8-10-2024

Training Day - 16

CATEGORIES OF FUNCTIONS:

1. Required Argument Functions: Where the no of arguments and sequence of arguments should match in formal and actual. Till now all functions we created, were required argument type
2. Keyword Argument Functions: Where the no of arguments should be same in formal and actual but there position can vary.
3. Default Argument Functions: Where we assign a default value to a variable. Now while calling, if we pass the value in actual then new values is used otherwise default value is used.
4. Variable Length Argument Functions (Tuple Based): In formal parameter, we add one extra star immediately before variable name. Python recommends that the formal variable name should be 'args' in case of variable length argument functions
5. Variable Length Keyword Argument Functions (Dictionary Based): In formal parameter, we add two extra stars immediately before variable name. Python recommends that the formal variable name should be 'kwargs' in case of variable length keyword argument functions
6. Lambda Functions: Anonymous Functions: Lambda functions are single liner anonymous functions. lambda arguments_passed:expression_calculated&returned

"""

1. Formal Arguments

```
# #New Program
# def add(a,b):
#     return a+b
#
# u,v,w=5,7,9
# s1=add(u,v,w)      #Error
# print(s1)
```

2. Keyword Arguments #addCustomer(id=id,name=name...)

```
# def sub(a,b):
#     return a-b
# u,v=5,7        #u-v
# r=sub(b=v,a=u)
# print(r)
```

3. Default Arguments

```
# def add(a=1,b=2):
#     return a+b
# r1=add()       #r1=3
# r2=add(5)      #r2=7
# r3=add(b=9)    #r3=10
# r4=add(5,7) #r4=12
# r5=add(b=8,a=2) #r5=10
# print(r1,r2,r3,r4,r5)
```

4. Variable Length Argument Functions (Tuple Based):

```
# #New Program
# def func1(*t):      #Formal Variable name is t
#     print(t)
# func1(2,3,4)
# func1(10,20,30,40,50,60)
# func1()
```

FUNCTION POINTER: Function pointer is a variable which holds the address of a function.

```
def func1():
    pass
a=func1()
```

here a is a function pointer which is holding the address of a function.

lambda arguments_passed:expression_calculated&returned

```
# print(lambda a,b:a+b)
```

```
# #New Program
# a=lambda a,b:a+b        #a is a funciton pointer
# r=a(5,7)
# print(r)
```

```
#New Program
add=lambda a,b:a+b        #a is a funciton pointer
r=add(5,7)
print(r)
```

9-10-2024 # Training Day – 17

OOPS: Object Oriented Programming: In python we have concept of classes and OOPS base is Class

Class: Is a collection of variables and functions. In OOPS or in class, generally the functions are called methods.

Python Supports Modular Approach Of Programming: We can develop any big project without OOPS also but complexity of the project will be increased a lot. Using OOPS, we can decrease the complexity of the project, the project is made scalable using OOPS, we can easily modify and enhance the project later using OOPS.

Syntax to create a class:

```
class Class_Name:
   variables and methods ie block of code
.
```

How to create object of any class:

```
obj_name=class_name(arguments)
obj_name=class_name()
```

```
# #New Program
# class C1:
#     pass
# obj=C1()       #obj is object of C1 Class
# print(type(obj))
# print(obj)
```

If we are calling a method which is created inside a class, then while calling from actual parameters, first object is passed to formal parameters and then rest of the arguments are passed.
"""

```
# #New Program
# d1={1:10,2:20,3:30}     #d1 is of dict type
# d2={11:100,12:200,13:300}
# res=d1.values()     #If values is a function values(d1)
# print(res)
```

When we call a method which is inside a class then from formal parameters object is passed to the first argument of actual parameters. Python recommends that the object name in formal parameters should be self.

```
# #New Program
# L1=[1,2,3]
# L1.append(5)
```

Class represents real time entity.

One of the benefits of using OOPS is that we can easily represent real time entities using classes

Syntax to call a method of a class:
obj_name.method_name(arguments)

How to access an instance variable:
obj_name.variable_name

```
# #New Program
# class C1:
#     pass
# obj1=C1()
# obj1.a=5    #a is a variable of obj1 object
# print(obj1.a)
# obj1.b=7
# obj1.c=9
# print(obj1.a,obj1.b,obj1.c)
# obj2=C1()
# obj2.a=20
# obj2.b=30
# print(obj1.a,obj1.b,obj1.c,obj2.a,obj2.b)
```

10-10-2024 # Training Day – 18

Create A Generalized Function which takes any no of arguments and return the multiplication of all arguments using variable length keyword argument function.

```
# #BLL
# def mul(*args):     #args=(2, 3, 4, 5, 6)
#     r=1
#     for i in range(len(args)):
#         r=r*args[i]
#     return r
# #PL
# r1=mul(2,3,4,5,6)
# r2=mul(1,2)
# print(r1,r2)

"""
Create your own generalized index function. Print all the matching index
positions of an element present in a list [2,3,4,5,6,2,3,4,2,3,4,2]. Take the
element as input from the user.
"""
# #New Program
# L=[2,3,4,5,6,2,3,4,2,3,4,2]
# ele=2
# for i in range(len(L)):     #i=0,1,2,...n-1
#     if(L[i]==ele):
#         print(i)
```

Constructor: is a method, which is called automatically everytime we create an object in python. In Python the name of the constructor is fixed ie __init__()

```
"""
# #New Program
# class C1:
#     def __init__(self):     #Constructor
#         print("CETPA")
#
# ob1=C1()
# ob2=C1()
# ob3=C1()
```

Generally in programming in real world, the variables of all objects of a class are common like all customers will have same variables like id, name, age, mob so we mostly create the variables inside constructor.

```
# class Customer:
#     def __init__(self):  #self=1000, self=2000
#         self.id=0       #1000.id=0, 2000.id=0
#         self.name=0     #1000.name=0
#         self.age=0      #1000.age=0
#         self.mob=0      #1000.mob=0
# cus1=Customer()      #cus1 1000, self 1000
# print(cus1.id,cus1.name,cus1.age,cus1.mob)
# cus2=Customer()     #cus2 2000 , self 2000
# print(cus2.id,cus2.name,cus2.age,cus2.mob)
```

Now class or static variables and methods. These variables or method are like normal variables or functions which we have studied outside class.

How To Create Static Variables:

Same syntax like outside class. Directly inside class, assign
the value

var_name=value

How To Access Static Variables: using class name

class_name.var_name

Static variables will be common variables

11-10-2024

Training Day – 19

Multiple Inheritance: One Child Multiple Parents

```
# class C1:
#     def __init__(self):
#         self.a=1
#         self.b=2
#     # def showData(self):
#     #     print("I am in Class C1")
# class C2:
#     def __init__(self):
#         self.c=3
#         self.d=4
#     # def showData(self):
#     #     print("I am in Class C2")
# class C3:
#     def __init__(self):
#         self.e=5
#         self.f=6
#     def showData(self):
#         print("I am in Class C3")
# class C4(C1,C2,C3):
#     def __init__(self):
#         self.g=7
#         self.h=8
#         super().__init__()
#         C2.__init__(self)
#         C3.__init__(self)
#     # def showData(self):
#     #     print("I am in Class C4")
# obj=C4()      #object of C4. obj.__init__()
# obj.showData()
# print(obj.a,obj.b,obj.c,obj.d,obj.e,obj.f,obj.g,obj.h)


"""
```

In above program, if we have showData method in all 3 parents but not in child. Now if we call showData method using child object then it may create an ambiguity or in Java it creates ambiguity that which parent showData method will be called? So this ambiguity in Java is called Diamond Problem and there is no direct solution made for diamond problem in Java ie Java doesn't support Multiple Inheritance. But in Python this a solution is made for this ambiguity using Priority setting and making MRO (Method Resolution Order) If there are multiple parents of one child ie C1, C2, C3 then direct priority will be given from left to right as we mention parents name in child class.

Now in above program as we have multiple parents for one child
so to create variables of all parents, we need to call constructor
of all the parents in child class.

# Hybrid Inheritance:

```
# Case 1 as shared in painting:
# """
# class C1:
#     def __init__(self):
#         self.a=1
#         self.b=2
#     def showData(self):
#         print("I am in Class C1")
# class C2(C1):
#     def __init__(self):
#         self.c=3
#         self.d=4
#         super().__init__()
#     # def showData(self):
#     #     print("I am in Class C2")
# class C3(C1):
#     def __init__(self):
#         self.e=5
#         self.f=6
#         super().__init__()
#     # def showData(self):
#     #     print("I am in Class C3")
# class C4(C2,C3):
#     def __init__(self):
#         self.g=7
#         self.h=8
#         super().__init__()
#         C3.__init__(self)
#     # def showData(self):
#     #     print("I am in Class C4")
# obj=C4()
# print(obj.a,obj.b,obj.c,obj.d,obj.e,obj.f,obj.g,obj.h)
# obj.showData()
```

Polymorphism:

Overloading: is also called compile time polymorphism

Overriding: is also called run time polymorphism

Real definition of access specifiers as per OOPS concepts used in C++ or Java or other:
Public: Can be accessed anywhere in itself or child or outside class
Protected: Can be accessed only in itself or in child class
Private: Can be accessed only in itself.
.

14-10-2024 # Training Day – 20

Exception Handling It is possible to write programs that handle selected exceptions. Look at the following example, which asks the user for input until a valid integer has been entered, but allows the user to interrupt the program (using Control-C or whatever the operating system supports); note that a user-generated interruption is signalled by raising
the KeyboardInterrupt exception

```
# class Count:
#     object_count = 0
#     def __init__(self):
#         Count.object_count+= 1
#     # classmethod
#     def my_object_count(cls):
#         return cls.object_count
# #PL
# obj1 =Count()
# obj2 =Count()
# obj3 =Count()
# obj4 =Count()
# print(Count.object_count)

# #New Program
# x=5            #x address 1000
# class x:       #x address 2000
#     pass
# print(x)
```

Exception Handling: How to handle exceptions (errors) In real life projects, if we get errors while program execution then program doesn't stop working or doesn't throw the error at the output rather some error message is given and projects keep on running.

The target of exception handling is: Catching/Managing the errors while program execution and try not to stop the program.

```
""""
# #New Program
# id_list=[10,20,30,40]
# id=int(input("Enter the ID:"))     #id=50
# i=id_list.index(id)
# print(i)
```

# import Module12   #ModuleNotFoundError: No module named 'Module12'

To handle exceptions in the program there are 4 keywords designed
for exception handling.
try:
except:

finally:
raise:

```
"""
# #New Program
# x=int(input("Enter First No:"))     #ValueError: invalid literal for int() with base 10: 'cetpa'
# y=int(input("Enter Second No:"))
# res=x/y                     #ZeroDivisionError: division by zero
# print("Result:",res)

# #New Program
# while(1):
#     try:
#         x=int(input("Enter First No:"))     #ValueError: invalid literal for int() with base 10:
'cetpa'
#         y=int(input("Enter Second No:"))
#         res=x/y                     #ZeroDivisionError: division by zero
#         print("Result:",res)
#         break
#     except:
#         print("Error!")
```

RAISE KEYWORD:
To raise ie to throw the errors/exceptions intentionally
 in the program

Syntax:
raise ErrorClass_Name(Message for user)

```
"""
# raise ValueError("Error!")


# #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
# while (1):
#     try:
#         no1 = int(input("Enter First No:"))
#        no2 = int(input("Enter Second No:"))
#        choice = input("Enter Any operation +,-,*,/,pow,log:")
#        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)
#        else:
#            raise NotImplementedError("Incorrect Choice")
#    except Exception as err:
#        print("Error!",err)
#        print(type(err))
```