Date
July 22, 2021

classmate
Date
Page

WEEK 8

Introduction to Object Oriented Programming

Python is a multi-paradigm programming language. It supports different programming approaches.

One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming.

An object has two characteristics:
→ attributes
→ behaviour

Let's take an example:

A parrot is an object, as it has the following properties:
→ name, age, colour as attributes
→ singing, dancing as behaviour

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't repeat yourself)

WHY OOP ?

Primitive data structures - like numbers, strings, and lists - are designed to represent simple pieces of information, such as the cost of an apple, the name of a poem, or your favourite colors, respectively. What if you want to represent something more complex ?

For example, let's say you want to track employees in an organization. You need to store some basic information about each employee, such as their name, age, position, and the year they started working.

One way to do this is to represent each employee as a list :

```
a = ["James", 34, "Captain", 2265]
b = ["Matt", 35, "Science Officer", 2254]
c = ["Leonard", 40, "Chief Medical Officer", 2266]
```

There are no. of issues with this approach.

First, it can make larger code files more difficult to manage. If you reference a[0], several lines away from where the a list is declared, will you remember that the element with index 0 is the employees names?

Second, it can introduce errors if not every employee has the same no. of elements in the list.

A great way to make this type of code more manageable and more maintainable is to use classes.

CLASSES vs INSTANCE

Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

A class is a blueprint for how something should be defined. It doesn't actually contain any data.

While the class is a blueprint, an instance is an object that is built from a class and contains real data.

In other words,

A class is like a form or questionnaire. An instance is like a form that has been filled out with information. Just like many people can fill out the same form with their own unique info., many instances can be created from a single class.

→ Below is a simple Python program that creates a class with a single method.

CODE:

```
class Test :

    def fun ( self ) :
        print ("Hello")

obj = Test( )                    # creating object
obj. fun ( )                     # calling class method via object
```

OUTPUT: Hello

→ As we can see above, we create a new class using the class statement and the name of the class. This is followed by an indented block of statements that form the body of the class. In this case, we have defined a single method in the class.

→ Next, we create an object (instance) of this class using the name of the class followed by a pair of parentheses.

(1) The 'self':

1. Class methods must have an extra first parameter in the method definition. We do not give a value

for this parameter when we call the method, Python provides it.

2. If we have a method that takes no arguments, then we still have to have one argument - the self.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) - this is all the special self is about.

(2) The '__init__' method

→ The __init__ method is similar to constructors in C++. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

```
# A sample class with init method
class Person:

    #init method or constructor
    def __init__(self, name):
        self.name = name

    #sample method
    def say_hi(self):
        print('Hello, my name is', self.name)
```

```
#creating object
p = Person ('Jack')
p.say_hi ()
```

Output : Hello, my name is Jack

→ Here we define the __init__ method as taking a parameter name (along with the usual self).

(3) Class and Instance Variables (or Attributes)

→ In Python, instance variables are variables whose value is assigned inside a constructor or method with self.

→ Class variables are variables whose value is assigned in class.

CODE:

```
# Python program to show that the variables with a
# value assigned in class declaration, are class
# variables and variables inside methods and
# constructors are instance variables.

class CSStudent :
    stream = 'cse'                    # class variable

    def __init__(self, roll):
        self.roll = roll              # instance variable
```

```
# Objects of CSstudent class
a = CSStudent (101)
b = CSstudent (102)

print (a.stream)
print (b.stream)
print (a.roll)

# class variable can be accessed using class name also
print (CSstudent.stream)
```

OUTPUT :
```
cse
cse
101
cse
```

→ We can define instance variables inside normal methods also.

CODE :
```
# program to show that we can create instance variables
# inside methods

class CSStudent :
    stream = 'cse' # class variable

    def __init__ (self, roll) :
        self.roll = roll # instance variable

    def setAddress (self, address) :
        self.address = address # instance variable
```

```
    # Retrieves instance variable
    def getAddress (self):
        return self.address

# main code
a = CSStudent (101)
a.setAddress ("Noida, UP")
print ( a.getAddress ())
```

OUTPUT : Noida, UP

(4) How to create an empty class ?

We can create an empty class using pass statement in Python.

```
class Sample:
    pass
```

Date
July 26, 2021

Inheritance And Method Over-riding

Inheritance is the process by which one class takes on the attributes and methods of another. Newly formed classes are called child classes, and the classes that child classes are derived from are called parent classes.

Child classes can override or extend the attributes & methods of parent classes. In other words, child classes inherit all of the parent's attributes and methods but can also specify attributes and methods that are unique to themselves.

Although the analogy isn't perfect, you can think of object inheritance sort of like genetic inheritance.

You may have inherited your hair color from your mother. It's an attribute you were born with. Let's say you decide to color your hair purple. Assuming your mother doesn't have purple hair, you've just overridden the hair colour attribute that you inherited from your mom.

You also inherit, in a sense, your language from your parents. If your parents speak English, then

you'll also speak English. Now imagine you decide to learn a second language, like German. In this case you've extended your attributes because you've added an attribute that your parents don't have.

Let's see a code for better understanding:

```
#parent class
class Bird:

    def __init__ (self):
        print ('Bird is ready')

    def whoisThis (self):
        print ('Bird')

    def swim (self):
        print ('Swim Faster')

#child class
class Penguin (Bird):

    def __init__ (self):
        #call super function
        super().__init__()
        print ("Penguin is ready")

    def whoisThis (self):
        print ('Penguin')
```

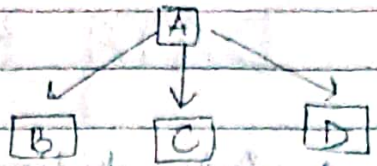
```
    def run(self):
        print('Run faster')

peggy = Penguin()
peggy.whoIsThis()
peggy.swim()
peggy.run()
```

OUTPUT

```
Bird is ready
Penguin is ready
Penguin
Swim faster
Run faster
```

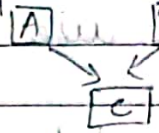
5. Hierarchical Inheritance



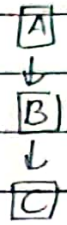
TYPES OF INHERITANCE

1. Single Inheritance

2. Multiple Inheritance.



3. Multilevel Inheritance



4. Hybrid Inheritance
(mixture of all)

→ In the above program, we created two classes, i.e, Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from the swim() method

→ Again, the child class modified the behavior of the parent class. We can see this from whoisThis() method. Furthermore, we extend the functions of the parent class, by creating a new run() method.

→ Additionally, we use the super() function inside the __init__() method. This allows us to run the __init__() method of the parent class inside the child class.

DATA HIDING

→ Data hiding means protecting the members of a class from an illegal or unauthorized access.

→ Data hiding concern about data security along with hiding complexity.

→ Data hiding focuses on restricting or permitting the use of data inside the capsule.

→ The data under data hiding is always pvt. and inaccessible.

let's see a code :

```
class MyClass :

    #hidden member of myclass
    __hiddenVariable = 0

        # member method that changes __hiddenVariable
        def add (self, increment):
            self.__hiddenVariable += increment
            print (self.__hiddenVariable)

obj = MyClass()
obj.add(2)
obj.add(5)
print (obj.__hiddenVariable)
```

OUTPUT :

```
2
7
Traceback (most recent call last):
    File "filename.py", line 14, in
        print (obj.__hiddenVariable)
AttributeError: 'Myclass instance has no attribute
'__hiddenVariable'
```

→ In the above program, we can see that we can hide a variable by using double underscore (__) before the attributes name and those attribute will not be directly visible outside.

→ Here, we tried access hidden variable outside the class using object and it throws an error.

→ A private variable and methods of a class can be accessed inside the class and inside a class method of the same class.

→ We can't access put attributes & methods in the child / subclasses or outside the class directly.

Introduction to NumPy

Comparison Between List & Numpy Array

PARAMETERS	PYTHON LIST	NUMPY ARRAY
Installation and importing	Not required	Required
Type of Elements	Heterogeneous	Homogeneous
Dimension of Ele.	No restriction	Has to be same
Memory allocation	Non-contiguous	Contiguous
Size	Requires more space	Requires less space
Performance	Slower	Faster
Element Wise Operations	Not possible	Possible
Functionality	Cannot handle arithmetic operations	Can handle arithmetic operations

WHAT IS NUMPY ?

Numpy is one of the most popular packages in Python.

Numpy is used for scientific computing.

Numpy is used for working with arrays.

Numpy provides arrays, which are great alternatives to traditional lists.

Numpy arrays are much faster than Python lists.

NUMPY ARRAYS

Numpy arrays are like the better version of Python lists.

Arrays are much faster than lists, and are easier to work with.

CREATING ARRAY

To create an array, use the array() method.

```
import numpy as np

arr = np.array([1, 2, 3, 4])
print(arr)
```

OUTPUT

```
[1, 2, 3, 4]
```

The ndarray Object

The object that gets created when we use the array() method is called ndarray.

This can be shown by checking the type of the object using the type() function.

CODE:

```
import numpy as np

arr = np.array([1,2,3])
x = type(arr)

print(x)
```

OUTPUT:

```
<class 'numpy.ndarray'>
```

Elements In An Array

→ The objects inside an array are called elements.

→ A std. numpy array is required to have elements of the same data type.

CODE:

```
import numpy as np

x = np.array([1,2,3])          # all elements are numbers
y = np.array(['hi','hello'])   # all elements are strings

print(x)
print(y)
```

OUTPUT: [1, 2, 3]
['hi', 'hello']

To Find the Dimension of An Array (ndim)

CODE:

```
import numpy as np
a = np.array(42)
b = np.array([1,2,3])
c = np.array([[1,2,3],[4,5,6]])
d = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])

print(a, a.ndim, '\n')
print(b, b.ndim, '\n')
print(c, c.ndim, '\n')
print(d, d.ndim, '\n')
```

OUTPUT:

```
42  0

[1, 2, 3]  1

[[1, 2, 3]
 [4, 5, 6]] 2

[[[1, 2, 3]
  [4, 5, 6]]
 [[1, 2, 3]
  [4, 5, 6]]] 3
```

Introduction to Matplotlib library

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi platform data visualisation library built on NumPy arrays and designed to work with the broader SciPy stack.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestable visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Importing matplotlib

```
from matplotlib import pyplot as plt
```

OR

```
import matplotlib.pyplot as plt
```

BASIC PLOTS IN MATPLOTLIB

Matplotlib comes with a wide variety of plots. Plots help to understand trends, patterns, and to make correlations. They're typically instruments for reasoning about quantitative information.

Let us look some plots:

(1) Scatter plot

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x,y)

plt.show()
```

(2) Bar Chart

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])

plt.bar(x, y)          #function to plot bar (vertical)

plt.show()             #function to show plot

plt.barh(x, y)         #function to plot horizontal bar
plt.show()
```

3. Histogram

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

(4) Pie Chart

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
l = ['Apples', 'Bananas', 'Cherries', 'Dates']

plt.pie(y, labels = l, startangle = 90)
plt.show()
```

SUBPLOTS

CODE:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array ([0,1,2,3])
y = np.arrby ([3,8,1,10])

plt.subplot (2,3,1)   # There are total 2 rows, 3 columns and
plt.plot (x,y)        # this plot is in the first plot

x = np.array ([0,1,2,3])
y = np.array ([10,20,30,40])

plt.subplot (2,3,2)
plt.plot (x,y)
x = np.array ([0,1,2,3])
y = np.array ([3,8,1,10])

plt.subplot (2,3,3)
plt.plot (x,y)
```

rows, columns, index of plot

```
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x, y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x, y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x, y)

plt.show()
```