Module-4

OBJECT-ORIENTED PROGRAMMING

PA N P

Topics

Classes and Objects

Creating Classes in Python

Creating Objects in Python

The Constructor Method

Classes with Multiple Objects

Class Attributes versus Data Attributes

Encapsulation

Inheritance

The Polymorphism.

Classes and Objects

- In python, everything is an object
- We can write a class to represent properties (attributes) and actions (behaviour) of object. Properties can be represented by variables, Actions can be represented by Methods.
- Ex:

```
l=[1,2,3,4]
l.append(5)
print(l)
print(type(l))
```

```
[1, 2, 3, 4, 5]
<class 'list'>
```

- student info
  - Class
  - Object
  - Reference variable

Class, Object, Reference variable

- **Class** is a blueprint to construct objects, it defines all properties and operations of object

- **Object** is physical existence of a class or real world entity. Memory will be allocated for object. Per class any number of objects can be created

  referencevariable = classname()

- **Reference variable** is pointing to our object, by using it we can perform operations on the object.

- In this example, 'l' is reference variable

```
l=[1,2,3,4]
l.append(5)
print(l)
print(type(l))
```

```
[1, 2, 3, 4, 5]
<class 'list'>
```

Creating Classes in Python

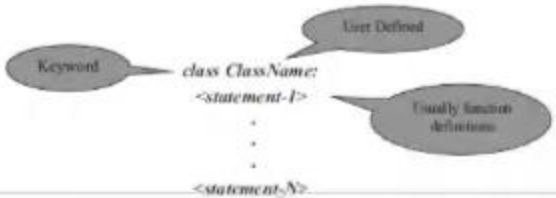
- Class contains both variables , methods

class className:

''' documenttation string '''

variables: instance variables, static and local variables(properties)

methods: instance methods, static methods, class methods(actions/ behaviour)



Class

- class classname:
  - ''' docstring'''
  - Variables(properties)
  - Methods(actions/behaviour)

Variables:

- Instance variables
- Static variables
- Local variables

Methods:

- Instance methods
- Static methods
- Class methods

Creating Objects in Python

- Object refers to a particular instance of a class where the object contains variables and methods defined in the class.
- Class objects support two kinds of operations:
  - attribute references and
  - instantiation.
- The term attribute refers to any name (variables or methods) following a dot. This is a syntactic construct.
- The act of creating an object from a class is called *instantiation*.

  The names in a class are referenced by objects and are called *attribute references*.

Example

```
class student:
    '''student information'''
    def __init__(self):
        self.name='Manu'
        self.age=40
        self.marks=80
    def studinfo(self):
        print("Hello I am :",self.name)
        print("My Age is:",self.age)
        print("My Marks are:",self.marks)
s=student()
s.studinfo()
#print(s.name)
```

Example

```
class Student:
    def __init__(self,Name,USN,CIE):
        self.name=Name
        self.rollno=USN
        self.marks=CIE

    def studinfo(self):
        print("Student Name is:",self.name)
        print("Student USN is:",self.rollno)
        print("Student Marks are:",self.marks)

s1=Student("Manu",56,45)
s1.studinfo()
s2=Student("vinu",61,24)
s2.studinfo()
s3=Student("vani",108,48)
s3.studinfo()
s4=Student("Manu",150,30)
s4.studinfo()
```

Self:

- Within the class to refer current object, python provides an implicit variable which is 'self'
- Self variable always pointing to current object
- Self variable is the first argument for constructor and instance methods
- By using self we can declare instance variables, we can access instance variables and instance methods of object
- We can use self within the class only and from outside of class we cannot use
- At the time of calling constructor and instance methods we are not required to provide value for self variable. PVM itself is responsible to provide value
- Instead of 'self' we can use any name , but recommended to use self