

UNIT IV-PYTHON OOP

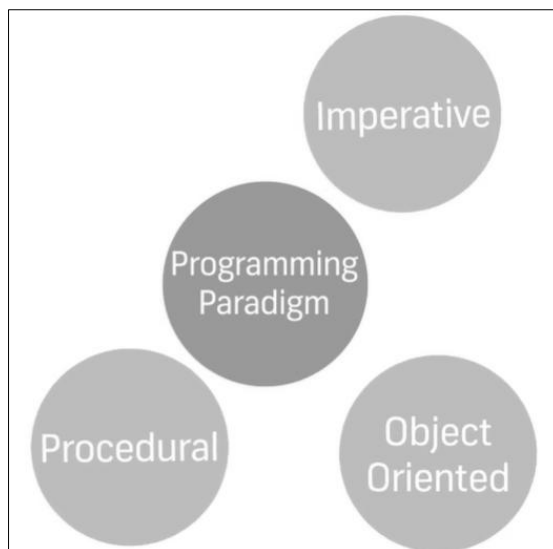
HOW OBJECT-ORIENTED PROGRAMMING WORKS	4
WHAT ARE THE CLASSES?	5
ESSENTIAL FEATURES OF OOP:	7
DATA ENCAPSULATION.....	7
INHERITANCE:.....	7
OVERRIDING:.....	8
POLYMORPHISM:	9
DECLARING CLASSES AND OBJECTS:	9
CLASS ATTRIBUTES AND CLASS METHODS:.....	12
DATA HIDING IN PYTHON.....	15
INHERITANCE:.....	16
OVERRIDING:.....	20
METHOD OVERLOADING:	23
METHOD OVERRIDING:.....	24
DIFFERENCE BETWEEN METHOD OVERLOADING AND METHOD OVERRIDING:25	

UNIT-IV

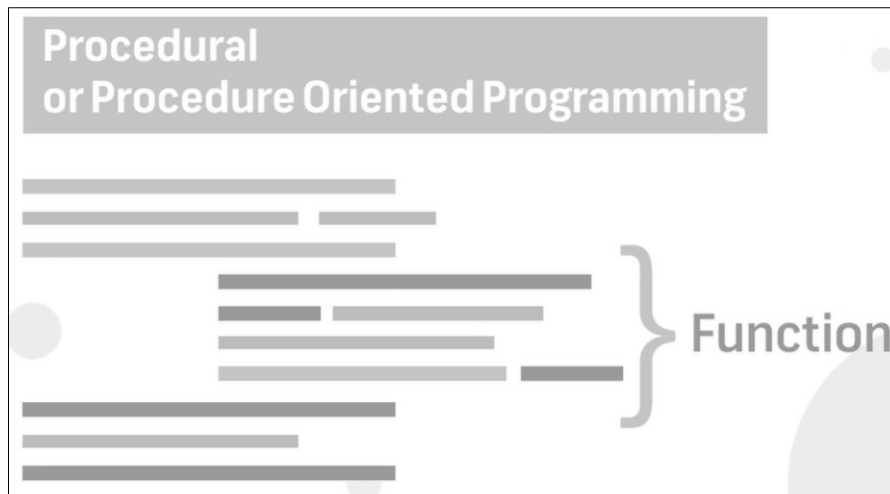
PYTHON OOP

Unit- IV Python OOP	4a. Create class and object for given problem. 4b. Write python code for data hiding. 4c. Write a program to use inheritance.	4.1 Creating classes and objects. 4.2 Data Hiding 4.3 Method overloading and overriding. 4.4 Inheritance
------------------------------------	---	---

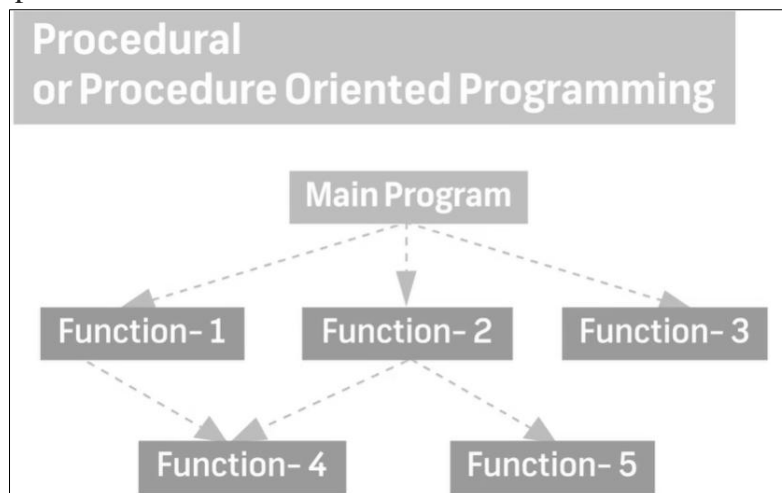
Programming languages are classified into various paradigms like **procedural, object oriented, imperative** etc. Programming paradigms are nothing but a way to classify programming languages based on the features.



One language can be classified into multiple paradigms for example python supports multiple programming paradigms including imperative, procedural, object oriented and functional programming. In this topic we will mainly focus on object oriented paradigm and learn a bit about procedural paradigm. Let's look at the procedural approach first. In procedural or procedure oriented programming to build a program step-by-step instructions are written in the logical order. If the program is complex, the logic is broken down into smaller but independent and reusable blocks of statements called functions.



It somewhat looks like this you have your main program. It has some functions in it. These main functions then would rely on some other functions and so on. This approach seems to be a sound one, but as this scope of the program becomes larger and complex you start noticing some prominent problems.



Problem No-1

Procedure oriented languages make the program difficult to maintain if you change something in one function it will effect all other functions that rely on that function. Thus making it harder for the programmer to make changes.

Problem No-2

Procedure oriented languages use a lot of global data items and this results in increased memory overhead. This happens especially in big and complex problems with large number of functions.

Problem No-3

Procedure approach gives more importance to process and doesn't consider data of same importance and takes it for granted.

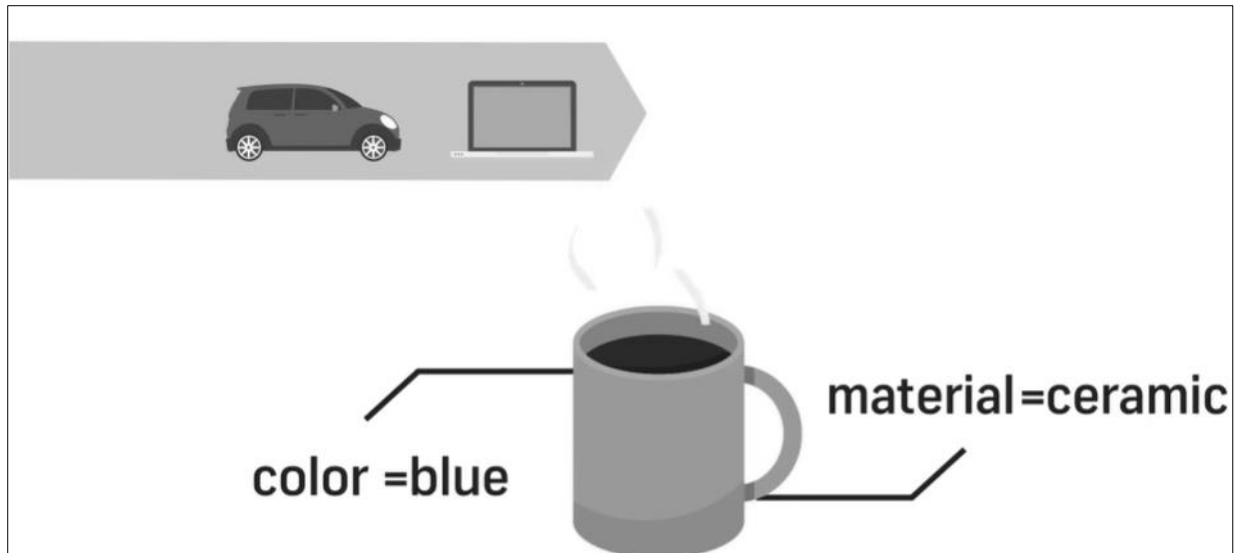
Problem No-4

Movement of data across functions is unrestricted as long as a number and type of arguments match, data from any function can be sent to any other function.

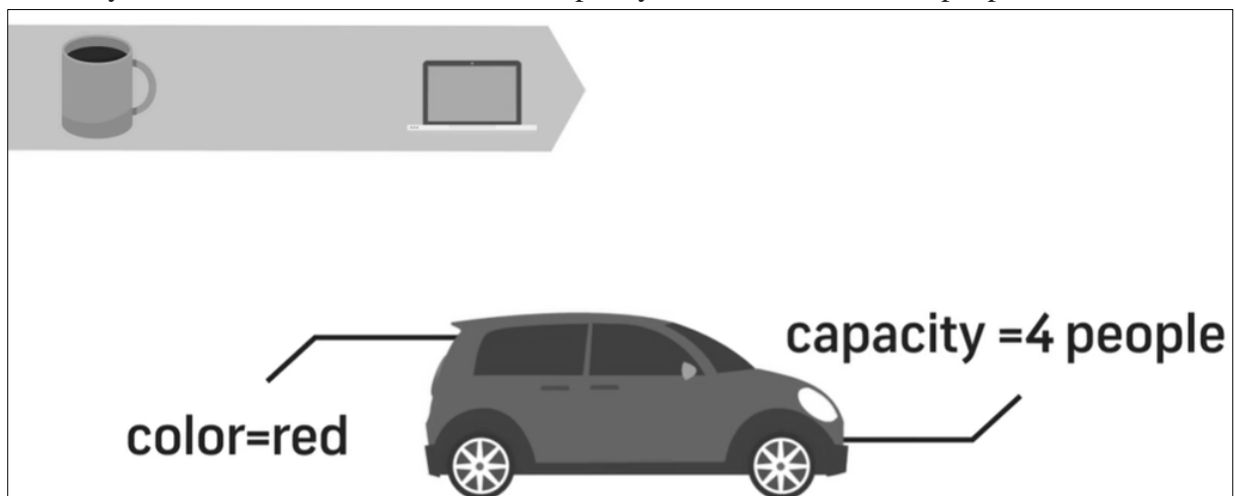
Now we know how procedural programming works what are its drawbacks.

HOW OBJECT-ORIENTED PROGRAMMING WORKS

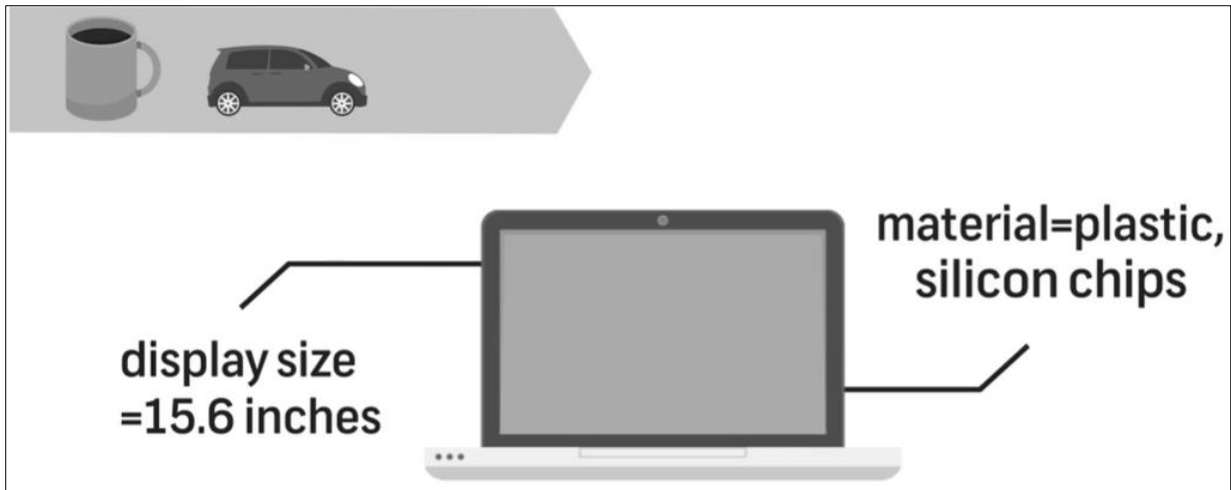
In object oriented programming, we build a program into parts or objects. To understand it better, let's look at it this way. In real world we come across different objects like a coffee mug, car, laptop etc. All these objects have some attributes like coffee mug will have a colour and it would be made up of some material and these attributes will have some values like its colour is blue and it is made up of ceramic.



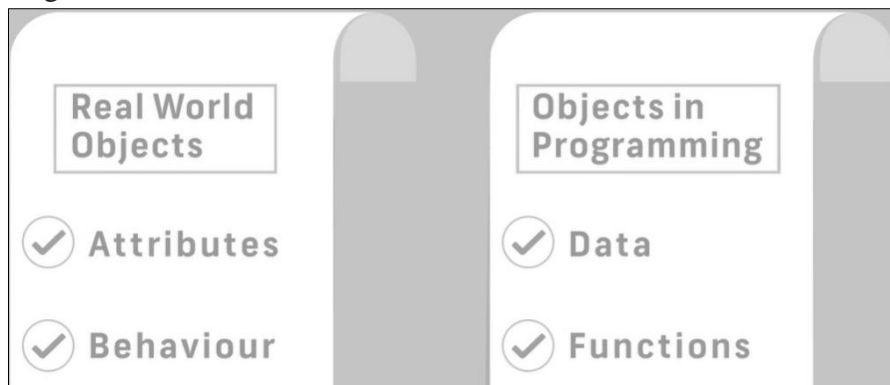
Similarly, Car is red in colour and it has a capacity to accommodate four people.



The laptop has 15.6 inches display and it is made up of plastic and silicon chips



On the other hand all these objects have some behaviour too. The coffee mug can hold coffee, the car take you from point A to point B , the laptop can send email. On top of this you can also combine different objects to perform different tasks for example your phone is an object and your earphones can be considered a separate object. By combining both of these together you can listen to music individually without disturbing others. Object oriented programming tries to mimic this real world behaviour. In it we can create different objects to perform different tasks in a program. Just like real-world objects and attributes and behaviour, objects in programming have data and functions.



Attributes are equivalent to data and behaviour is equivalent to functions. We create these objects using classes

WHAT ARE THE CLASSES?

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. 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.

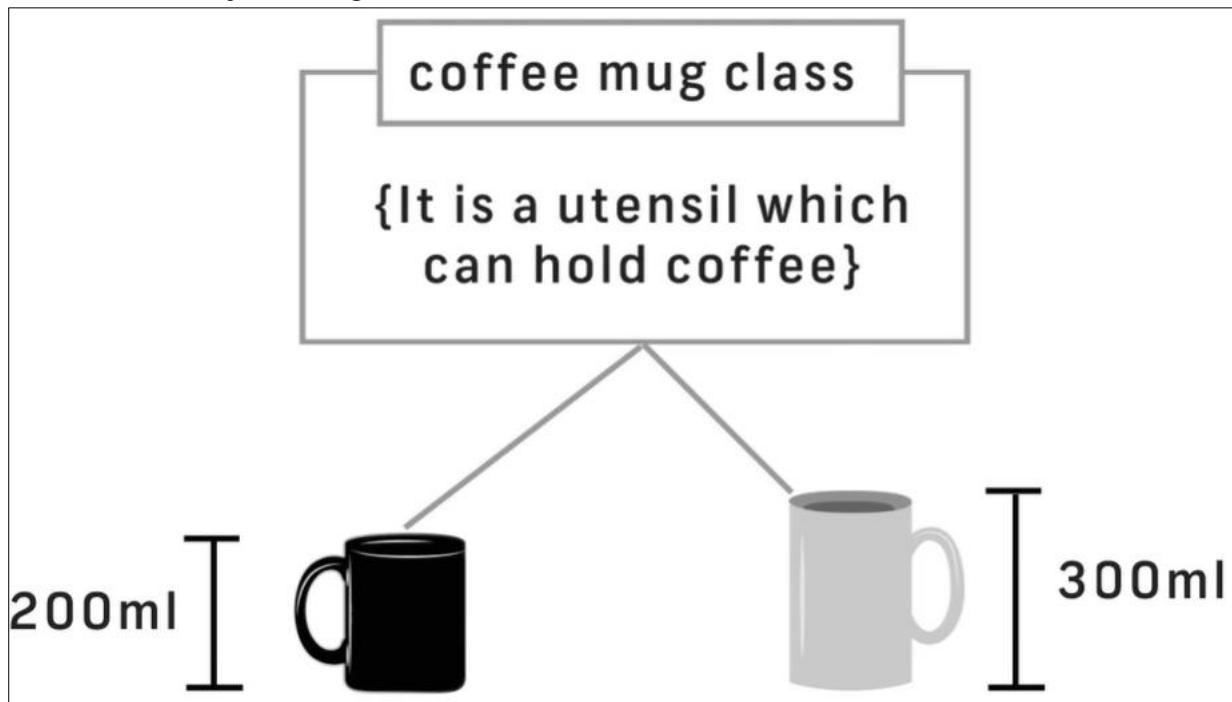
Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator.
- Eg.: Myclass.Myattribute

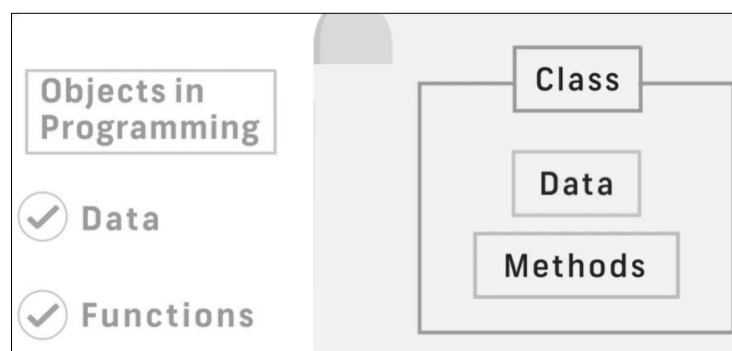
Class Definition Syntax:

```
class ClassName:
    # Statement-1
    .
    .
    .
    # Statement-N
```

You can think of class as a general idea of a category of an object. If you take an example of a coffee mug, a class will state that a coffee mug is a utensil which can hold coffee and give you a broad idea of what a coffee mug is. Using this class you can create specific coffee mugs like black coffee mug which can hold 200 ML of coffee and it will have a curved handle. You can also create another coffee mug which is yellow in colour and can hold 300 ML of coffee. this mug will keep your coffee hot as it won't let the heat go away. So this is how we create objects using a class.



Previously we learnt that objects in object oriented programming have data and functions. Since objects are made from classes, classes also have data and functions in it. But the functions inside a class are referred to as methods.

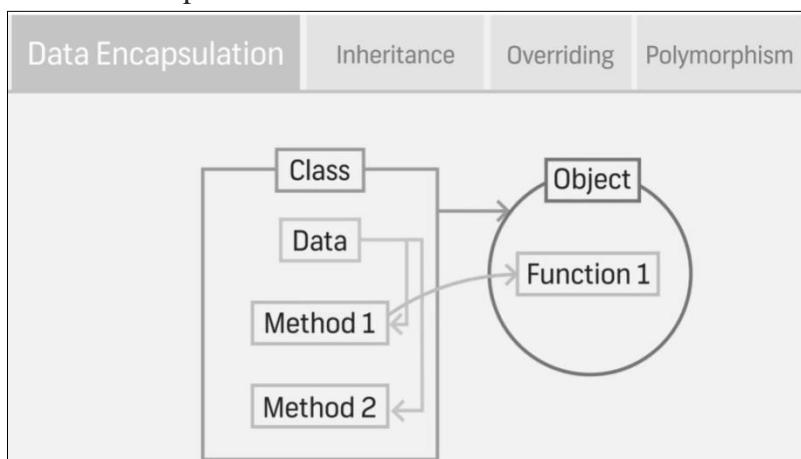


ESSENTIAL FEATURES OF OOP:

We learnt about object oriented programming classes and objects. Now let's broadly look at some of the essential features of object oriented programming. The features that we go through are **Data Encapsulation, Inheritance, overriding and polymorphism**.

DATA ENCAPSULATION:

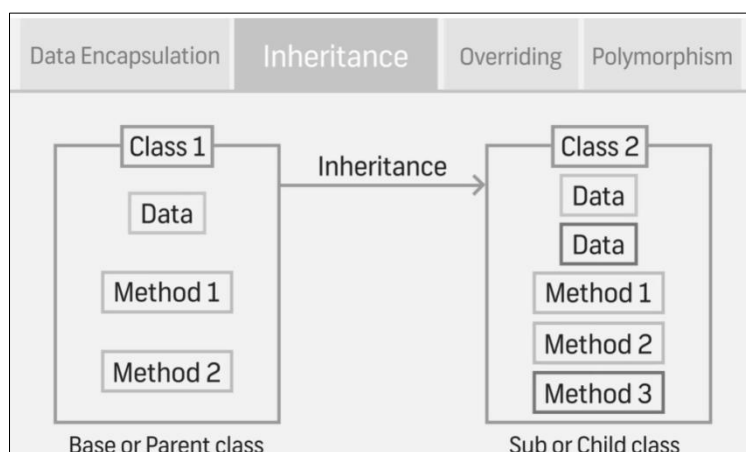
It is the feature that binds together the data and functions that manipulate the data, this keeps both data and methods of a class or object safe from outside interference and misuse. If this is a class, its data would be available to its method only and to use these methods we will have to create an object from this class and then call a method. Encapsulating this data and method, to make a class is called encapsulation.



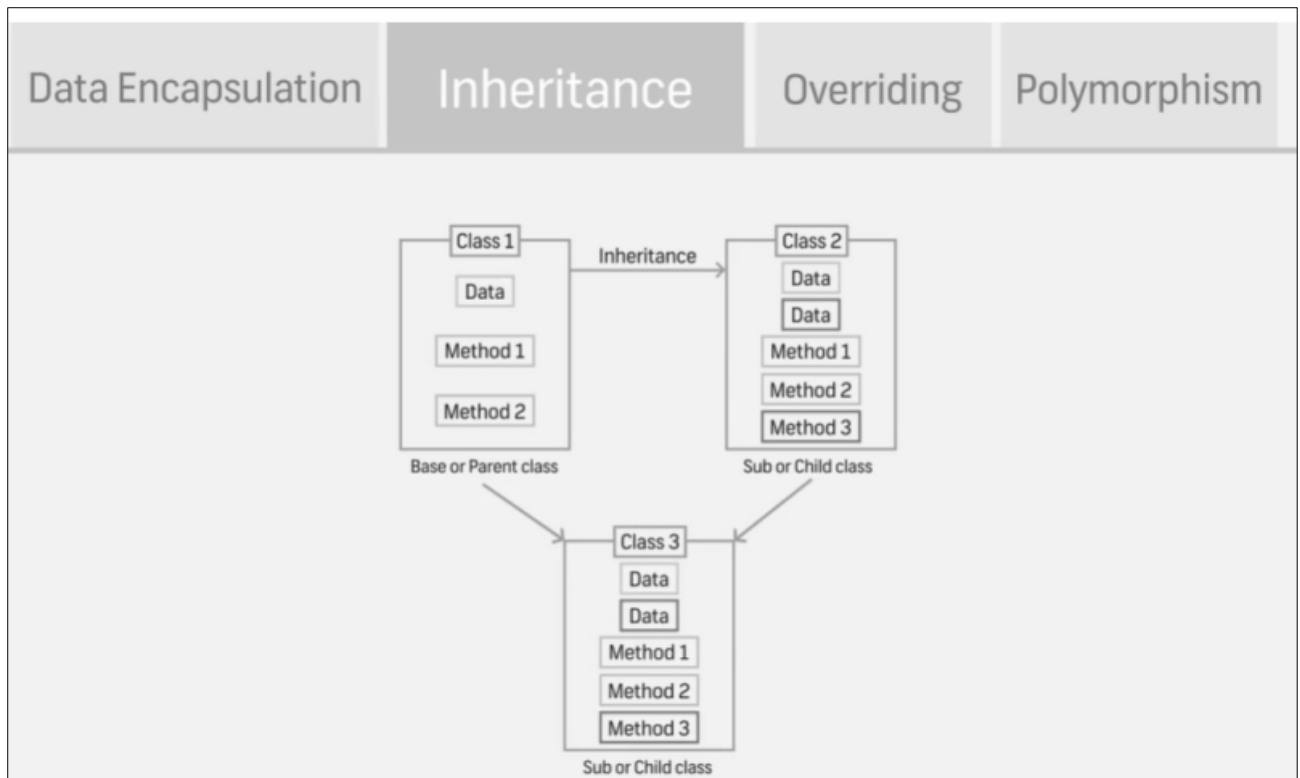
INHERITANCE:

It is this feature that lets you create a new class from an existing class.

So if this is a class with some data and methods you can build a new class using this class like this. This new class will have all the data and methods of the old class and you can add more data and methods of your own. In OOP terminology existing class is called base or parent class while new class is called as Sub or child class.

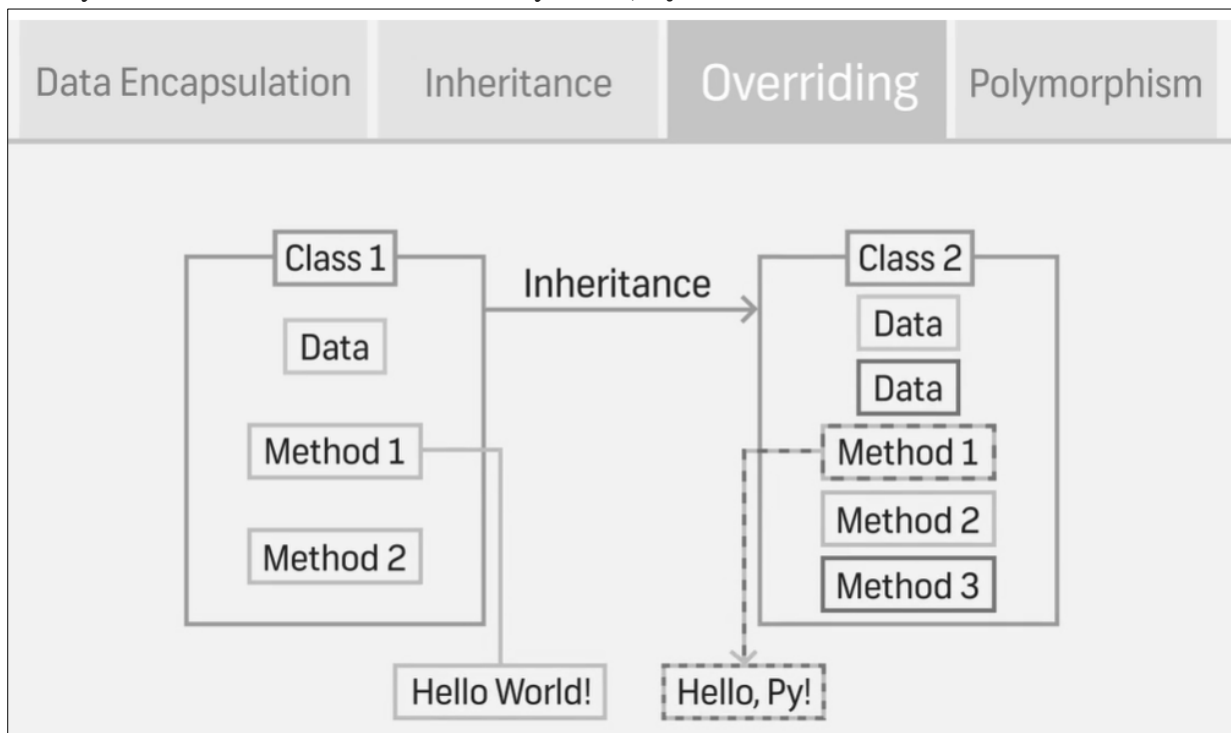


Child class inherits data and methods from parent's class. This facilitates reuse of features that are already available. Child class can add all or redefine a parent class function. It is also possible to design a new class based upon more than one existing classes, this feature is called Multiple inheritance.



OVERRIDING:

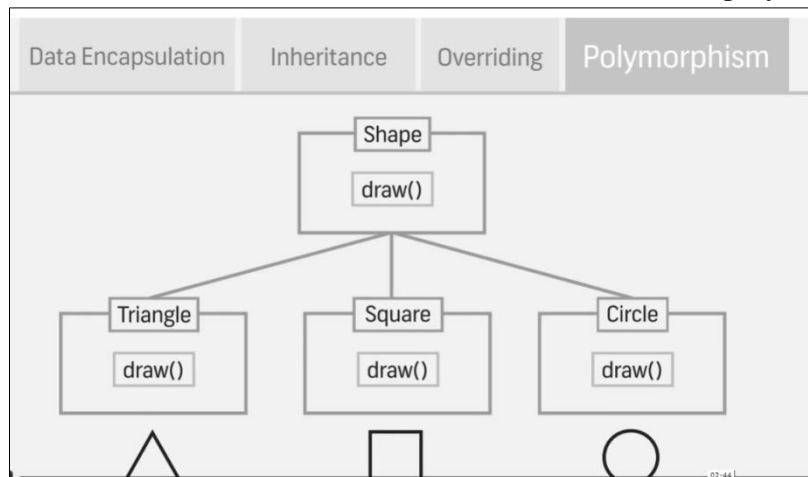
With inheritance you create child class from parent class. Overriding is a feature which lets you redefine parent class method. Like name suggests, you can override the parent class function or a method in the Child class. Look at this way. Using inheritance you created a new class from an existing class, now method 1 display **Hello World** on the screen you can modify this method in the new class to say **Hello, Py**.



This is called overtrading method.

POLYMORPHISM:

In OOP, when each child class provides own implementation of an abstract function in parent class, it's called polymorphism. You can understand it using shapes. Suppose you have an abstract class named Shape and it is an abstract draw method in it. You make several other concrete classes using this class like triangle, square and circle. All these classes have the draw method. The draw method in first class will draw a triangle. second draw function will draw square and third draw function will draw circle. This is known as polymorphism.



QUIZ:

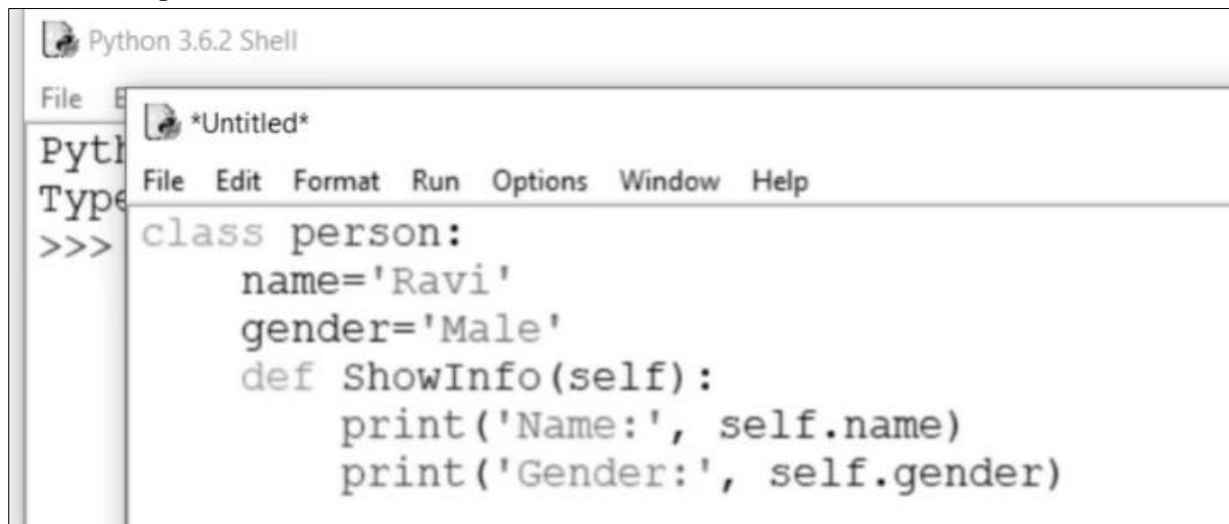
DECLARING CLASSES AND OBJECTS:

The design of Python language is such that each and every element in Python program is an object of one or the other class. Number, string, list, dictionary etc all objects of corresponding built-in classes. You can test this by using the type function.

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (
Type "copyright", "credits" or "license()" for more information.
>>> num=20
>>> type(num)
<class 'int'>
>>> num1=55.50
>>> type(num1)
<class 'float'>
>>> s="Internshala"
>>> type(s)
<class 'str'>
>>> dct={'a':1, 'b':2, 'c':3}
>>> type(dct)
<class 'dict'>
>>> def SayHello():
    print("Hello world!")
    return
>>> type(SayHello)
<class 'function'>
>>> |
```

class {name of the class}:

We will make a basic person Class and create different people from it. We will start with class name the class person and add a colon after it. The person would have a name. So lets add a name variable and initialise it with the value Ravi/ the person would have a gender so we add another variable gender and initialise it with male. Now let's add a showInfo method. In Python whenever we are creating a method in a class, we have to pass the arguments self. Just keep in mind that whenever you are creating a method in the class you have to pass this argument as the first argument. Then we print the information of the person like this. Our class is complete.



```
Python 3.6.2 Shell
File Edit Format Run Options Window Help
*Untitled*
class person:
    name='Ravi'
    gender='Male'
    def ShowInfo(self):
        print('Name:', self.name)
        print('Gender:', self.gender)
```

Now we create an object from this class like this and we can get the info of person 1. With this you have created an object from your own class but there is a problem with this.

```
>>> p1=person()
>>> p1.ShowInfo()
Name: Ravi
Gender: Male
>>> |
```

No matter how many different objects are person to create from this class, each person would have the same name as Ravi and gender as male. What we want is that we give the name and gender to the class and it creates person for us. To do this we have a special method in python called `__init__` method. The `__init__` method is invoked every time in object is created from a class and is called as constructor. Its job is to initialises the attributes of the object. so we just copy the variables and add them inside `__init__` method like this. we add self. before each variable and change their values to a and b. Then we add a and b as the arguments after self and initialise a and b with RAVI and male. Now our class is ready. let's make objects of this. we do it like this.

```
class person:
    def __init__(self, a='Ravi', b='Male'):
        self.name=a
        self.gender=b
    def ShowInfo(self):
        print('Name:', self.name)
        print('Gender:', self.gender)
```

Give the name and gender as the arguments call the showInfo method. Let's make another object and Run it. Yes we created two objects of two people. Sophie and Mark,

```
>>> p1=person('Sophie', 'Female')
>>> p1.ShowInfo()
Name: Sophie
Gender: Female
>>> p2=person('Mark', 'Male')
>>> p2.ShowInfo()
Name: Mark
Gender: Male
>>>
```

Now let's take a step back and see what we did

We made a class called person and then we added some variables to it inside `__init__` method. Then we added another method called showInfo which basically prints out the information of the person. Then we created two objects from it. When you create an object from a class like this, this is what happens in the background. A request is sent to the class stating that the user is asking for an object with the name Sophie and gender female. The class then takes the name and gender values and passes it on to the `__init__` method. The `__init__` method then assigns these values to the respective variables. The ShowInfo method then takes the value from `__init__` method and prints along with the statement.

```

> class person:
  > def __init__(self, a='Ravi', b='Male'):
    > self.name=a
    > self.gender=b
  def ShowInfo(self):
    print('Name: ', self.name)
    print('Gender: ', self.gender)

>>>p1=person('Sophie', 'Female')
>>>p1.ShowInfo()

>>>p2=person('Mark', 'Male')
>>>p2.ShowInfo()

```

The variables inside the `__init__` method are referred as attributes or Instance variables and the `__init__` method and `ShowInfo` methods are called as Instance methods or Methods.

```

class person:
    def __init__(self, a='Ravi', b='Male'):
        self.name=a
        self.gender=b
    def ShowInfo(self):
        print('Name: ', self.name)
        print('Gender: ', self.gender)

>>>p1=person('Sophie', 'Female')
>>>p1.ShowInfo()

>>>p2=person('Mark', 'Male')
>>>p2.ShowInfo()

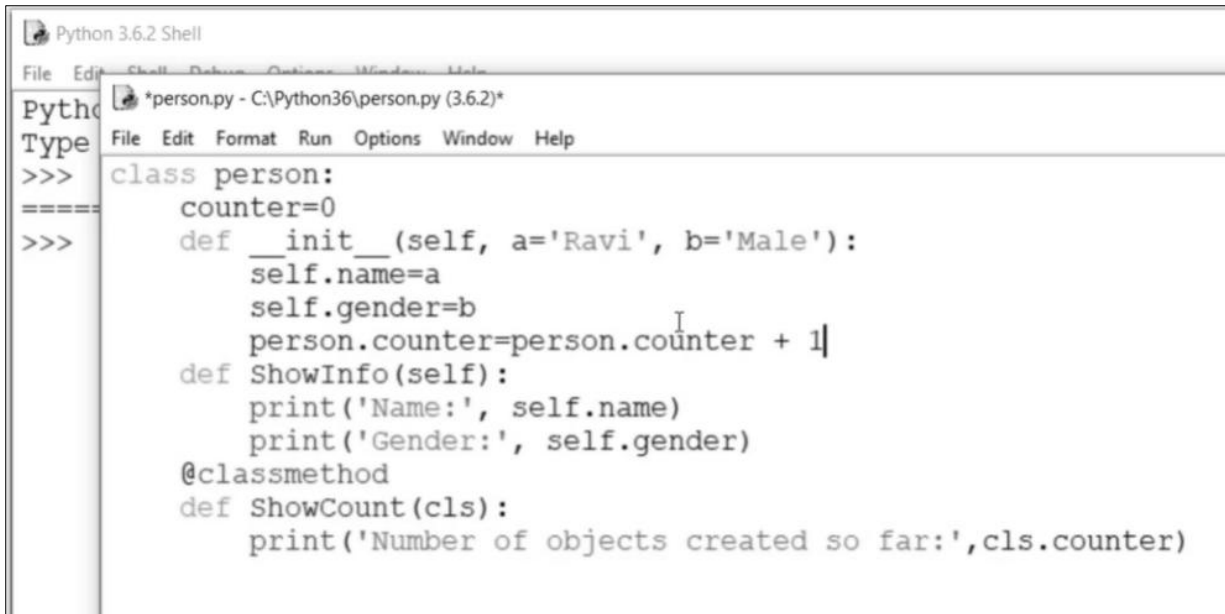
```

Instance Methods or Methods

Attributes or Instance Variables

CLASS ATTRIBUTES AND CLASS METHODS:

To understand these terms, Let us add a functionality to see how many objects have been created from our person class. For that first we will add class attribute to counter in our class and outside the `__init__` method .We initialise it with zero. Then we create a class method to print a object created so far like this, also we need to add one to the counter every time when object is created. So we will add this functionality in the `__init__` method as `__init__` method is invoked everytime whe object is created. Now test this create 3 objects.



```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
*person.py - C:\Python36\person.py (3.6.2)*
File Edit Format Run Options Window Help
>>> class person:
=====
>>>     counter=0
>>>     def __init__(self, a='Ravi', b='Male'):
>>>         self.name=a
>>>         self.gender=b
>>>         person.counter=person.counter + 1
>>>     def ShowInfo(self):
>>>         print('Name:', self.name)
>>>         print('Gender:', self.gender)
>>>     @classmethod
>>>     def ShowCount(cls):
>>>         print('Number of objects created so far:',cls.counter)

```

and then call the class method to see if the code is working fine. We call a class method by the name of class like this. We created 3 objects and it says 3 objects have been created, that means our code is working fine.

```

>>> p1=person('Deepak', 'Male')
>>> p2=person('Matt', 'Male')
>>> p3=person('Sophie', 'Female')
>>> person.ShowCount()
Number of objects created so far: 3
>>>

```

As we saw, the name and gender attributes have different values for each object. An attribute whose value is same for all instances of a class is called class attribute. Value of class attribute is shared by all objects. Class attribute is defined at class level rather than inside `__init__` method. Unlike Instance attributes which belong to a specific object, they are accessed through the name of the class. In the person class `__init__` as well as `ShowInfo` methods are instance methods as they receive the object's reference in the form of `self` argument. A method callable by a class and not by individual object is called as class method. We add `@classmethod` before such method and instead of passing `self` as an argument, we pass `cls` as an argument.

```

class person:
    counter=0 —————• Class Attribute
    def __init__(self, a='Ravi', b='Male'):
        self.name=a
        self.gender=b
        person.counter=person.counter + 1
    def ShowInfo(self):
        print('Name: ', self.name)
        print('Gender: ', self.gender)
    @classmethod
    def ShowCount(cls): —————• Class method
        print('Number of objects created so far: ', cls.counter)

```

- You add the attributes inside the `__init__()` method in a class.
- `__init__()` method is invoked every time an object is created from the class.
- You have to pass 'self' as the first argument whenever you are creating a method in a class.
- Attributes have different values for each object. But a Class Attribute's value is same for all the instances of a class.
- Class Methods are callable by class and not by an individual object. You add '@classmethod' before every class method and you pass the argument 'cls' instead of 'self' in a class method.

DATA HIDING IN PYTHON

In this article, we will discuss data hiding in Python, starting from data hiding in general to data hiding in Python, along with the advantages and disadvantages of using data hiding in python.

What is Data Hiding?

Data hiding is a concept which underlines the hiding of data or information from the user. It is one of the key aspects of Object-Oriented programming strategies. It includes object details such as data members, internal work. Data hiding excludes full data entry to class members and defends object integrity by preventing unintended changes. Data hiding also minimizes system complexity for increase robustness by limiting interdependencies between software requirements. Data hiding is also known as information hiding. In class, if we declare the data members as private so that no other class can access the data members, then it is a process of hiding data.

Data Hiding in Python:

The Python document introduces Data Hiding as isolating the user from a part of program implementation. Some objects in the module are kept internal, unseen, and unreachable to the user. Modules in the program are easy enough to understand how to use the application, but the client cannot know how the application functions. Thus, data hiding imparts security, along with discarding dependency. Data hiding in Python is the technique to defend access to specific users in the application. Python is applied in every technical area and has a user-friendly syntax and vast libraries. Data hiding in Python is performed using the `__` double underscore before done prefix. This makes the class members non-public and isolated from the other classes.

Example:

class Solution:

```
__privateCounter = 0
```

```
def sum(self):
```

```
    self.__privateCounter += 1
```

```
    print(self.__privateCounter)
```

```
count = Solution()
```

```
count.sum()
```

```
count.sum()
```

```
# Here it will show error because it unable
```

```
# to access private member
```

```
print(count.__privateCount)
```

Output:

Traceback (most recent call last):

```
File "/home/db01b918da68a3747044d44675be8872.py", line 11, in <module>
```

```
    print(count.__privateCount)
```

```
AttributeError: 'Solution' object has no attribute '__privateCount'
```

To rectify the error, we can access the private member through the class name :

class Solution:

 __privateCounter = 0

def sum(self):

 self.__privateCounter += 1

 print(self.__privateCounter)

count = Solution()

count.sum()

count.sum()

Here we have accessed the private data

member through class name.

print(count._Solution__privateCounter)

Output:

1

2

2

Advantages of Data Hiding:

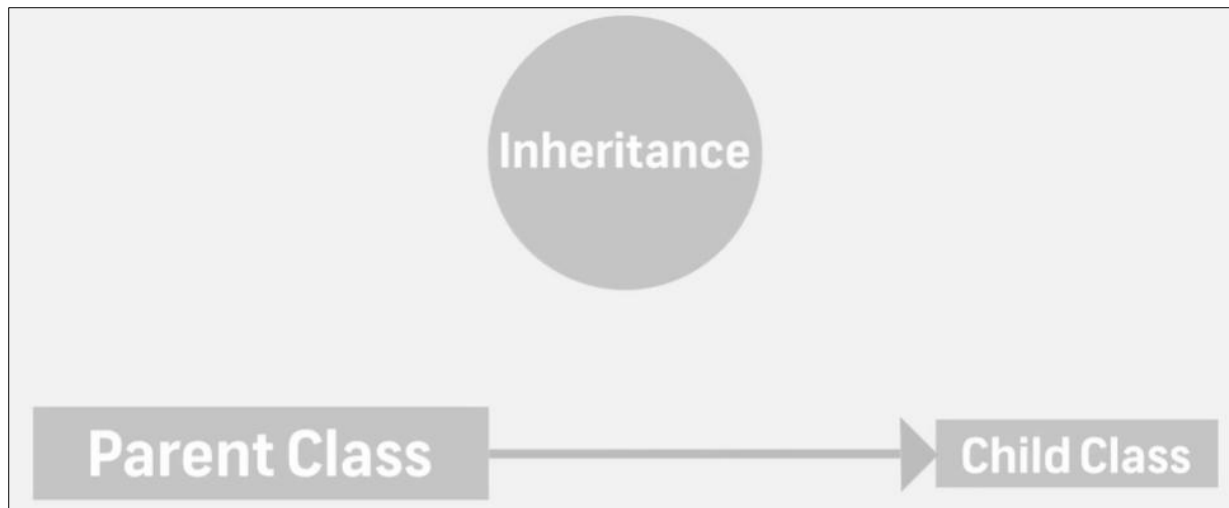
- It helps to prevent damage or misuse of volatile data by hiding it from the public.
- The class objects are disconnected from the irrelevant data.
- It isolates objects as the basic concept of OOP.
- It increases the security against hackers that are unable to access important data.

Disadvantages of Data Hiding:

- It enables programmers to write lengthy code to hide important data from common clients.
- The linkage between the visible and invisible data makes the objects work faster, but data hiding prevents this linkage.

INHERITANCE:

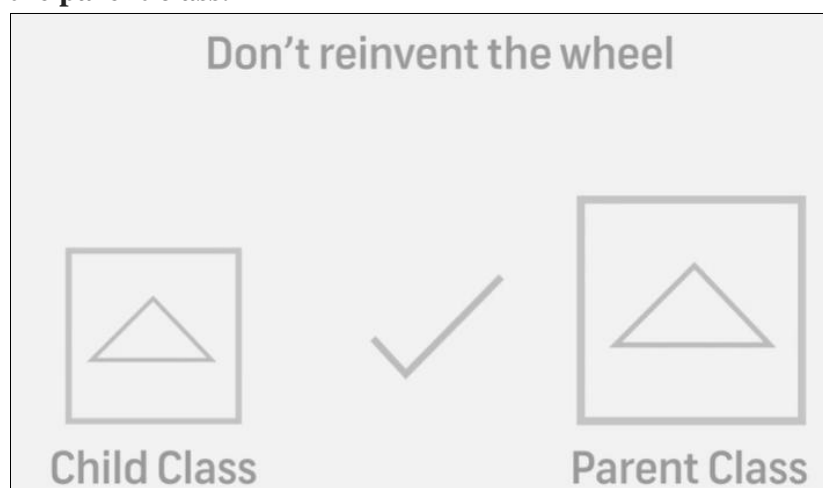
We know that every child class has parent call or base class. This child class is inherited class or subclass inherits the attributes of the base class.



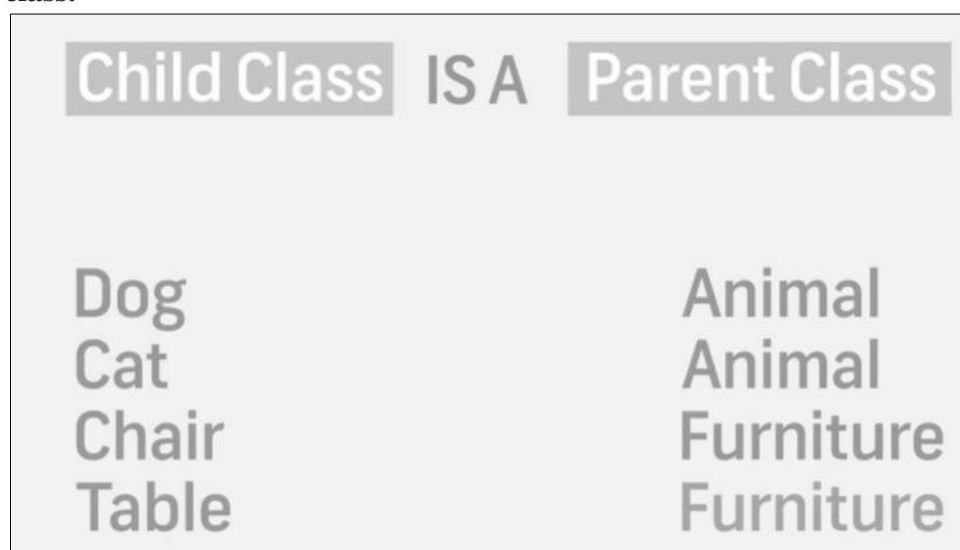
But before we see how ? let us understand why do we need inheritance in programming.

Principle of Inheritance: Don't reinvent the wheel

If you have to redefine a new class each time ,most of whose features are same as that of its parent class,then why must we redefine them.Why should the new class not just used the fetures of the parent class.



So in inheritance the new child class possesses is a relationship with a parent class or existing class.



Syntax of inheritance:

```
class parent:
    statement1
    statement2
    .
    .

class child(parent):
    statement1
    statement2
    .
    .
```

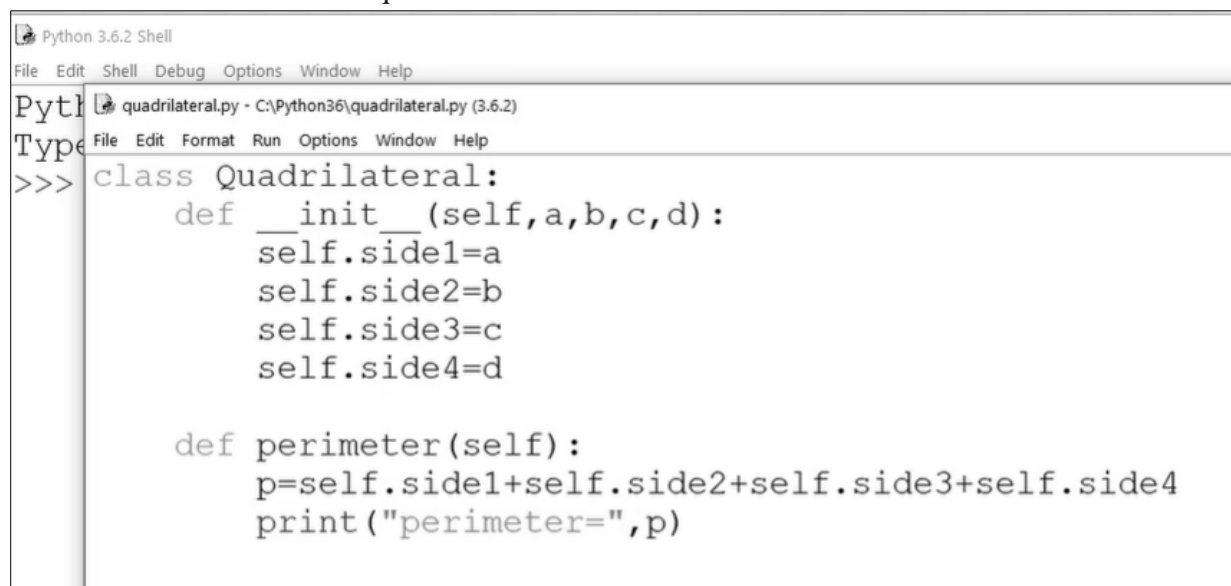
Here the parent class is defined first and has its own set of statements. Then the child class is defined. While defining child class the name of the parent class is put in paranthesis in front of it. This indicates that the attributes and methods of the parent class will be inherited by the object of child class.

How inheritance works:

Quadrilateral=Base class

Rectangle=Child class

Let us define a class quadrilateral. This class has four instance variables for a four sides and perimeter method. The constructor of the class receives four parameters which are a,b,c and d. Assigns them four instance variables side1, side2, side3 and side4. Then the perimeter method is defined which is equal to the sum of the four sides.



```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 Shell - C:\Python36\quadrilateral.py (3.6.2)
File Edit Format Run Options Window Help
Type
>>> class Quadrilateral:
>>>     def __init__(self, a, b, c, d):
>>>         self.side1=a
>>>         self.side2=b
>>>         self.side3=c
>>>         self.side4=d
>>>
>>>     def perimeter(self):
>>>         p=self.side1+self.side2+self.side3+self.side4
>>>         print("perimeter=",p)
```

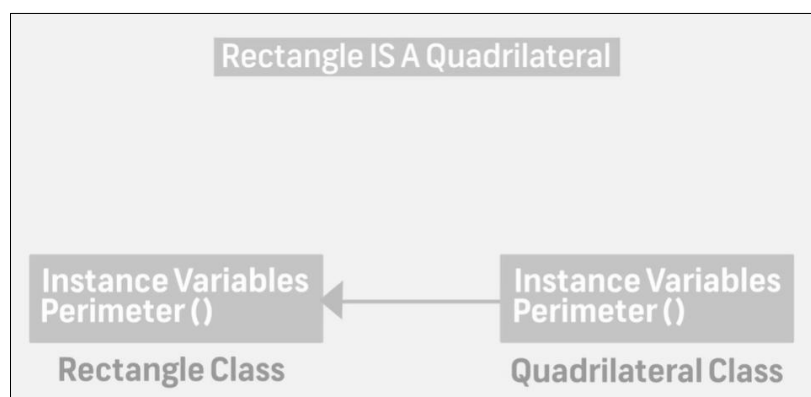
Now to test the class, let us declare its object and invoke the perimeter method.

```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Python36\quadrilateral.py
>>> q1=Quadrilateral(7,5,6,4)
>>> q1.perimeter()
perimeter= 22
>>>

```

Since rectangle is a quadrilateral, let us define a class rectangle based on the class quadrilateral that we have just defined. Now according to the rule of Inheritance, the instance variables and perimeter method of the base class quadrilateral should be automatically available to the new class rectangle. But we will customise variables according to the class rectangle.



Let us see how?

Let us first declare the class rectangle and put the base class within parenthesis, now since the two opposite side of rectangle are equal, we need only two adjacent sides to construct its object. Hence the other two parameters of the `__init__` method are set to none. The `__init__` method forwards the parameters to the constructor of its base class by using the `super` function. The object is now initialised with side3 and side 4 set to none. The opposite sides are made equal by the constructor of the rectangle class. Remember, we don't need to redefine the perimeter method. Since it is inherited from the base class. We cannot declare the object of the class rectangle and we call perimeter method like this.

```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 Shell
quadrilateral.py - C:\Python36\quadrilateral.py (3.6.2)
File Edit Format Run Options Window Help
>>> class Quadrilateral:
=====
>>>     def __init__(self, a, b, c, d):
=
>>>         self.side1=a
>>>         self.side2=b
>>>         self.side3=c
per:         self.side4=d
>>>
>>>     def perimeter(self):
>>>         p=self.side1+self.side2+self.side3+self.side4
>>>         print("perimeter=",p)
>>>
>>> class Rectangle(Quadrilateral):
>>>     def __init__(self, a, b, c=None, d=None):
>>>         super().__init__(a, b, c, d)
>>>         self.side3=self.side1
>>>         self.side4=self.side2

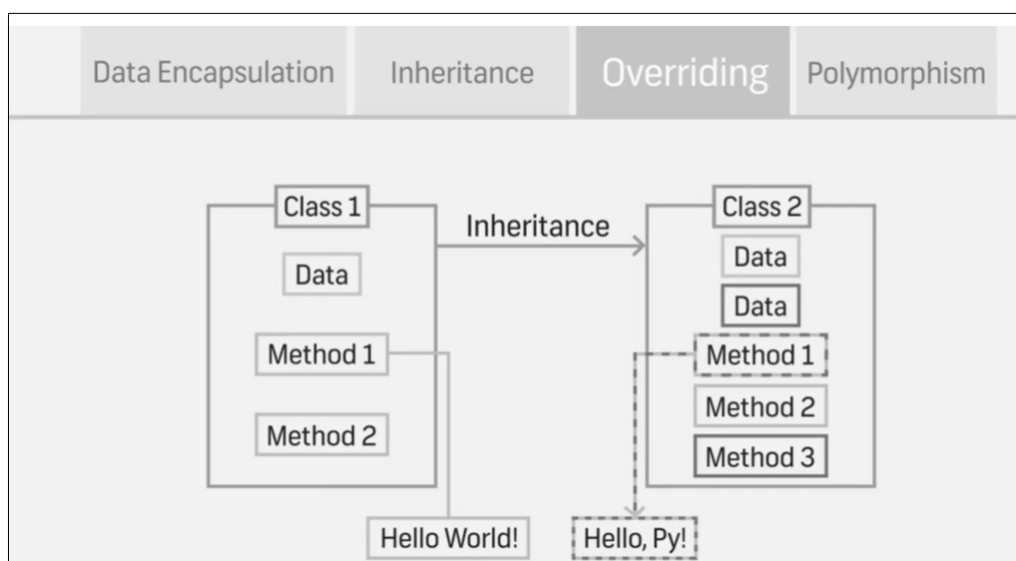
```

```

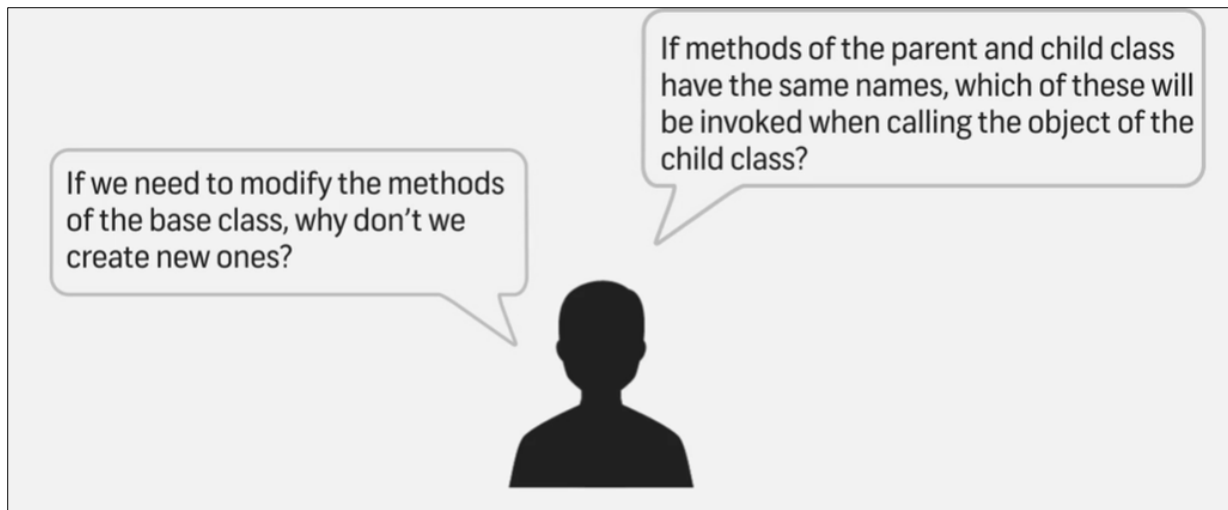
>>> r1=Rectangle(10,20)
>>> r1.perimeter()
perimeter= 60
>>>

```

OVERRIDING:



We know that while a child class inherits the method of parent class, it can modify these methods to suit its requirements. Let us answer for two questions below.

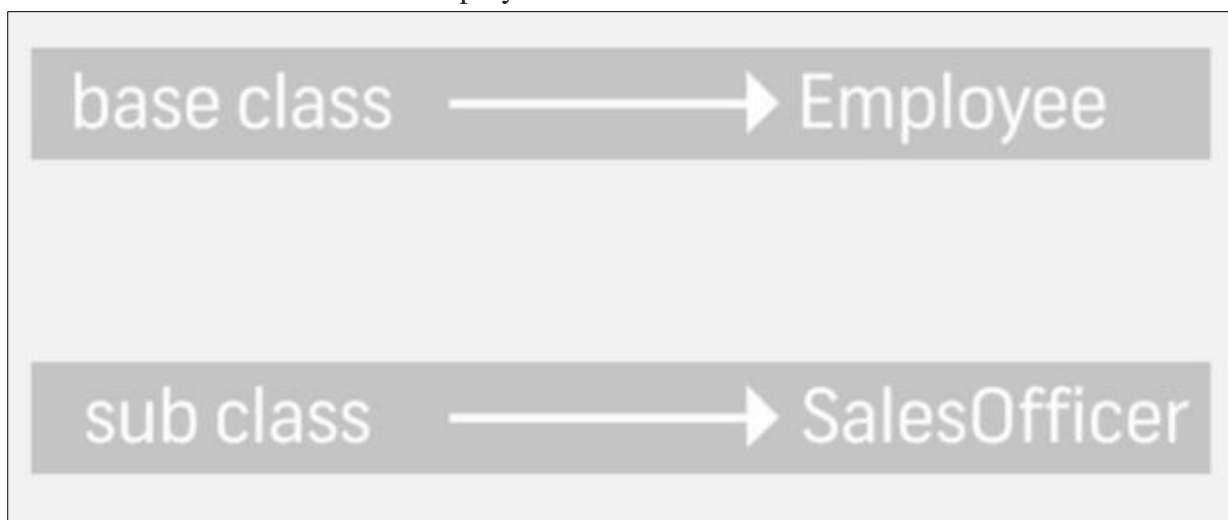


Answer of first question: That is because if we can utilise at least half of it, why shouldn't we?

Answer of 2nd question: Since we have modified the method for the child class it is given precedence over the method of the base class.

If we wanted to use method of base class only why would be modified in the first place?. Right. This is what meant by overriding. So the method of the child class overrides the method of the base class. Keep one thing in mind that only if both methods have the same names does overriding work. Now let us see how this works.

Let us define a base class called employee and subclass called sales officer.



Class employee has name and salary as instance variables and getname and getsalary methods. Let us declare class salesofficer with the base class employee in parenthesis. Now let us customise the getsalary method. We add incentive to the salary. So we see that getsalary method overrides base class method of the same name. To verify this let us create objects of both classes and invoke the getsalary method.

```

employee.py - C:/Python36/employee.py (3.6.2)
File Edit Format Run Options Window Help
class Employee:
    def __init__(self, nm, sal):
        self.name=nm
        self.salary=sal
    def getName(self):
        return self.name
    def getSalary(self):
        return self.salary

class SalesOfficer(Employee):
    def __init__(self, nm, sal, inc):
        super().__init__(nm, sal)
        self.incnt=inc
    def getSalary(self):
        return self.salary + self.incnt

e1=Employee("Rajesh", 9000)
print ("Total salary for {} is Rs {}".format(e1.getName(),e1.getSalary()))
s1=SalesOfficer("Kiran", 10000, 1000)
print ("Total salary for {} is Rs {}".format(s1.getName(),s1.getSalary()))

```

OUTPUT:

```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900]
Type "copyright", "credits" or "license()" for more information
>>>
===== RESTART: C:/Python36/employee.py =====
Total salary for Rajesh is Rs 9000
Total salary for Kiran is Rs 11000
>>>

```

We see that same method returns different values when invoked by objects of different classes.

```

employee.py - C:/Python36/employee.py (3.6.2)
File Edit Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900]
Type "copyright", "credits" or "license()" for more information
>>>
===== RESTART: C:/Python36/employee.py =====
Total salary for Rajesh is Rs 9000
Total salary for Kiran is Rs 11000
>>>

class Employee:
    def __init__(self, nm, sal):
        self.name=nm
        self.salary=sal
    def getName(self):
        return self.name
    def getSalary(self):
        return self.salary

class SalesOfficer(Employee):
    def __init__(self, nm, sal, inc):
        super().__init__(nm, sal)
        self.incnt=inc
    def getSalary(self):
        return self.salary + self.incnt

e1=Employee("Rajesh", 9000)
print ("Total salary for {} is Rs {}".format(e1.getName(),e1.getSalary()))
s1=SalesOfficer("Kiran", 10000, 1000)
print ("Total salary for {} is Rs {}".format(s1.getName(),s1.getSalary()))

```

Code Challenge

Make a new 'gift' class from the existing 'product' class and override 'get_price' method to add wrapping charge of Rs. 100 in the price.

```
class product:
    deliveryCharge=50
    def __init__(self,nam="Teddy Bear", prc=500):
        self.name=nam
        self.price=prc
    def get_name(self):
        return self.name
    def get_price(self):
        return self.price + product.deliveryCharge
    def __str__(self):
        return "The { } will cost you Rs.{ }.".format(self.get_name(),self.get_price())
```

METHOD OVERLOADING:

Method Overloading is an example of Compile time polymorphism. In this, more than one method of the same class shares the same method name having different signatures. Method overloading is used to add more to the behavior of methods and there is no need of more than one class for method overloading.

Note: Python does not support method overloading. We may overload the methods but can only use the latest defined method.

Example:

Function to take multiple arguments

```
def add(datatype, *args):
```

```
    # if datatype is int
    # initialize answer as 0
    if datatype == 'int':
        answer = 0
```

```
    # if datatype is str
    # initialize answer as ""
    if datatype == 'str':
```

```

    answer =''

    # Traverse through the arguments
    for x in args:

        # This will do addition if the
        # arguments are int. Or concatenation
        # if the arguments are str
        answer = answer + x

    print(answer)

# Integer
add('int', 5, 6)

# String
add('str', 'Hi ', 'Geeks')
```

Output:

11

Hi Geeks

METHOD OVERRIDING:

Method overriding is an example of run time polymorphism. In this, the specific implementation of the method that is already provided by the parent class is provided by the child class. It is used to change the behavior of existing methods and there is a need for at least two classes for method overriding. In method overriding, inheritance always required as it is done between parent class(superclass) and child class(child class) methods.

Example of Method Overriding in python:

class A:

```

    def fun1(self):
        print('feature_1 of class A')

    def fun2(self):
        print('feature_2 of class A')
```

class B(A):

```

    # Modified function that is
    # already exist in class A
    def fun1(self):
        print('Modified feature_1 of class A by class B')
```



```
def fun3(self):
    print('feature_3 of class B')
```

```
# Create instance
obj = B()
```

```
# Call the override function
obj.fun1()
```

Output:

Modified feature_1 of class A by class B

DIFFERENCE BETWEEN METHOD OVERLOADING AND METHOD OVERRIDING:

S.NO	Method Overloading	Method Overriding
1.	In the method overloading, methods or functions must have the same name and different signatures.	Whereas in the method overriding, methods or functions must have the same name and same signatures.
2.	Method overloading is a example of compile time polymorphism.	Whereas method overriding is a example of run time polymorphism.
3.	In the method overloading, inheritance may or may not be required.	Whereas in method overriding, inheritance always required.
4.	Method overloading is performed between methods within the class.	Whereas method overriding is done between parent class and child class methods.
5.	It is used in order to add more to the behavior of methods.	Whereas it is used in order to change the behavior of exist methods.
6.	In method overloading, there is no need of more than one class.	Whereas in method overriding, there is need of at least of two classes.

PYTHON PROGRAMS

1. Write a Python program to create a class to print an integer and a character with two methods having the same name but different sequence of the integer and the character parameters. For example, if the parameters of the first method are of the form (int n, char c), then that of the second method will be of the form (char c, int n)

```

class test:
    def testci(self,c,i):
        self.c=c
        self.i=i
        print("character==",self.c)
        print("Integer==",self.i)

    def testci(self,i,c):
        self.c=c
        self.i=i
        print("character==",self.c)
        print("Integer==",self.i)
a=test()
a.testci('a',12)

```

OUTPUT:--

```

character== 12
Integer== a

```

2. Write a Python program to create a class to print the area of a square and a rectangle. The class has two methods with the same name but different number of parameters. The method for printing area of rectangle has two parameters which are length and breadth respectively while the other method for printing area of square has one parameter which is side of square.

```

class area:
    def find_area(self,side):
        self.a=side*side
        return self.a
    def find_area(self,l,b):
        self.a=l*b
        return self.a
a=area()
result=a.find_area(4,6)
print(result)

```

OUTPUT:--

```

24

```

3. Write a Python program to create a class 'Degree' having a method 'getDegree' that prints "I got a degree". It has two subclasses namely 'Undergraduate' and 'Postgraduate' each having a method with the same name that prints "I am an Undergraduate" and "I am a Postgraduate" respectively. Call the method by creating an object of each of the three classes.

```

class degree:
    def getdegree(self):
        print("I got a degree")

class undergraduate(degree):
    def getdegree(self):

class postgraduate(degree):
    def getdegree(self):
        print("I am postgraduate")

a=degree()
b=undergraduate()
c=postgraduate()

a.getdegree()
b.getdegree()
c.getdegree()

```

OUTPUT:--

```

I got a degree
I am undergraduate
I am postgraduate

```

4.

Method overloading

To overload a method in Python, we need to write the method logic in such a way that depending upon the parameters passed, a different piece of code executes inside the function. Take a look at the following example:

```

class Student:
    def hello(self, name=None):
        if name is not None:
            print('Hey ' + name)
        else:
            print('Hey ')
# Creating a class instance
std = Student()
# Call the method
std.hello()
# Call the method and pass a parameter
std.hello('Prasad')

```

Output

```

Hey
Hey Prasad

```

5.

Method Overriding

The method overriding in Python means creating two methods with the same name but differ in the programming logic. The concept of Method overriding allows us to change or override the Parent Class function in the Child Class.

Example

Python Method Overriding

```
class Employee:
    def message(self):
        print('This message is from Employee Class')
class Department(Employee):
    def message(self):
        print('This Department class is inherited from Employee')
emp = Employee()
emp.message()
print('.....')
dept = Department()
dept.message()
```

Output

This message is from Employee Class

This Department class is inherited from Employee

6. What is the output of the following program?

```
# parent class
class Animal:
    # properties
    multicellular = True
    # Eukaryotic means Cells with Nucleus
    eukaryotic = True
    # function breath
    def breathe(self):
        print("I breathe oxygen.")
    # function feed
    def feed(self):
        print("I eat food.")
# child class
class Herbivorous(Animal):
    # function feed
    def feed(self):
        print("I eat only plants. I am vegetarian.")
herbi = Herbivorous()
herbi.feed()
# calling some other function
herbi.breathe()
```

Ans:

I eat only plants. I am vegetarian. I breath oxygen