**Object Oriented Programming in Python**

**OOP allows us to create data types of our needs we can create one for our specific needs instead of using the one’s that are already present.**

**What are the core components of OOP?**

* 1. **Object**
  2. **Class**
  3. **Polymorphism**
  4. **Encapsulation**
  5. **Inheritance**
  6. **Abstraction**

**Class:** Class is a blueprint, blueprint of how an object will behave**.** Everything in python is an object and every object is part of an object, and an object cannot exist without a class.

A class contains data or property and functions or behavior.

Example:

Class Car:

Color = ‘blue’ #data

Model = ‘sports’ #data

def calculate\_avg\_speed(km, time):

# code to calculate speed from the given inputs

* The name of the class should be in pascal case
  + Example: ThePascalCase

**Object**: it is an instance of a class

example: Car ----------🡪 wagnor // wagnor = car()

sports----------------🡪 cricket // cricket = sports()

animals -----------🡪 dog // dog = Animal()

1. **What is the difference between method and a function?**
   1. Methods are functions which are written inside a class
   2. Whereas functions are normal functions which are not bound to any class
2. What is \_\_init\_\_() method?
   1. Init is a constructor which is a special method. Whatever is written inside init is always executed when an object of the class is created.
3. What are magic methods?
   1. These are special methods in python which can be identified by double underscore before and after the method
   2. These are automatically triggered
   3. Constructors are also magic methods
   4. Constructors are used for configuration purposes, where we do not want the user to interact with.

Example: when we open an app we want it to directly access the internet without the use explicitly connecting to the internet, so we write the configuration for the app to connect to the internet within a constructor which gets executed upon opening the app which connects it to the internet.

1. What is self?
   1. Self refers to the current instance or object of the class.
   2. Every method in the class requires self as the methods of a class can only be accessed by the objects of that class and without an object of the class the methods do not exist. So self is must for methods inside a class.
2. What are magic methods?
   1. K
3. What are dunder methods?
4. What are instance variables?
   1. Instance variables are those variables which are used inside the constructors, these are variables which store different values for different objects and hence the value of a variable depends on the instance or the object they are working on at that time and hence they are called as instance variables.
5. What are private variables and public variables?
   1. We can make variables and even methods private by adding double underscore before them. For example: self.\_\_pin, self.\_\_balance, def \_\_sum():, etc..
   2. When we do this python interpreter understands that these are private variables and methods and it hides them from the user
   3. But behind the scenes what happens is that, the python interpreter converts the variable and method names to something like this, \_<class\_name>\_\_<variable/method name> and if we try to access them using the object like this object.\_\_<variable/method name> python creates a new variable with this name but not method, and hence this does not effect the program in any way.
   4. But we can still manage to access it by using the \_classname followed by \_\_variablename.
   5. Nothing in python is truly private.
   6. And public variables are those which can be accessed by the user.

**Encapsulation:**

* **It includes getter and setter methods.**
* **Private attributes**
* We should not let the user/anyone access the data directly, but if there is a need to access the data then we should at least allow them to view it and to do this we can use 2 methods, getter and setter methods.
* Getter is used to get data, and setter is used to set the data.
* And since we are using methods, we can control the functionality of what the user can do with these variables/ data.

**Reference variables:**

* **These are the variables which are used to create an object of a class.**
  + For example: sbi = Atm(); cricket = Sports(). Here sbi and cricket are reference variables used to point to the atm and sports objects.

**Pass by Reference:**

* We can pass an object of a class as an argument to a function.

**Example:**

**c**lass Customer:

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def greet(customer):

if customer.gender == ‘male’:

print(f’Hello {customer.name} sir’)

elif customer.gender == ‘female’:

print(f’Hello {customer.name} ma’am)

cust = Customer(‘Rohith’, ‘male’)

greet(cust)

when we run this we get **“Hello Rohith sir”.**

* Objects of a class are mutable in python.
* For example,
  + class Customer:
    - def \_\_init\_\_(self, name):
      * self.name = name
  + def greet(customer):
    - print(id(customer) # this gives the address of customer object.
    - customer.name = ‘pandu’
    - print(customer.name) # this will print pandu as we have changed the name
    - print(id(customer)) # we have just changed the name but the object still points to the same location of address and hence the address remains same.
* cust = Customer(‘Rohith’)
* print(id(cust)) # the id will be same as above
* greet(cust)
* print(cust.name) # since during the function execution the name is changed to pandu this will remain effective and the output is pandu.

This proves that objects are mutable.

**Static:**

* **static variable/ class variable**
  + static variables are always initialized outside the methods.
  + For example,
    - class Atm:
      * \_\_counter =1
      * def \_\_init\_\_(self):
        + self.pin = “”
        + self.balance = 0
        + self.sno = Atm.counter
        + Atm.counter += 1
      * …..code
* Static variables are used when there is something that needs to be shared between objects.
* **static method**
  + similar to variables if we make the static variable private and then allow the user or others to access the static variable, we can do it using setter and getter methods.
  + But here since the static variable is not a instance variable it does not need self-argument in the method.
  + Consider the above Example:
    - **@staticmethod**
    - def get\_counter():
      * return Atm.\_\_counter
    - @staticmethod
    - def set\_counter(new):
      * if type(new) == int:
        + Atm.\_\_counter = new
      * else:
        + print(“not allowed)
* since these methods use static variable we use @staticmethod above them to indicate that these are static methods

**Relationship between classes:**

**There are 2 major types of relationship between classes**

1. **aggregation (has a relationship)**
   1. **example: customer has an address, here customer class requires address class, hence aggregation**
2. **inheritance (is a relationship)**
   1. **example: smartphone is a product, car is a vehicle; here car will have all the properties of vehicle and similarly smartphone will have all the properties of product. This is called inheritance.**

**Example of aggregation:**

**'''**

**Classes majorly exhibit 2 kinds of relationships**

**1. Aggregation**

**2. Inheritance**

**'''**

**#Aggregation demo**

**class Customer:**

**def \_\_init\_\_(self, name, gender, address):**

**self.name = name**

**self.gender = gender**

**self.address = address**

**def edit\_profile(self, new\_name, new\_city, new\_pin, new\_state):**

**self.name = new\_name**

**self.address.change\_address(new\_city, new\_pin, new\_state)**

**class Address:**

**def \_\_init\_\_(self, city, pincode, state):**

**self.city = city**

**self.pincode = pincode**

**self.state = state**

**def change\_address(self, new\_city, new\_pincode, new\_state):**

**self.city = new\_city**

**self.pincode = new\_pincode**

**self.state = new\_state**

**add = Address('Nspt', 506132, 'TG')**

**cust = Customer('Rohith', 'Male', add)**

**cust.edit\_profile('Pandu','Wgl', 506100, 'TG')**

**print(cust.address.pincode)**

**Inheritance:**

* **A child class inherits all the properties of the parent class but there are a few restrictions too like**
  + **It inherits data members, member functions or methods, constructors**
  + **But it does not inherit private members like private variables and private methods.**
* **When the child class does not have a constructor and it inherits the parent class then the parent class constructor gets called.**
* **But at the same time if there is a constructor in the child class then this will be executed and the parent constructor is not invoked and the parent constructor will not have anything.**

**Super:**

* **Super only works inside the child class and it cannot be used outside the class.**
* **When we use super() inside the child class this will go back to the parent class for execution depending on the following operation.**

**Types of inheritance:**

* **Single level inheritance: parent🡨 child (here child inherits the parent class)**
* **Multi-level inheritance: parent1🡨 parent2🡨 child (here child inherits both parent1 and parent2 while parent2 will only inherit parent1)**
* **Hierarchial Inheritance: here multiple children inherit the same parent. P 🡨 C1, P🡨C2**
* **Multiple inheritance: here a single child inherits multiple parents: P 🡨 C, M🡨 C.**
* **Hybrid Inheritance: this is a combination of multiple of the above inheritance types.**

**Method Resolution Order:**

* **If a child class inherits multiple parent classes and both the parent classes have the same method then upon invoking the method using the child class the child class will invoke the method of the parent which is first passed to it.**
* **For example: class phone, class product, both have method buy and if a child class SmartPhone(phone, product) inherits both and calls the buy method then it will invoke phone class as this class is first passed to it.**
* **And this is called MRO (Method Resolution Order).**

**Polymorphism:**

**Method Overriding:**

* **When both child class and the parent class have the same method names then the child class will override the parent class method when called.**
* **This is due to polymorphism.**

**Method Overloading:**

* **This does not work in python but works in java.**
* **This means the same function works differently for different inputs**
* **Consider an example of calculating the area of a circle and rectangle**

**class Geometry:**

**def area(self, a):**

**return 3.14\*a\*a**

**def area(self, a, b):**

**return a\*b**

* **But the above code does not work in python as in python the second function takes precedence over the first function.**
* **But we can use the below to do the above operation.**

**class Geometry:**

**def area(self, a, b=0):**

**if b==0:**

**print(“circle”, 3.14\*a\*a)**

**else:**

**print(“rectangle”, a\*b)**

**this is also a form of method overloading as for different inputs this function will behave differently.**

**a = Geometry()**

**a.area(4) # this prints area of a circle**

**a.area(4,5) # this prints area of rectangle**

**Operator Overloading:**

* **Operator overloading is when operators behave differently with different inputs.**
* **For example, addition operation performs differently with strings where it performs string concatenation while it adds 2 integers or numbers.**