**The Python OOPs Moments (Part — 01)**

[[Isha Choudhary](https://medium.com/@shahooda637?source=post_page-----9e16bfd4cc09--------------------------------)](https://medium.com/@shahooda637?source=post_page-----9e16bfd4cc09--------------------------------)

[Isha Choudhary](https://medium.com/@shahooda637?source=post_page-----9e16bfd4cc09--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fa0449bd47c69&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40shahooda637%2Fthe-python-oops-moments-part-01-9e16bfd4cc09&user=Isha+Choudhary&userId=a0449bd47c69&source=post_page-a0449bd47c69----9e16bfd4cc09---------------------post_header-----------)

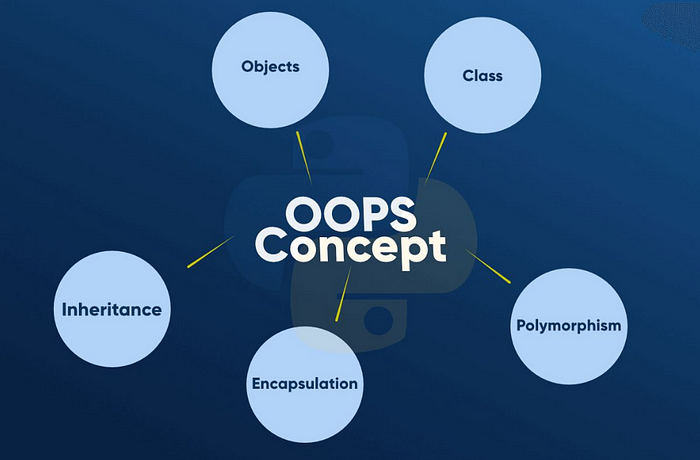
8 min read

·

Sep 13

5

**Object Oriented Programming in Python**



OOPs or Object Oriented Programming is a programming model used by the developers and computer programmers to organize the software design around data, or objects beings used for the software rather than around the functions and logic.

Wrapping and designing the program around data or objects enables the programmer to create a reusable program and/or code and write a clean and structured code. This helps them work in team in a structured manner where all the team members follow a defined structure making the complete process clean and well maintained.

An object in python or any other programming language can be defined as a data field which has its own unique attributed and own behavior. Objects allows the program to understand the specification about the tasks and variables being used throughout the program.

OOP focuses on the objects which are required to be manipulated through a logic rather than keep the logic at the center. This approach helps the programmers and their manager and team mates to manage and maintain large, complex, and actively updated program/code, which includes the code for mobile applications or system simulation software.

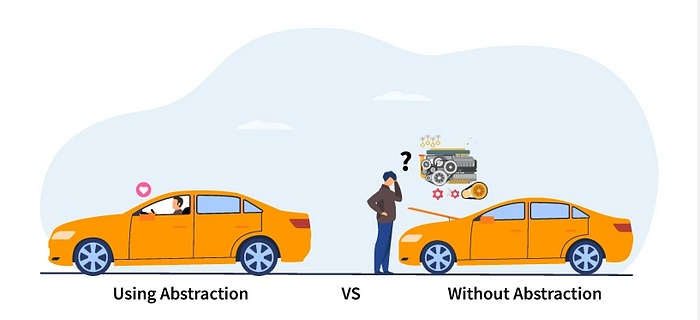
OOPs marked the workers/programmers to maintain the code and work in a collaborative environment on a same project.

Furthermore, some additional benefits of OOPs are reusability, scalability and efficiency.

This article discusses the 4 pillars of OOPs that are Abstraction, Encapsulation, Inheritance, and Polymorphism.

Following cells of code explains the four pillars of the object oriented programming through code.

**Abstraction**



**What is abstract class?**

The methods which do not have body, or is not performing any task or does not possess any logic is called an abstract method. The class consisting the abstract method is called the abstract class. And we cannot create an object of an abstract class. However, python does not support creating an abstract class. To create an abstract class in python we use a library ‘abc’ and import ‘ABC’ and ‘abstractmethod’ from it. Then we have to use a decorator ‘@abstractmethod’ inside a class to define the class as an abstract class.

Also, the class inheriting the abstract class must define all the abstract method present in the parent class. This is how we are able to use the abstraction and inheritance in the object oriented programming in python.

**Why do we need it?**

The need of an abstract class in the object oriented programming. It makes sense when we are developing a system and writing the code for that in the oops method. It give our code a flow and a systematic structure making the code clean and reusable. Abstraction is a way to create a flow in the code and to create a clean and structured code for our project. Large applications having multiple features, and classes for them, the practice of following oops and using abstraction helps the programmer to design their code in a good and understandable way.

The following cells shows a basic example of abstraction in python.

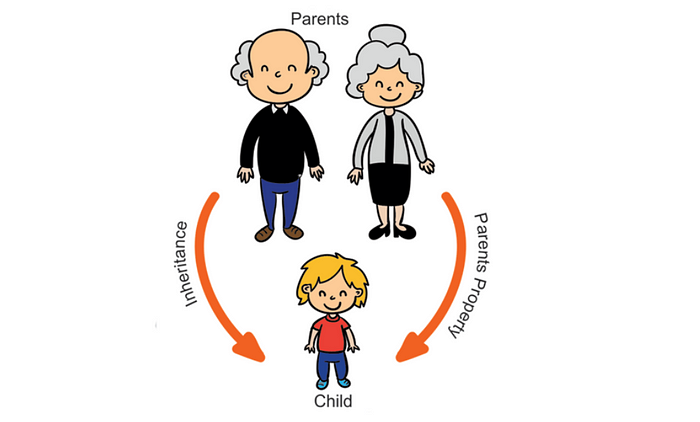
from abc import ABC, abstractmethod

#defining an abstract class with one abstract method  
class computer(ABC):  
 @abstractmethod  
 def process(self):  
 pass

#defining a class inheriting the abstract class  
class laptop(computer):  
 def process(self):  
 print("a working portable computer")  
  
#creating an object of child class  
lap = laptop()  
  
#calling the abstract method defined in the class "laptop"  
lap.process()

output:  
  
a working portable computer

**Inheritance**



In Object Oriented Programming, Inheritance is a concept to connect or establish a relationship between the classes and the objects , or among multiple classes and objects.

The literal meaning of inheritance is to inherit the properties from the parent. In OOP, the concept of inheritance allows us to use the properties of the parent class in the child class very easily.

The interesting thing here is that along with inheriting the properties and methods from the parent class, the child class can also have its own properties and methods. Also, we can also perform multiple inheritance along with direct and indirect inheritance.

The ease in using the properties, objects, and methods defined in the parent class through the child class allows the programmer to write clean, concise, and connected code.

The following cells showcase some examples of inheritance through code in python.

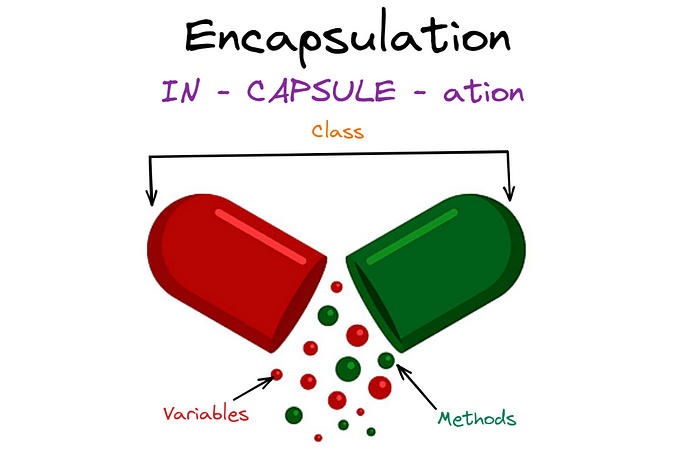
**Example — Human Body**

# defining the first class, the first parent class  
class head:  
 def head(self):  
 print("At the top we have head. \n")  
  
# inheriting the properties and methods form head into torso  
class torso(head):  
 def torso(self):  
 print("Below head is the torso. The hands are attatched to the torso of a human body. \n")  
  
# inheriting the properties and method from torso to seat while indirectly inheriting from head also  
class seat(torso):  
 def seat(self):  
 print("Below the torso is the private section of a human body. The below belt body starts here. \n")  
  
# inheriting from all head, torso, and seat indirectly and directly (from seat only)  
class legs(seat):  
 def legs(self):  
 print("At the end, the human body has legs, which include the foot and toes. \n")  
  
# inheriting from all head, torso, seat, and legs  
class human\_body(legs):  
 def body(self):  
 print("This is the complete human body!")  
  
# creating an object of class human\_body only  
Human = human\_body()

# class human\_body possess all the methods from all the classes through direct  
# and indirect inheritance  
  
Human.head()  
Human.torso()  
Human.seat()  
Human.legs()  
Human.body()

Output:  
  
At the top we have head.   
  
Below head is the torso. The hands are attatched to the torso of a human body.   
  
Below the torso is the private section of a human body. The below belt body starts here.   
  
At the end, the human body has legs, which include the foot and toes.   
  
This is the complete human body!

**Encapsulation**



In one word, “Encapsulation” means hiding the stuff from the users to ensure safety of your code and program.

In Object Oriented Programming, we have a concept through which we can hide the variables, and objects from the users or restrict them to accessing these directly by creating specific methods for users to use and do their desired tasks, without touching the real variables.

The principle of Encapsulation states that all the imporatant information is contained by the objects and as developers, you should only expose the selective, most needed information only.

The use of “Encapsulation”, the idea of hiding provides a greater security to your code and avoids unnecessary or unintended data corruption.

In real world, or industry level use cases, “Encapsulation” is very important to secure the valuable assets of the company.

The following cells shows the code for how do we do the “Encapsulation” in Python.

**Example 01 — Bank Account**

class bank\_account:  
  
 def \_\_init\_\_(self, name, age, typee, balance, interest):  
 self.\_\_name = name  
 self.\_\_age = age  
 self.\_\_typee = typee  
 self.\_\_balance = balance  
 self.\_\_interest = interest  
  
 def credit(self, amount):  
 if amount < 0:  
 return False  
 else:  
 self.\_\_balance = self.\_\_balance + amount  
 return True  
  
 def withdraw(self, amount):  
 if self.\_\_balance >= amount:  
 self.\_\_balance = self.\_\_balance - amount  
 return True  
 else:  
 return False  
  
 def get\_balance(self):  
 return self.\_\_balance

account01 = bank\_account("Isha Choudhary", "24", "Savings", 15000, 3.25)

account01.get\_balance()

Output:  
  
15000

account01.credit(-4253)

Output:  
  
False

account01.credit(5000)

Output:  
  
True

account01.get\_balance()

Output:  
  
20000

account01.withdraw(200000)

Output:  
  
False

account01.withdraw(2000)

Output:  
  
True

account01.get\_balance()

Output:  
  
18000

**Example 02 — Healthcare**

class healthcare:  
  
 def \_\_init\_\_(self, name, age, weight, height, bp, sugar, history, current\_disease):  
 self.\_\_name = name  
 self.\_\_age = age  
 self.\_\_weight = weight  
 self.\_\_height = height  
 self.\_\_bp = bp  
 self.\_\_sugar = sugar  
 self.\_\_history = history  
 self.\_\_current\_disease = current\_disease  
  
 def intro\_details(self):  
 print("Name: ", self.\_\_name)  
 print("Age: ", self.\_\_age)  
 print("Weight: ", self.\_\_weight)  
 print("Height: ", self.\_\_height)  
 print("Blood Pressure: ", self.\_\_bp)  
 print("Sugar Level: ", self.\_\_sugar )  
  
 return  
  
 def get\_history(self):  
 return self.\_\_history  
  
 def get\_problem(self):  
 return self.\_\_current\_disease

patient01 = healthcare("Shina", "45", "79", "168", "120/90", "80/70", "heart patient", "Anxiety")

patient01.intro\_details()

Output:  
  
Name: Shina  
Age: 45  
Weight: 79  
Height: 168  
Blood Pressure: 120/90  
Sugar Level: 80/70

patient01.get\_problem()

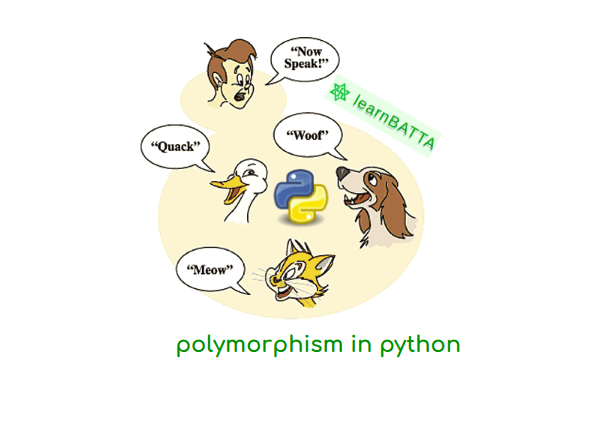
Output:  
  
'Anxiety'

patient01.get\_history()

patient01.get\_history()

Output:  
  
'heart disease'

**Polymorphism**



Polymorphism is one of the four pillers of Object Oriented Programming. It is a concept where objects are designed in a way to share behaviours with one another but process the data in different way.

The simple real life connection here is the humans, we are always the same person, everywhere we go. However, our behaviour and our actions differs in different situation. Same way the class having a function with one logic can process different objects and different data in different ways.

In polymorphism, the program determines which meaning or usage is necessary for each execution of the object from a parent class. It reduces the need to duplicate the code for every new possibility and helps the programmer to maintain clean code.

Furthermore, a child class is created extending the functionality of the parent class allowing different types of objects to pass through same interface, producing respective results for each different type of object.

Following cells of code explains the concept of “Polymorphism” in the form of python code.

**Example 01 — Class and Syllabus**

# creating polymmorphism through class and objects  
  
# defining class 01  
class data\_science:  
 def syllabus(self):  
 print("The Data Science Syllabus contains heavy Mathematics")  
  
# defining class 02  
class web\_development:  
 def syllabus(self):  
 print("The Web Development Syllabus in heavy on programming concepts")  
  
# creating the polymorphism concept as a class parcer  
def class\_parcer(class\_obj):  
 for i in class\_obj :  
 i.syllabus()  
  
# creating objects from both the classes  
data\_science = data\_science()  
web\_dev = web\_development()  
  
# one variable for multiple objects  
class\_obj = [data\_science, web\_dev]  
  
# passing the variable through function  
class\_parcer(class\_obj)

Output:  
  
The Data Science Syllabus contains heavy Mathematics  
The Web Development Syllabus in heavy on programming concepts

**Example 02 — Student and Teacher**

# defining students class  
class students:  
  
 def \_\_init\_\_(self, name, standard):  
 self.name = name  
 self.standard = standard  
  
 def details(self):  
 print("The student details are as follows: ")  
 print('Student Name: ', self.name)  
 print("Studying in class: ", self.standard)  
 print('\n')  
 return self.name, self.standard  
  
# defining teachers class  
class teachers:  
  
 def \_\_init\_\_(self, name, key\_class):  
 self.name = name  
 self.key\_class = key\_class  
  
 def details(self):  
 print("The teacher details are as follows: ")  
 print('Teacher Name: ', self.name)  
 print('Teaching Class: ', self.key\_class)  
 return self.name, self.key\_class  
  
# creating the polymorphism concept  
def class\_parcer(class\_obj):  
 for i in class\_obj:  
 i.details()  
  
# creating objects  
Shaan = students("Shaan Singh", "10 A")  
Rina = teachers("Rina Shah", "12 B")  
  
#creating common variable for all objects from different classes  
class\_obj = [Shaan, Rina]  
  
# the final class parcer to perform the polymorphism  
class\_parcer(class\_obj)

Output:  
  
The student details are as follows:   
Student Name: Shaan Singh  
Studying in class: 10 A  
  
  
The teacher details are as follows:   
Teacher Name: Rina Shah  
Teaching Class: 12 B

*Thanks for reading! Try clapping twice to experience something new on medium now.*