International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 03 | Mar 2021 www.irjet.net p-ISSN: 2395-0072

Object Oriented Programming Concepts Using Python

Abhishek V Tatachar[1], Vishwas K V[2], Shivanee Kondur[3]

[1,2,3] *Students, Department of Information Science and Engineering, Global Academy of Technology, Bangalore.*

Abstract - *With the incrementing need of representing real-world objects into software programs, object oriented programming has emerged as one of the popular methods to do so. Python is one such object oriented programming language, that is used for performing a variety of tasks such as interactive desktop application development, web application development, artificial intelligence and image processing, and many more applications. This paper concentrates on how object oriented concepts can be implemented using the python programming language.*

Key Words: Object Oriented Concepts, Python, Class Diagrams, Classes, Objects, Polymorphism, Data Abstraction, Inheritance.

1. INTRODUCTION

Object Oriented Programming or OOP as it is more commonly abbreviated as, is the process of implementing the program in terms of objects and classes. An object is a representation of a real life entity which comprises of state, behavior and properties. A class is basically a blueprint or a template to create an object. In other words, classes can be put up as group of objects that have similar characteristics. The class specifies the type and scope of its constituent members. We will be learning about objects and classes in the section 5. Based on these properties of object oriented programming, there are certain characteristics such as inheritance, polymorphism, abstraction and encapsulation, which makes more sense as to why object oriented programming is advantageous.

Fig-1: Objects and Classes

Here there are some different types of cars (sedan, SUV, sports, etc.) referred to as objects that belong to a particular class called cars

Python is one such object oriented programming language that was created by Guido van Rossum, which was released in the year 1991. In [1] Akshansh Sharma Et al have clearly indicated as to why python has grown as one of the most popular programming language. According to [1] python – has a largest Stack Overflow community, is one the most in demand skill and has a large career opportunity and it is the 4th most used language on GitHub. Python is known for its simplicity in terms of the syntax used, it being an open source platform, portability and a variety of available libraries. Python has a variety of fields of application. One can find python being used in Machine Learning and artificial intelligence, system programming, web applications, system programming, developing graphical user interfaces or GUIs and many more. In fact, the peer to peer file sharing network, Bit Torrent has been written in one. Google and Nasa are other users of python. One key benefit of python is that it is object oriented. That is, it allows the usage of classes and objects. To create a class in python we make use of the class keyword. The following is the syntax of how a class can be created in python.

```
class class_name:
    #block of code containing data members and methods
```

Now that a class has been created, how do we create an object, we do that by calling the class name as a function. Let's consider the following syntax for creating an object.

```
object_name = class_name()
```

This is just the introduction to how it is done, section 5 has more to explain about objects and classes.

2. LITERATURE SURVEY

The history of computing dates back to the invention of Pascaline by Blaise Pascal and the invention of the Analytical Engine by Charles Babbage. In the year 1936, Alan Turing, an English mathematician and computer scientist proposed a machine which was later called the Turing Machine, that consisted of an infinitely long tape which was divided into cells, a read-write head, and a memory for storing instructions. Soon, the first digital computer, called the Electronic Numerical Integrator and Computer (ENIAC) was furnished in 1942. Interested in this field, John von Neumann, a mathematician proposed a set of ideas which are now called the Von Neumann Architecture. Along these lines were the evolution of the computers and computing in general.

With the evolution of computing, programming paradigms gradually came into existence. Programming paradigms are methods that are used to solve problems using some programming languages. In [2] Dr. Selvakumar Samuel describes a programming paradigm as the core and basis of any programming language design. In [2] the author has reviewed most of the relevant literature pertaining to the concepts of programming paradigms, such as mainstream programming paradigms, programming paradigm notations, imperative programming paradigm concepts and much more. The different programming paradigms that are most commonly known are the imperative programming paradigm, declarative programming paradigm (contrasting to the imperative programming paradigm), procedural programming paradigm, functional programming paradigm, object oriented programming paradigm and so on. Programming languages are based on these programming paradigms.

One such programming paradigm is the structured programming paradigm. In [3], Karl Et al describes structured programming as a technique that was devised to improve the reliability and clarity of programs. He also tells that in structured programming, “the control of program flow is restricted to three structures, sequence, IF THEN ELSE, and DO WHILE, or to a structure derivable from a combination of the basic three”. Structured programming came to light in the late 1950s with the emergence of programming languages like ALGOL 58 and ALGOL 60. Although the structured programming is easy to understand due to the effective usage of selection statements, sequence and iteration statements, structured programming also has certain disadvantages. One disadvantage is its relative inefficiency in use of memory and the speed of execution. Additionally, structured programming being language dependent makes development quite slower.

[4] describes procedural programming as a set of procedures or a set of functions. In procedural programming a set of procedures are extensively used to perform tasks. Procedures are also called functions, routines or subroutines which basically are set of instructions or computational steps that has to be performed. Throughout the program, the procedures or functions can be called at any point and any number of times. These functions or procedures could also be called by other procedures. Most believe that the procedural programming and object oriented programming are mutually exclusive. However Irene Govender tells that John Lewis refused the myth “object-orientation and procedural concepts are mutually exclusive” [5][6].

With the necessity of representing real world objects in software program, object oriented programming came into existence. The core of object oriented programming is to create objects that have certain properties and methods [4]. According to [7], the object-oriented programming approach will makes it possible to break down a complex program into smaller and manageable modules that enables development easier to understand and collaborate by the members within a development team and also helps in better communication to those who provide requirements.

2.1 Object Oriented Programming Vs Structured Programming

The concepts of structured and object-oriented programming methods are not relatively new but these approaches used are very much still relevant in today's paradigm. Structured Programming also known as Modular Programming. According to [3], most of the development work of structured programming can be traced to the work done by Dijkstra. In Structured Programming, programs are divided into small self-contained functions. Structured Programming does not provide proper security to the data declared as there is no way of data hiding. OOC supports inheritance, encapsulation, abstraction, polymorphism. In Object Oriented Programming, Programs are divided into small entities called objects. Object Oriented Programming can solve any complex programs.

The main difference between OOP and structured programming is the way the code is written and processed by computer. In structured programming all functions are written globally and executed in a sequential manner whereas in OOC the execution is based on the event when the object is created. The code lines are processed one by one in structured programming. By making use of the concept of OOP works more dynamically, making calls according to the need of the code for a certain time and gives the ability to reuse the source code. Currently, object-oriented programming is more widespread due to the production gains in large scale. However, the structured language is not totally ruled out as its advantages for performance ends up compensating when it comes to software or hardware that need a much higher performance.

2.2 Object Oriented Programming Vs Procedure Oriented Programming languages.

POP, abbreviates as Procedural Oriented Programming and it mainly deals with programs and methods or functions. Programs are divided into functions and data declared in the code is global.

The main difference one can find in POP and OOP is that POP follows Top-down approach and OOP follows Bottom-up approach of programming. In POP, the main focus is on “how to solve a problem” i.e. on the procedure or structure of a program whereas in OOP main focus is on 'data security'. Hence, only objects are permitted to access the entities of a class. In POP large programs are divided into divisions called functions where each function performs different operations and whereas in OOP the program is divided into class and objects are created to invoke a class. However, talking about the property of data sharing, In POP the data once declared can be used by any member functions of the program. But in

OOP data is shared among the objects through the member functions and objects can be used to invoke a class. Languages such as C, VB, FORTRAN, Pascal uses only POP and does not support OOP but JAVA, C++, C#, Python supports both OOP and POP. Procedural Oriented Programming does not have support for the concept of overloading or polymorphism in it. On the other hand, Object Oriented Programming supports polymorphism, which means that using we use the same function name for performing different operations based on the signatures. It is possible to overload the functions, constructors, and operators as well in Object Oriented Programming. In Procedural Oriented Programming if there is some data that is to be shared among all the functions in the program, it can declared in a global scope outside all the functions. Where as in Object Oriented Programming the data members of the class will be accessed through the member functions of the class.

3. DATA FLOW IN OBJECT ORIENTED PROGRAMMING

In the figure shown below, the data and methods or functions of each class can be accessed by only a particular object instantiated for that class. Thus the class can be invoked by creating an object for that class and calling its functions to perform a particular function. The data used in each class can be accessed by only the functions or methods of that class and cannot be used by other classes though each function can communicate with other functions of other classes. In [8] Iqbaldeep Kaur Et al tell that the relations between different objects are expressed by the interactions in the form of message passing.

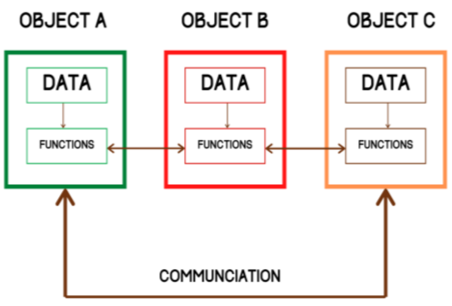

Fig -2 : Data flow in object oriented programming

4. CLASS DIAGRAMS

A class diagram is a static diagram and represents the static view of an application. It is used for visualizing, describing, and documenting different aspects of a system. By formalizing a system using class diagram, we need to show all of the entities, specifications and behaviour of the system as a class diagram [9]. Class diagrams provide a structural view of systems. Class diagrams provides the static structure of Object-Oriented systems. They provide support the design and also represent the basics of Object-oriented systems. They will identify what the classes are, and how they interrelate with each other and how they interact. These are useful in identifying the collection of classes, interfaces, associations, collaborations, and constraints and also called as a structural diagram of the program

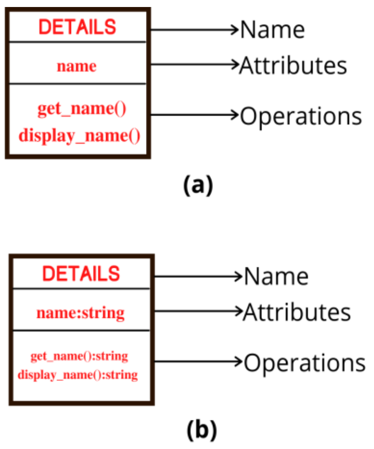

Fig -3: Class diagrams

Here, name refers to name of the class that appears in the first partition. Next, attributes represent the state of an object of the class and are descriptions of the structural or static features of a class, these appear in the second partition. Operations are found in the third partition. They define the way in which objects may interact. Operations are descriptions of behavioural or dynamic features of a class. In (a), the signature or the data type is not mentioned and are called as class without signature whereas in (b) the signature or the data type is mentioned and are called as class with signatures. A class may be involved in one or more relationships with other classes. The way of representing each relationship is unique. Consider the following image.

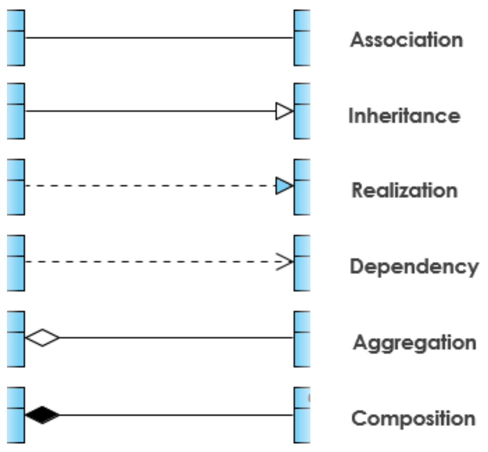

Fig -4: Types of Relationships

Summarizing the concepts of class diagrams, we can say that it is used in Describing the static view of the system. Describing the functionalities performed by the system. Gives structural information of all the methods used in programs and the detailed information of the attributes along with their signatures or data types.

5. ELEMENTS OF OBJECT ORIENTED PROGRAMMING IN PYTHON.

The concepts of object oriented programming rotate around the basic elements of the object oriented programming systems. However, the concepts of OOP can be divided into two subcategories, the elements and the features or characteristics of OOP. In this section we shall concentrate on the Elements of OOP and the later section (section 6) will concentrate on the features of OOP. Python being an object oriented programming language provides the support for these elements. These elements are class, objects, attributes, behaviour, methods and messages. Classes and objects are sometimes also considered features of object oriented programming languages, but however we will be considering them elements of object oriented programming as they form the basic building blocks of every object oriented programming language.

5.1 Classes

Classes enable tying up data and functionality together into a single unit. In [10] the author, Rushikesh S Raut describes the class as a user defined data type that holds the data members and member functions that operate on these data members. A class can be considered as a blueprint or template to create objects. Python enables the usage of classes. In Python, a class can be created using the “class” keyword, followed by a class name, a semicolon, and a block of code with increased indentation. This block of code can contain the data members and the member functions. A simple example that describes how a class definition looks in python is given below.

```
class student:
    def __init__(self, student_name, student_age):
        self.student_name = name
        self.student_age = student_age
    def display(self):
        print("Student Name = " + self.student_name)
        print("Student Age = " + str(self.student_age))
```

In this example we have created a class called student, that has data members as student_name and student_age, and the functions as __init__() and display(). The __init__() function is a special function, what in other object oriented programming languages is considered as constructors. We will discuss these special functions and constructors in section 7. So, this class called “student” can be a representation of a real student who has a name and age, in a programming language.

5.2 Objects

Generally, an object is any real world entity that has certain attributes and behaviour. However, in a programming language, an object is said to be an instance of a class. For instance, let us consider an example of a class called mobile phones. Samsung, Nokia, Motorola, Mi, etc. are objects of the class mobile. A class can have multiple objects. Each of these objects have their own copy of the class attributes. That is, they share the class attributes, but have their own individual values for these attributes. In python objects can be created by calling the class name as though we are invoking a function, that is by using a pair of parentheses. The concept of constructors comes handy here, when dealing with object oriented programming in general. But coming back to python, the following example explains how an object is created in python.

```
alex = person(“Alex”, 21)
bob = person(“Bob”, 34)
```

In this example, we have created two objects called alex and bob. Both of these objects are the instances of the person class that was used in the section 5.1. So, both of these objects have attributes name and age, but they have their own values stored in it. That is the object named alex has the values “Alex” and 21 for name and age respectively and similarly, the object bob holds the values “Bob” and 34 for name and age.

The attributes and functions associated with these objects can be accessed by making use of the dot(.) operator. That is to extract the name and age from the object alex, and to call the display function, we do as follows.

```
alex.age = 25
alex.display()
```

When we set alex.age=25, the value of the age attributed associated with the alex object get overwritten. Now, when the display function is called we get the output as

```
Name = Alex
Age = 25
```

5.3 Attributes

Attributes are the data values that are held in the class. As explained earlier, attributes are usually shared among the objects or instances. These attributes are called the class attributes. However, there is another type of attribute called the instance attributes. The instance attributes are those variables that are associated with only one object and are not shared. These attributes will belong only to this object's scope.

5.4 Behaviour

The behaviour of a class is how an instance or object of a class will operate or react when it is asked to perform something by some other class or an object or if its internal state changes. Behaviour is the only way in which an object can do anything to itself or have anything done to it. For example, let us consider the car class, here are some behaviours that this car class might have:

- Start the engine
- Stop the engine
- Speed up
- Change gear
- Stall

So, typically a behaviour is what an object can do to itself.

5.5 Methods

As discussed earlier, on OOP (Object-Oriented Programming), we have instances or objects. These objects have characteristics and behavior. Objects additionally can also be associated with methods. Methods are those functions that belong to the object. For example if we get back to the student class, we remember that there is one function which is called the display(). This display() function is invoked upon an object of that class.

```
class student:
    def __init__(self, student_name, student_age):
        self.student_name = student_name
        self.student_age = student_age

def display():
        print("Name = " + self.student_name)
        print("Age = " + str(self.student_age))

alex = person("Alex", 21)
alex.display()
```

So, the difference between a function and a method is that, unlike functions, the methods are called over objects of classes, and methods basically alter the state of an object, while functions don't do that.

5.6 Messages and Message Passing

One additional concept of object orientation is the use of messages and message passing. Objects communicate with each other by passing messages. When we say two or more objects are communicating, the idea behind is that they are passing messages to each other.

The concept of message passing comes from parallel processing or parallel computing, where it requires two or more processes to communicate with each other. Most object oriented programming languages make use of threads to allow message passing.

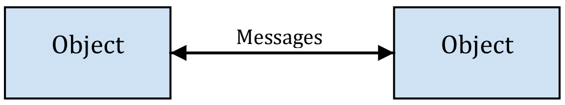

Fig-4: Message Passing

6. FEATURES OR CHARACTERISTICS OF OBJECT ORIENTED PROGRAMMING

Now that we know what the elements of object oriented programming are, we can now go ahead and understand the features or characteristics of OOP. The characteristics basically define the existence of object oriented programming. There are four features or characteristics of OOP. They are abstraction, polymorphism, encapsulation and inheritance.

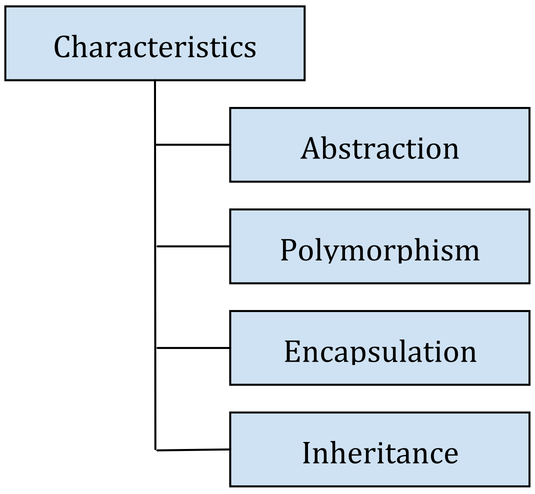

Fig-5: Features or characteristics of Object Oriented Programming

6.1 Inheritance

Inheritance is a property of object oriented programming that enables a class to inherit the characteristics of the super class or the parent class. To better explain this scenario, we can consider the example of the parent child relationship. We commonly see a son or a daughter replicate their parents, that is they inherit the features of their parents, this is how inheritance works. There is one parent class that has multiple subclasses, each of these subclasses will inherit the properties of the parent class and then add to it their own set of properties. Suppose there is a parent class called window, and two subclasses normal window and the scrolling window. Both of these subclasses have the common properties of windows, that is open and close, however the normal window has a hinge component in addition and the scrolling window has the Let us consider the following example to understand inheritance.

```
class person:
   def __init__(self, name, age)
      self.name = name
      self.age = age

   def display(self):
      print("Student name" + self.name)
      print("Student age" + str(self.age))


class student(person):
   pass

alex = student("Alex", 21)
alex.display()
```

In this example we have considered the student class. The student also has the properties of the person, which means that the student inherits the properties of the person. It is important to note that we created an object of the child class and not the parent class.

There are different types of inheritance supported in python. They are Single inheritance, Multiple inheritance, Multilevel inheritance, Hierarchical inheritance and Hybrid inheritance.

6.1.1 Single Inheritance

In single inheritance, the child class simply inherits the features or properties of a single parent class. Single inheritance will enable reusability of code as well as add new features to the existing code. The following diagram will depict the single inheritance.

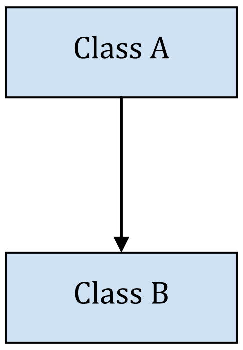

Fig-6: Single Inheritance

Let us now put this example in terms of python code.

```
class A:
    def display1(self):
        print("This is class A")

class B(A):
    def display2(self):
        print("This is class B")

obj = classB()
obj.display1()
obj.display2()
```

6.1.2 Multiple inheritance

Multiple Inheritance is said to be implemented when a class inherits the properties of more than one class. So, this is just how the kid inherits the properties from the mother as well as the father.

The following diagram will depict the multilevel inheritance.

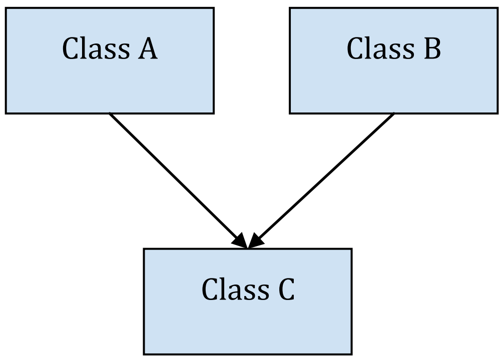

Fig-7: Multiple Inheritance

Let us now put this example in terms of python code.

```
class A:
    def display1(self):
        print("This is class A")

class B:
    def display2(self):
        print("This is class B")

class C(A, B):
    def display3(self):
        print("This is class C")

obj = classC()
obj.display1()
obj.display2()
obj.display3()
```

6.1.3 Multilevel Inheritance

In multilevel inheritance, one class inherits the properties of a parent class, adds to it it's own set of properties and this class is further inherited by another class. This kind of inheritance happens at different levels and so it is aptly called the multilevel inheritance.

The following diagram will depict the multilevel inheritance.

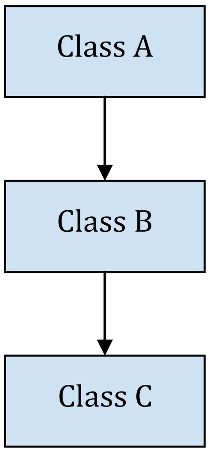

Fig-8: Multilevel Inheritance

Let us now put this example in terms of python code.

```
class A:
    def display1(self):
        print("This is class A")

class B(A):
    def display2(self):
        print("This is class B")
class C(B):
    def display3(self):
        print("This is class C")

obj = classC()
obj.display1()
obj.display2()
obj.display3()
```

6.1.4 Hierarchical Inheritance

In hierarchical inheritance, one class is inherited by more than one class. This case is similar to a family hierarchy, where one person can have multiple sons and daughters.

The following diagram depicts the hierarchical inheritance.

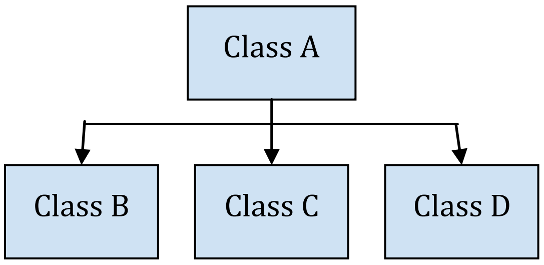

Fig-9: Hierarchical Inheritance

Let us now put this example in terms of python code.

```
class A:
    def display1(self):
        print("This is class A")

class B(A):
    def display2(self):
        print("This is class B")

class C(A):
    def display3(self):
        print("This is class C")

class D(A):
    def display4(self):
        print("This is class D")

obj1 = B()
obj2 = C()
obj3 = D()
obj1.display1()
obj1.display2()
obj2.display3()
obj3.display4()
```

6.1.5 Hybrid Inheritance

When there is more than one kind of inheritance involved, this kind of inheritance is called the hybrid inheritance. The following diagram will depict the hybrid inheritance that involves multilevel inheritance as well as the hybrid inheritance.

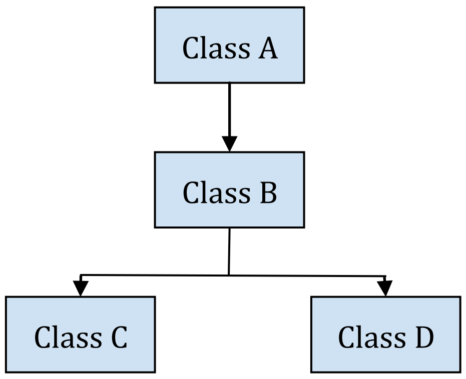

Fig-10: Hybrid Inheritance

Let us now put this example in terms of python code.

```
class A:
    def display1(self):
        print("This is class A")
class B(A):
    def display2(self):
        print("This is class B")

class C(B):
    def display3(self):
        print("This is class C")

class D(B):
    def display4(self):
        print("This is class D")

obj1 = C()
obj1.display1()
obj1.display 2()
obj1.display 3()

obj2 = D()
obj2.display1()
obj2.display2()
obj2.display4()
```

6.2 Abstraction

Abstraction is the process of showing only the necessary details and hiding every other detail, that is not necessary to the user. For example, suppose we consider an example of a car, the user of the car that is the driver is required to know how to accelerate, brake, change gears, raise and lower the window shutter, turn on and turn off the indicators and so on. However, it is not required for him to know how each of these individual components are devised, that is how the window mechanism works from the inside or how the engine works.

The problem with abstraction is that, an abstract data type will define a kind of black box, once it has been defined, it does not interact with the rest of the program. This can lead to inflexibility [11].

This feature of hiding what the user is not required to know is called abstraction. Abstraction is helpful because this will reduce the complexity of the program.

Let us consider the following example of a student class to explain abstraction

```
from abc import ABC
class student(ABC):
    def studid(self, id, name, age):
        pass
class juniorstudent(student):
    def studid(self, id):
        print("Student id is" +str(id))

st = juniorstudent()
st.studid(1101)
```

In the above example we have imported the class ABC. The class student becomes an abstract class as we have defined the abstract method in it. The abstract methods are not supposed to contain any implementation, hence we make use of a pass statement along with the abstract method. Hence the studid becomes the abstract method of the student class. We have another class called the junior student that inherits the student class.

There are two important things we need to remember about an abstraction in python, one - an abstract class can contain both normal methods as well as abstract methods, and two - an abstract method cannot be instantiated, that is, it is not possible to create an object of the abstract class.

6.3 Encapsulation

One of the important characteristics of object oriented programming language is the encapsulation. Encapsulation is the process of wrapping up of data and the functions that operate on this data into one single unit. Encapsulation will apply restriction on the access to data members and member functions and prevents any accidental changes to the data. Class is one such encapsulated entity that will encapsulate within it the data members and the member functions.

To better understand the concept of encapsulation let us consider the example of a educational institution. This institution consists of a number of departments and branches. Suppose there is a necessity where the faculty of one department, say department A needs to view the details of a student of another department, say department B. This faculty cannot simply go and fetch the details of that student. He will have to ask the head of the department B in which the student is a studying and ask for the required details. This feature itself is the encapsulation. There was a restriction applied on the access of the details of the student by the faculty of another department.

The following figure shows the encapsulation process.

Fig-11: Encapsulation

In python it is possible to set restrictions to access variables and methods. We make use of private attributes. Private attributes in python are denoted by prefixing the attributes using the single underscore(_) or double underscore (__). Consider the following example.

```
class student:
    def __init__(self):
        self.__id = 1101

    def display(self):
        print("Student id is "+str(self.__id))

    def setstudid(self, id)
        self.__id = id

obj1 = student()
obj1.display()

obj1.__id = 1102
obj1.display()

obj1.setstudid(1102)
obj1.display()
```

Now, when we tried self.__id = 1102, the value of the variable will not change, it will still be the same old value. However, when we tried setstudid (1102), the value of the variable changes.

6.4 Polymorphism

In object oriented programming, polymorphism is one of the most useful characteristics. Polymorphism enables data to be processed in several different ways. That is polymorphism enables a single action to be performed in several different ways. The word polymorphism itself can be split into two halves - poly meaning many and morphism meaning forms or types.

Let us consider the plus (+) operator. The plus operator while operating on two numeric (integer or floating point) values will act as an addition operator, it adds the two values. But when the plus operator is used with two string values, it acts as a concatenation operator. That is it joins the second string to the end of the first string. There is a similar case with the asterisk (*) operator. While working with numeric operands, the * will act as a multiplication operator, but while working with one string value and one integer value, the * acts as a string replication operator. This feature using which an operator behaves differently according to the type of data passed to it, is called operator overloading.

Python also supports function polymorphism. Function polymorphism means the same function operates on different data types. There polymorphic functions that are built in as well as user defined polymorphic functions. Let us consider the len() function. The len() function works well with strings, lists, tuples and dictionaries as well. While working with strings, it will return the length of the string. While working with lists and tuples it will return the number of items and while working with dictionaries it will return the number of keys. Consider the following.

```
print(len("Hello World") #output: 11
print(len([1, 2, 3, 4, 5])) #output : 5
print(len({a:1, b:2})) #output : 2
```

Similarly let consider the add() function which can be defined as .

```
def add(a, b):
    return a+b

print(add(2, 3)) #Prints 5
print(add("good", " day") #prints good day
```

The add function in the above example uses the plus operator. So as discussed earlier, the value obtained on using a + operator depends on the data type of the value passed. So this function add() behaves according to the value that has been passed to the function. If numeric values are passed it will return the sum of the values and if strings are passed it will return the concatenated string.

Polymorphism can also be used while creating the class methods. Suppose we have two classes, say student and teacher. Both the student and the teacher have a similar function called display_info(). When a call is made to this function it depends to which class the object belongs to and suitable operation is performed.

```
class Student:
    def __init__(self, id):
        self.id = id
    def display_info(self):
        print("Student id is " + str(self.id))

class Teacher:
    def __init__(self, id):
        self.id = id
    def display_info(self):
        print("Teacher id is " + str(self.id))

obj1 = Student(1101)
obj2 = Teacher(121)
for x in (obj1, obj2):
    x.display()
```

We will get the following output when we run the above code.

```
Student id is 1101
Teacher id is 121
```

Since python also supports inheritance, there might be a necessity to redefine certain functions of the parent class in the child class. This feature is called the method overriding. Polymorphism makes it possible to access the overridden methods that have the same name as the one in the parent class. Let us consider one simple example to understand the concept of method overriding. We have a parent class called animal and a child class called dog. The following code will implement method overriding on a method called the eat().

```
class animal:
    def __init__(self, name):
        self.name = name
    def eat(self):
        pass
class dog:
    def __init__(self, food):
        super().__init__("Jim")
        self.food = food
    def eat(self):
        print("Jim eats "+self.food)
```

7. CONCEPTS OF CONSTRUCTORS

When we are hiding the representation of a type, then there must be some way in which we initialize values to the variables. A usable solution would be to have the user call a function to initialize the value. Such a function is called a constructor. Using a constructor the allocation of memory as well as initializing a value to the variable becomes a single operation rather than it being two different operations [11]. A constructor is a unique method that is invoked when an object of the class is created. In other words, the work of a constructor is to initialize the objects of a class. Understanding the name constructor, it is called so because they construct the values of the data members of a class.

In most Object Oriented Programming Languages, like java and C++, constructors are defined by using the class name as a function name and they are called by calling the class names as function calls. However in python, constructors are created using the __init__() method. The __init__() method is basically the definition of the constructor, as in other object oriented programming languages. However, while calling the constructor, we use the class name as usual.

Consider the following example of a constructor. In this example we have considered a student class. The __init__() method will take one mandatory argument called the self-variable followed by the two other parameters, student_name and student_id. The values to these parameters are called during the function call.

```
class students:
    def __init__(self, student_name, student_id):
        self.student_name = student_name
        self.student_id = student_id
    def display(self):
        print("Name = " +self.student_name)
        print("Id = " + str(self.student_id))

alex = students("Alex", 101)
alex.display()
```

The word "self" is used to represent the instance of class. By using the keyword "self" we can access the attributes and methods of a class in python. It is important to use the self-parameter inside an object's method if have to persist value with the object.

Constructors are of two types, they are the default constructor and the parameterized constructor. The default Constructor is simply a constructor which is without any arguments. Consider the following example.

```
class demo:
    n=10
    def  display(self):
        print(self.n)

obj=demo()
obj.display()
```

This will give an output 10, because we initialized the variable with the value 10 by default.

A Constructor with parameters is known as a parameterized constructor. For example consider the following snippet of code.

```
class demo:
    def  __init__(self,data):
        self.data=data
    def display(self):
        print(self.data)

obj=demo(55)
obj.display()
```

We have passed a value 55 when we created the object of the class, so this value is initialized to the data variable for that object. Hence when we call the display() function we will get an output as 55.

Constructors also verify that there are enough resources for the object to perform any task.

8. CONCEPT OF DESTRUCTOR

In cases where certain objects of the class are not of significance, in those cases one might have to perform a complementory oprtstion of the constructor, that is these objects have to be cleaned up after the object has been used for the last time. The Destructor in a program plays the role of reversing the operation performed by the constructor, that is used for clearing the object of the class that was created by the constructor.

Python has a garbage collector that handles memory management. Destructors are defined by '__del__()' in python. When it is called ,all the reference to the object gets deleted, which is also known as garbage collection.

Let us consider an example to see how destructors work in python. We have considered the Computer class and the __del__() function is called on one of the objects of the class.

```
class Computer:
      def __init__(self):
            print("Instance of computer created")
      def__del__(self):
            print("destructor called, computer destroyed")

laptop = Computer ()
del laptop
```

The output is displayed, even though there is no method that is called. Just like the constructors, the destructors are also called automatically .The del keyword is used to delete an object with no further references for an object,it will automatically invoke this method to clean up the memory occupied by the object. Exceptions explicitly raised in case the __del__ is ignored.

9. ADVANTAGES AND DISADVANTAGES OF OBJECT ORIENTED PROGRAMMING

Just by the name we can know that it breaks the program based on the basic of the object in it. It's aim is to bind the data and functions together to operate on them.

Some advantages of object oriented programming include:

- Object oriented programming is modular, this results in high maintainable code. It is extensible, as objects can be extended to include new attributes and behaviors. Objects can be reused in other projects as well. Due to extensibility, modularity and the reusability, object oriented programming provides improved and better software development productivity .
- Object oriented software is also easier to maintain. Since the design is modular, part of the system can be updated if any issues arise without need of any large scale changes.
- Object oriented programming languages come with rich libraries of objects and code development during projects , reuse also enables faster development .
- The reusability of code makes it lower the cost of development, much effort is seen in object oriented analysis and design ,which lowers the overall cost of development.
- Faster development and lower cost allows more time and resources to be used in verification of software.Object oriented programming tends to result in higher quality software.

Some disadvantages of object oriented programming include:

- Object oriented programs typically involve more lines of code than procedural programs.
- Object oriented programs are typically slower than procedural programs, hence require more instruction to execute .
- One should know the detailed knowledge of the software being developed is needed in order to create objects.

10. CONCLUSION

In this paper we have seen what Object Oriented Programming or OOP as it most commonly known is, and some salient features and concepts of object oriented programming. Object oriented programming is the programming paradigm of the future and most programming langugaes like C++, Java, PHP, python are based on OOPS. OOP is advantageous for a fact that it breaks down a program into smaller and effective modules which enables better understanding of the code. Python being an object oriented programming language provides all the features of OOP such as inheritance, abstraction, polymorphism and encapsulation. We have choosen python to convey these concepts in the paper due to its easy and understandable syntax.

REFERENCES

[1] Akshansh Sharma, Firoj Khan, Deepak Sharma, Dr. Sunil Gupta, "Python: The Programming Language of Future". IJIRT, May 2020, Volume 6 Issue 12.
http://ijirt.org/master/publishedpaper/IJIRT149340_PAPER.pdf

[2] Dr. M. Selvakumar Samuel, "An Insight into Programming Paradigms and Their Programming Languages". Journal of Applied Technology and Innovation vol. 1, no. 1, (2017), pp. 37-57. https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2018/07/17035708/2017_Issue1_Paper4.pdf

[3] Hunt, Karl. (1979). An introduction to structured programming. Behavior Research Methods. 11. 229-233. 10.3758/BF03205654.
https://www.researchgate.net/publication/225724365_An_introduction_to_structured_programming

[4] Mala Dutta, "Basic Concept of Object Oriented and Procedure Oriented Programming". International Journal of Information and Tech nology (IJIT) – Volume 2 Issue 4, Jul-Aug 2016

http://www.ijitjournal.org/volume-2/issue-4/IJIT-V2I4P1.pdf

[5] Govender, Irene. (2010). From procedural to object-oriented programming (OOP) - An exploratory study of teachers' performance. South African Computer Journal. 46.

https://www.researchgate.net/publication/236156223_From_procedural_to_object-oriented_programming_OOP_-_An_exploratory_study_of_teachers'_performance

[6] Lewis, John. (2000). Myths about object-orientation and its pedagogy. ACM Sigcse Bulletin. 245-249. 10.1145/331795.331863.

https://www.researchgate.net/publication/221538451_Myths_about_object-orientation_and_its_pedagogy

[7] Yilmaz, Rahime & Sezgin, Anil & Kurnaz, Sefer & Arslan, Yunus Ziya. (2018). Object-Oriented Programming in Computer Science. 10.4018/978-1-5225-2255-3.ch650.

https://www.researchgate.net/publication/317957956_Object-Oriented_Programming_in_Computer_Science

[8] Iqbaldeep Kaur, Navneet Kaur, Amandeep Ummat, Jaspreet Kaur, Navjot Kaur, "Research Paper on Object Oriented Software Engineering". IJCST Vol. 7, Issue 4, Oct - Dec 2016. http://www.ijcst.com/vol74/1/8-iqbaldeep-kaur.pdf

[9] Souri, Alireza & Shariffloo, Mohammad & Norouzi, Monire. (2011). Formalizing class diagram in UML. 10.1109/ICSESS.2011.5982368.

https://www.researchgate.net/publication/235635905_Formalizing_class_diagram_in_UML

[10] Mr. Rushikesh S. Raut. "Research Paper on Object-Oriented Programming", International Research Journal of Engineering and Technology (IRJET). Volume: 07 Issue: 10 | Oct 2020.

https://www.irjet.net/archives/V7/i10/IRJET-V7I10247.pdf

[11] Stroustrup, Bjarne. (1988). What is "Object-oriented Programming"?. Software, IEEE. 5. 10 - 20. 10.1109/52.2020.

https://www.researchgate.net/publication/3246605_What_is_Object-oriented_Programming