Python-Class & Object

INT213

Introduction

- Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy.
- This chapter helps you become an expert in using Python's object-oriented programming support.

How to create a Class in Python

- There are following terms which you need to know while working with classes in Python
  1. The “class” keyword
  2. The instance attributes
  3. The class attributes
  4. The “self” keyword
  5. The “__init_” method

1. The “Class” Keyword

With the class keyword, we can create a Python class as shown in the example below.

```
class BookStore:
    pass
```

2. The Instance Attributes

These are object-specific attributes defined as parameters to the **__init__** method. Each object can have different values for themselves.

In the below example, the “attrib1” and “attrib2” are the instance attributes.

Receives the instance of class automatically

```
class BookStore:
        def __init__(self, attrib1, attrib2):
                self.attrib1 = attrib1
                self.attrib2 = attrib2
```

Overview of OOP Terminology

- **Class: A** user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.

Overview of OOP Terminology

- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Instance:** An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation:** The creation of an instance of a class.
- **Method :** A special kind of function that is defined in a class definition.
- **Object:** A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.

Overview of OOP Terminology

- **Inheritance:** The transfer of the characteristics of a class to other classes that are derived from it.
- **Function overloading:** The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Operator overloading:** The assignment of more than one function to a particular operator.

Example 1

```
#class definition
class employee:
    def __init__(self): # constructor
        self.name="abc"
        self.salary=10
    def disp(self): # member function
        print("Name=",self.name)
        print("Salary=",self.salary)
emp1=employee() # object creation
emp1.disp() # member function calling
#Access by object
print(emp1.name)
```

Example 2

- The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows −

```
#class definition
class employee:
   ec=0 # employee count
   def __init__(self,n,s): # constructor with arguments
      self.name=n # data member
      self.salary=s #data member
      employee.ec+=1 # accessing class member
   def disp(self): # member function
      print("Name=",self.name)
      print("Salary=",self.salary)
   def empcount(self): # second member function
      print("Total employee: = %d"%employee.ec)
emp1=employee("SUN",2000) # object creation
emp1.disp() # member function calling
#Access by object
print(emp1.name)
print(emp1.salary)
emp2=employee("MOON",1000)
emp3=employee("STAR",10000)
emp1.empcount()
```

Creating Instance Objects

- "This would create first object of Employee class"

  emp1 = employee("Zara", 2000)

- "This would create second object of Employee class“

  emp2 = employee("Manni", 5000)

Accessing Attributes

- The **getattr(obj, name[, default])** : to access the attribute of object.
- The **hasattr(obj,name)** : to check if an attribute exists or not.
- The **setattr(obj,name,value)** : to set an attribute. If attribute does not exist, then it would be created.
- The **delattr(obj, name)** : to delete an attribute.

Example 3

```
class employee:
    def __init__(self,n,s):
        self.name=n
        self.salary=s
    def disp(self):
        print("Name=",self.name)
        print("Salary=",self.salary)

emp1=employee("SUN",2000)
print(hasattr(emp1,"name")) # True
print(hasattr(emp1,"age")) # False
print(getattr(emp1,"name")) # SUN
#ERROR
#print(getattr(emp1,"age"))
setattr(emp1,"name","Moon") # value updated
emp1.disp()
delattr(emp1,"salary")
print(emp1.name)
#ERROR
print(emp1.salary)
```

Public and private member of class

```
class employee:
    def __init__(self,n,s,a):
        self.name=n #public
        self.salary=s #public
        self.__age=a # Private
    def disp(self):
        print("Name=",self.name)
        print("Salary=",self.salary)
        print("Age=",self.__age) # can be access

emp1=employee("SUN",2000,35)
emp1.disp()
print(emp1.name)
print(emp1.salary)
# access private member directly
print(emp1._employee__age)
#can't access directly
print(emp1.__age) # ERROR
```

Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute –

- **__dict__:** Dictionary containing the class's namespace.
- **__doc__:** Class documentation string or none, if undefined.
- **__name__:** Class name.
- **__module__:** Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- **__bases__:** A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Example 5

```
class employee:
    'Class emloyee having name and age'
    def __init__(self,n,a):
        self.name=n
        self.age=a
    def disp(self):
        print("Name=",self.name,"Age=",self.age)
#class documentation
print("employee.__doc__:",employee.__doc__)
#class name
print("employee.__name__:",employee.__name__)
#module name in which class is defined
print("employee.__module__:",employee.__module__)
#base classess
print("employee.__bases__:",employee.__bases__)
# dictionary containing class name space
print("employee.__dict__:",employee.__dict__)
```

Destroying Objects (Garbage Collection)

- Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space.
- The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.

Example 6

```
class Point:
    def __init__( self, x=0, y=0): #constructor
        self.x = x
        self.y = y

    def __del__(self): #distructor
        class_name = self.__class__.__name__
        print (class_name, "destroyed")
    def show(self):
        print("x=",self.x,"y=",self.y)
pt1 = Point()
pt2 = pt1
# prints the ids of the obejcts
print (id(pt1))
print (id(pt2))
#delete object
del pt1
#pt1.show()
pt2.show()
del pt2 # distructor call
```

Questions

1. Write a class name point for creating 2D points through constructor.
2. Read and print details of a student using class programming having attributes name, roll no, reg no, percentage.
3. Write a class name time, having attributes hours,minutes, seconds. Define a member function to add two times.
4. Write a class name threeD, having attributes x, y and z. Define a member function dist to calculate Euclidian distance of two threeD points.