Classes and Objects

Python Classes and Objects

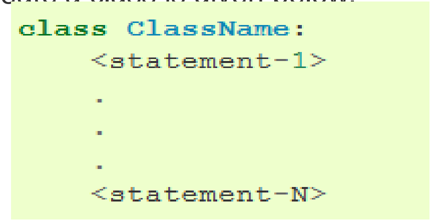
- Python is an object oriented programming language.
- Python supports the concept of objects and classes.

Class in Python

- Classes provide a means of bundling data and functionality together.
- The class is a user-defined data type that binds the data members and methods into a single unit.
- A class is a blueprint or template of entities (things) of the same kind.
- Using a class, you can create as many objects as you want.
- Creating a new class creates a new type of object, allowing new instances of that type to be made.
- Each class instance can have attributes attached to it for maintaining its state.
- Class instances can also have methods (defined by its class) for modifying its state.

Class Definition Syntax

- A class can be created by using the keyword class, followed by the class name.
- The syntax to create a class is given below.



```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

- ClassName represents the name of the class and statements represents the attributes and methods.
- Class definitions, like function definitions (def statements) must be executed before they have any effect
- When a class definition is entered, a new namespace is created, and used as the local scope — thus, all assignments to local variables go into this new namespace.

Objects in Python

- Object: An object is an instance of a class. Objects get their variables and functions from classes.
- An object is a collection of data (variables) and methods (functions).
- An object is a datatype that stores data, but also has operations defined to act on the data.
- Objects have two characteristics: They have states and behaviors (object has attributes and methods attached to it) Attributes represent its state, and methods represent its behavior.
- Example: A Circle drawn on the screen:
  - Has attributes (knows stuff): radius, center, color
  - Has methods (can do stuff): move, change color
- Objects have individuality, and multiple names (in multiple scopes) can be bound to the same object.

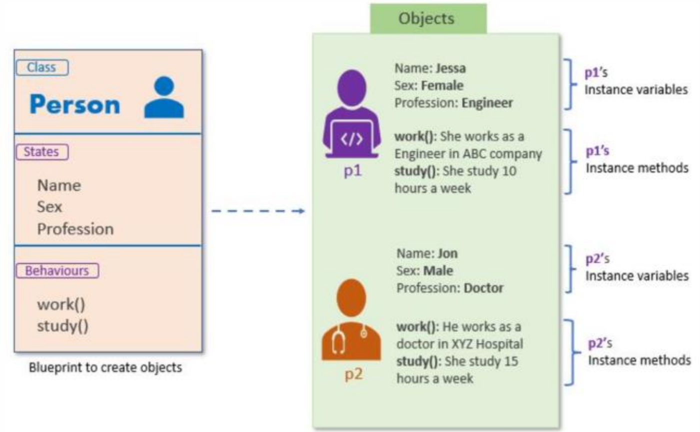
Design of circle object

- The syntax to create an object is

  objectName=ClassName(arguments)

- A Circle object:
  - center, which remembers the center point of the circle,
  - radius, which stores the length of the circle's radius.
  - color, which stores the color
  - The draw method examines the center and radius to decide which part of the paper should be colored.
  - The move method sets the center to another location, and redraws the circle.
- All objects are said to be an **instance** of some **class.** The class of an object determines which attributes the object will have.
- A class is a description of what its instances will know and do.

Circle: classes and objects



understand class and objects in Python

Class Object

- Class objects support two kinds of operations: attribute references and instantiation.
- Attribute references use the standard syntax used for all attribute references in Python: obj.name.
- Valid attribute names are all the names that were in the class's namespace when the class object was created.

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

Class Object

- Class instantiation uses function notation.
- The class object is a parameterless function that returns a new instance of the class.
- Example: x = MyClass() → it creates a new instance of the class and assigns this object to the local variable x.
- The instantiation operation ("calling" a class object) creates an empty object.
- Many classes like to create objects with instances customized to a specific initial state.
- Therefore a class may define a special method named __init__().

```
def __init__(self):
    self.data = []
```

- When a class defines an __init__() method, class instantiation automatically invokes __init__() for the newly-created class instance.

Class object

- Example:

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

x = Complex(3.0, -4.5)
x.r, x.i
```

Class object

- Example:

```
class student:
   def __init__(self,n,a):
      self.full_name = n
      self.age = a

s1=student(“XYZ",25)
s2=student(“ABC",32)
print(s1.age)
print(s2.age)
```

Circle: Init function and Self Parameter

Class functions that begin with double underscore __ are called special functions.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , it can be called it whatever you like, but it has to be the first parameter of any function in the class.

Write a Python code to calculate the area of a room:

```
class room:
    length=int(input("Enter the length"))
    breadth=int(input("Enter the breadth"))
    def calculatearea(self):
        area=self.length*self.breadth
        print("Area of the room is:",area)
ab208=room()
ab208.calculatearea()
```

Write a Python code to calculate the area of a room:

```
class room:
    length=int(input("Enter the length"))
    breadth=int(input("Enter the breadth"))
    def calculatearea(self):
        area=self.length*self.breadth
        print("Area of the room is:",area)
Area=room()
Area.calculatearea()
a2=room()
a2.length=45
a2.breadth=50
```

Circle: Accessing Functions

Ex: Create a class defined for vehicles. Create two new vehicles called car1 and car2. Set car1 to be a red convertible worth Rs.40 lakh with a name of Ferrari, and car2 to be a blue SUV named Jeep worth Rs.25 lakh.

```
class vehicle:
   name=""
   color=""
   type=""
   price=0.00
   def description(self):
      str="%s is a %s %s worth Rs.%.2f" %(self.name,self.color,self.type,self.price)
      return str
car1=vehicle()
car1.name = "Ferrari"
car1.color = "Red"
car1.type = "Car"
car1.price = 4000000
car2 = vehicle()
car2.name = "Jeep"
car2.color = "Blue"
car2.type = "SUV"
car2.price = 2500000
print(car1.description())
print(car2.description())
```

Student: Accessing Functions

```
class Student:
    course = "CSE3041"
    def __init__(self, name, test1=0, test2=0):
      self.name = name
      self.test1 = test1
      self.test2 = test2
    def compute_average(self):
      return (self.test1 + self.test2)/2
    def print_data(self):
        print(self.compute_average())
David = Student("David",90,100)
Bob = Student("Bob",60,80)
David.print_data()
Bob.print_data()
```

List of objects

Create a student class with name and age as attributes. Then create a list of students and display their names in increasing order of age.

List of objects

```
class student:
   counter=0
   def __init__(self,n,a):
      self.full_name = n
      self.age = a
      student.counter=student.counter+1

s=[]
m=int(input("Enter no of students"))
for i in range(0,m):
   name=input("Enter the name ")
   age=int(input("Age:"))
   s.append(student(name,age))
print(s)
for i in range(0,m):
   print("Name "+s[i].full_name ,"   Age: ",s[i].age)
```

```
for i in range(len(s)):
        for j in range(i+1,len(s)):
            if s[i].age > s[j].age:
                s[j],s[i]=s[i],s[j]

print("After Sorting")
for i in range(0,m):
    print("Name "+s[i].full_name,"   Age:
",s[i].age)
```

Python Scopes and Namespaces

- In Class definitions it is necessary to know how scopes and namespaces work.
- A namespace is a mapping from names to objects.
- Examples of namespaces are:
  - The set of built-in names → abs()
  - The global names in a module
  - The local names in a function invocation
- In the same sense the set of attributes of an object also form a namespace.
- There is absolutely no relation between names in different namespaces.
- Example: two different modules may both define a function ”maximize” without confusion

Python Scopes and Namespaces

- There is a straightforward mapping between the module's attributes and the global names defined in the module: they share the same namespace.
- Namespaces are created at different moments and have different lifetimes.
- The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted.
- The global namespace for a module is created when the module definition is read in; normally, module namespaces also last until the interpreter quits.
- The local namespace for a function is created when the function is called, and deleted when the function returns or raises an exception that is not handled within the function.
- A scope is a textual region of a Python program where a namespace is directly accessible.
- "Directly accessible" here means that an unqualified reference to a name attempts to find the name in the namespace.

Functions, Modules and Packages

Functions, modules and packages are all constructs in Python that promote code modularization.

Modules in Python are just Python files with a .py extension. The name of the module is the same as the file name. A Python module can have a set of functions, classes, or variables defined and implemented.

```
def displayMsg(name):
    print("Hi "+name)
```

Save this func in file.py

Import the file.py in another file.

```
import file
file.displayMsg("sample")
s=input("Enter name")
file.displayMsg(s)
```

calculation.py:

```
#place the code in the calculation.py
def summation(a,b):
    return a+b
def multiplication(a,b):
    return a*b;
def divide(a,b):
    return a/b;
```

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py

a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b))

 '''we do not need to specify the module name while
accessing summation()'''
```

The from...import statement is always better to use if we know the attributes to be imported from the module in advance. It doesn't let our code to be heavier. We can also import all the attributes from a module by using *.

Consider the following syntax.

```
from <module> import *
```

Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in our python source file.

The syntax to rename a module is given below.

```
import <module-name> as <specific-name>
```

Example:

```
'''the module calculation of previous example is imported in
this example as cal.'''
import calculation as cal;
a = int(input("Enter a?"));
b = int(input("Enter b?"));
print("Sum = ",cal.summation(a,b))
```

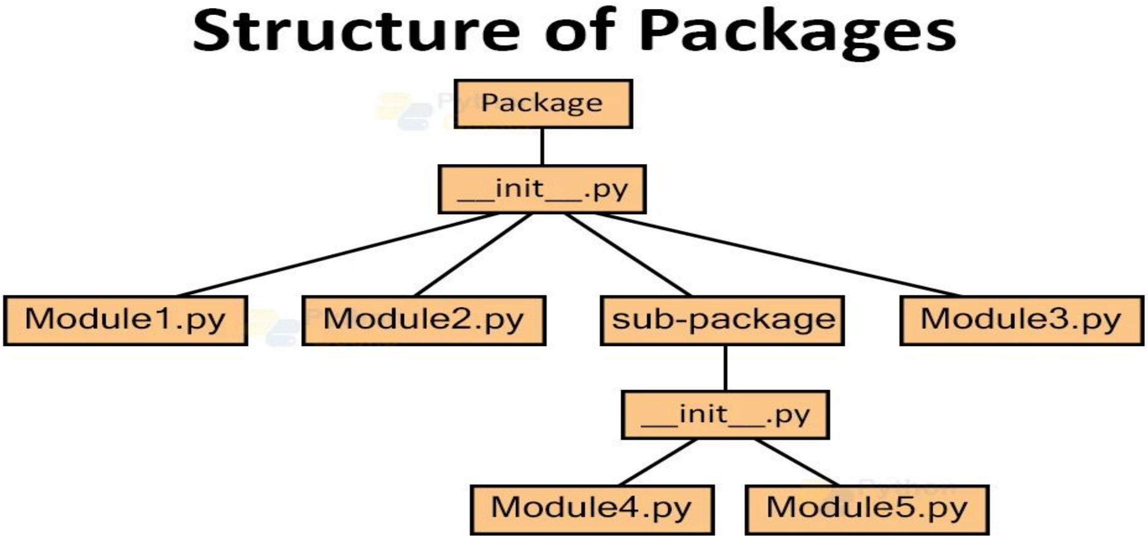
Output:

```
Enter a?10
Enter b?20
Sum =  30
```

Python Packages

Package is a folder/container that contains various modules/functions as files to perform specific tasks.
Python modules may contain several classes, functions, variables, etc. whereas
Python packages contain several modules and sub-packages.
A simple Python package contains multiple modules and an __init__.py file.
The __init__.py file makes an ordinary directory into a Python package.
For ex. the math package includes the sqrt() function to perform the square root of a number.

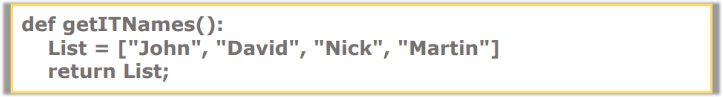
Structure of a Package



Let's create a package named Employees in your home directory. Consider the following steps.

1. **Create a directory with name Employees on path /home.**
2. **Create a python source file with name ITEmployees.py on the path /home/Employees.**

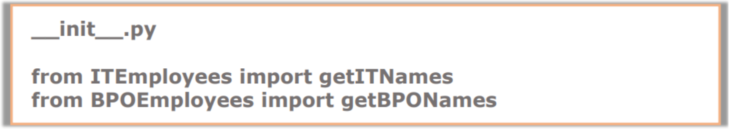
ITEmployees.py



```
def getITNames():
    List = ["John", "David", "Nick", "Martin"]
    return List;
```

3. Similarly, create one more python file with name BPOEmployees.py and create a function getBPONames().

4. Now, the directory Employees which we have created in the first step contains two python modules. To make this directory a package, we need to include one more file here, that is __init__.py which contains the import statements of the modules defined in this directory.

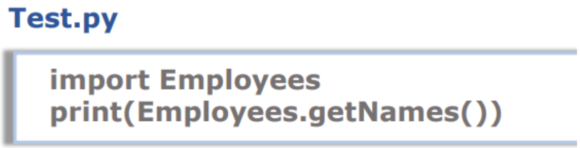


```
__init__.py

from ITEmployees import getITNames
from BPOEmployees import getBPONames
```

5. Now, the directory Employees has become the package containing two python modules. Here we must notice that we must have to create __init__.py inside a directory to convert this directory to a package.

6. To use the modules defined inside the package Employees, we must have to import this in our python source file. Let's create a simple python source file at our home directory (/home) which uses the modules defined in this package.


Test.py

```
import Employees
print(Employees.getNames())
```

Thank you