[39,41,45]                     #index=0,1,2
# for i in range(len(id_list)):       #range(3), i=0,1,2
#     print("Cust ID:",id_list[i],"Cust Name:",name_list[i],"Cust Age:",age_list[i])
```

Check whether a number input by user is a Prime no or not?
""""

```
#BLL
def checkPrime(no):     #no=11
    for i in range(2,no):       #i=2, 3
        if(no%i==0):
            return "not Prime"
    return "Prime"
```

# #PL
```
# no=int(input("Enter any number:"))      #no=9
# res=checkPrime(no)
# print("The entered no is:",res)
```

# #Find all the primary and non-primary numbers in a given limit:
```
# no_low=int(input("Enter lower limit:"))      #7
# no_high=int(input("Enter higher limit:"))    #13
# for no in range(no_low,no_high+1):  #no=7 to 13
#     res=checkPrime(no)
#     print("The no",no,"is",res)
```

# #New Program: Check all Prime nos in a given list
```
# #BLL
# def checkAllPrime(no_low,no_high):
#     L=[]
#     for no in range(no_low,no_high+1):        #no=7,8,9,...13
#         for i in range(2, no):  # i=2, 3,4,5,6, no=7
#             if (no % i == 0):
#                 break
#         else:
#             L.append(no)        #L=[7,11]
#     return L
#
# #PL
# no_low=int(input("Enter lower limit:"))      #7
# no_high=int(input("Enter higher limit:"))    #13
# res=checkAllPrime(no_low,no_high)
# print("The Prime Nos are:",res)
```

```
*
**
***
****
*****
```
Step 1: No of lines:5, so exteranl loop will run for 5 times

n=5

for i in range(5):

Step 2:

Table:

i	n	j_range
0	5	1
1	5	2
2	5	3

```
3   5       4
4   5       5
no Python, maths starts:
Find relation between j_range and i and n:
j_range=i+1


"""
# #New Program
# for i in range(5):          #i=0, i=1
#     for j in range(4):      #i=0,j=0,1,2,3  i=1,j=0,1,2,3
#         print("*",end="")
#     print()

# #New Program
# for i in range(5):          #i=0, i=1,...4
#     for j in range(i+1):
#         print("*",end="")
#     print()
```

4-10-2024 Training Day - 14

PROGRAM ON STRING OPERATION AND LIST METHODS

……….New Program……

```
# L=[]
# e=10
# L.append(e)
# print(L)
# e=20
# L.append(e)
# print(L)
```

……….New Program……

```
# s="cetpa"
# r=s.upper()
# print(r)
```

……….New Program……

```
# L1=[10,20,30]
# L2=L1.append(40)
# print(L2, L1)


# # #New Program
# s="cetpa infotech"
# r=s.replace("e","*")
# print(r)
```

LIST METHODS

……….New Program……

```
# L=[10,20,30]
# print(id(L))
# L.append(40)
# print(L)
# print(id(L))
```

……….New Program……

```
# L=[10,20,30]
# print(id(L))
# L[0]=100
# print(L)
# print(id(L))

# #New Programm
# L=[10,20,30]
# L.extend(40)       #Error: Extend method works on iterator
# print(L)

# #New Program
# L=[10,20,30]
# r=L.pop(1)      #index is optional to pass
# print(L)
# print(r)

# #New Program
# L=[10,20,30,40,50,60]
# i=2
# L.pop(i)
# print(L)

# #New Program
# L=[10,20,30,20,40]
# ele=20
# r=L.count(ele)
# print(r)
# print(L)

# #New Program
# L=[10,20,30]
# print(id(L))
# L1=L.copy()
# print(id(L1))
# print(L,L1)
```

7-10-2024

Training Day - 15

Dictionary:

1. Dictionary is a collection of heterogeneous data types.
2. Dictionary is a collection of key-value pairs. One key-value pair is called one item.
3. Dictionary is mutable in nature
4. Dictionary can have only unique keys ie can't have duplicate keys.
5. Dictionary is a collection of unordered items. Dictionary elements don't have direct index.

Syntax:
{comma separated key-value pairs}
dict_var={key1:value1,key2:value2,key3:value3....}

Syntax to access elements of a dictionary:
dict_var[key]

```
# #New Program
# d={1:10,2:50,"CETPA":80,90:"ABC",50:[2,3,4]}
# print(d[1])
# print(d["CETPA"])
# print(d[50])

# #New Program: Aditi says if duplicate keys are there
# d={1:10,2:20,3:30,2:50,4:40,2:80}
# print(d)

# #New Program
# d={1:10,2:20,3:30}      #d address 1000
# print(d,id(d))
# d[2]=80                 #d address 1000
# print(d,id(d))
```

Benefit of using dictionary:

In real life, in almost all cases, we are not aware about the index of a data rather we are aware about the actual data elements. Now if data is stored in a list or tuple, and if we want to find the index of a particular element and in case the element is far away from starting or end point, then it takes a lot of time to search the element in a big data. But the better approach can be, we can store the data in a dictionary and can make the unique values of data as a key like customer id, employee id, student roll no etc. And now if we are aware about the key, we can immediately access the data element.

```
cus_dict={10:["Vikas",39,9212468020],20:["Anil",41,9654444252],...}
id=20
print(cus_dict[id])
```

In dictionary, the keys are first converted to hash codes,

```
List
L=[10,20,30,40]
""""
# L=[10,20,30,40,50,60,70]      #40 index
```