Object Oriented Programming Using Python

1. Introduction to Object Oriented Programming in Python
2. Python Constructors & python all about Methods
3. What are Classes and Objects?
4. Object-Oriented Programming methodologies:

- **Inheritance**
- **Polymorphism**
- **Encapsulation**
- **Abstraction**

1. Introduction to Object OrientedProgramming in Python

Object Oriented Programming is a way of computer programming using the idea of

"objects" to represents data and methods. It is also, an approach used for creating neat and reusable code instead of a redundant one.

2.Python Constructors

Python Constructors:

Python facilitates a special type of method, also called as Python Constructors, to initialize the instance members of the class and to verify enough object resources for executing any startup task.

Types of Constructors:

- Parameterized Constructor
- Non- Parameterized Constructor Features of Python Constructors:
- In Python, a Constructor begins with double underscore (_) and is always named as __init__().
- In python Constructors, arguments can also be passed.
- In Python, every class must necessarily have a Constructor.
- If there is a Python class without a Constructor, a default Constructor is automatically created without any arguments and parameters.

Example:

```
#without constructor
class parent:


   def family(self,name,age):

       print(f"my nane is {name} &{age}")
```

```
p = parent()

p.family("sayem",26)                    #sayem

#with constructor

class parent:
    def __init__(self,name,age):
        print(f"my nane is{name} & age
{age}")


p1 = parent("sayem",26)
# p1.family("sayem",26)
                                        #sayem
```

Python all about Method

- Instance Methods
- Class Methods
- Static Methods

1.Instance methods are associated with an instance of a class and can modify the data stored within the instance.

Example:

```
class classname:
```

```
    def instancemethod(self):#instancem

        print("hellow instancemethod")



v1 = classname()

v1.instancemethod()                          #sayem
```

2.Class methods are associated with the class rather than an instance and can access and modify class-level data. **example:**

```
class classname:

    @classmethod

    def classmethod(self):#@classmethod

        print("hellow classmethod")


classname.classmethod()                      #sayem
```

3. Static methods are similar to functions outside of the class and cannot access any instance or class data. **example:**

```
class classname:

    @staticmethod
```

```
    def staticmethod():#@staticmethod
        print("hellow classmethod")

classname.staticmethod()                    #sayem
```

3. What are Classes and Objects?

A class is a collection of objects or you can say it is a blueprint of objects defining the common attributes and behavior. Now the question arises, how do you do that? Class is defined under a “Class” Keyword

Creating an Object and Class in python:
Example:

```
class father:#father is class
    house = "10th floor"
    car = "BMW"
y = father()#y is object
print(y.car)
                                            #sayem
```

4. Object-Oriented Programming Methodologie

- ❑ **Inheritance**
- ❑ **Polymorphism**
- ❑ **Encapsulation**

❑ Abstraction

Inheritance:

Ever heard of this dialogue from relatives “you look exactly like your father/mother” the reason behind this is called ‘inheritance’. From the Programming aspect, It generally means “inheriting or transfer of characteristics from parent to child class without any modification”. The new class is called the derived/child class and the one from which it is derived is called a parent/base class.

```
class baba:
    house = "10 floor"
    car = "bmw"
    tk = '10 core'
class son(baba):
    phone = "iphone"
```

```
    car1 = "toyota"
b = son()
print(son.tk)                                    #sayem
```

Polymorphism:

You all must have used GPS for navigating the route, Isn't it amazing how many different routes you come across for the same destination depending on the traffic, from a programming point of view this is called 'polymorphism'. It is one such OOP methodology where one task can be performed in several different ways. To put it in simple words, it is a property of an object which allows it to take multiple forms

Example:

```
class Vehicle:
   def __init__(self, model, brand, component):
       self.model = model
       self.brand = brand
       self.component = component


class Computer(Vehicle): # Corrected the class name to "Computer" pass



class Car(Vehicle): # Corrected the class name to "Car"
   pass


p1 = Computer("2024", "BMW", "All") # Corrected the spelling of "BMW"

c1 = Car("2023", "CMW", "Fixt") # Corrected the spelling of "CMW"

print(c1.brand)
print(p1.brand)                                          #sayem
```

Encapsulation:

Encapsulation, which is the practice of obscuring data or implementation details within a class, is one of the fundamental concepts of object-oriented programming (OOP). This method combines the methods that act on the data with the data itself into a single unit. This enables the creation of objects with a predetermined set of attributes and behaviors that can be accessed and changed in a controlled way.

Encapsulation is a technique for hiding a class' internal operations and exposing only the information required for external interactions. It enables the creation of objects with clearly defined behaviors and properties, enhancing their usability and security. Encapsulation restricts access to a class's data to its methods and keeps the class's variables private.

Example:

```python
class Vehicle:

    def __init__(self, model, brand, component):

        self.__model = model #__is private

        self.brand = brand

        self.component = component



class Computer(Vehicle):  # Corrected the class name to "Computer"

    pass



class Car(Vehicle):  # Corrected the class name to "Car"

    pass



p1 = Computer("2024", "BMW", "All")  # Corrected the spelling of "BMW"



c1 = Car("2023", "CMW", "Fixt")  # Corrected the spelling of "CMW"



print(c1.__model)                                    #sayem
```