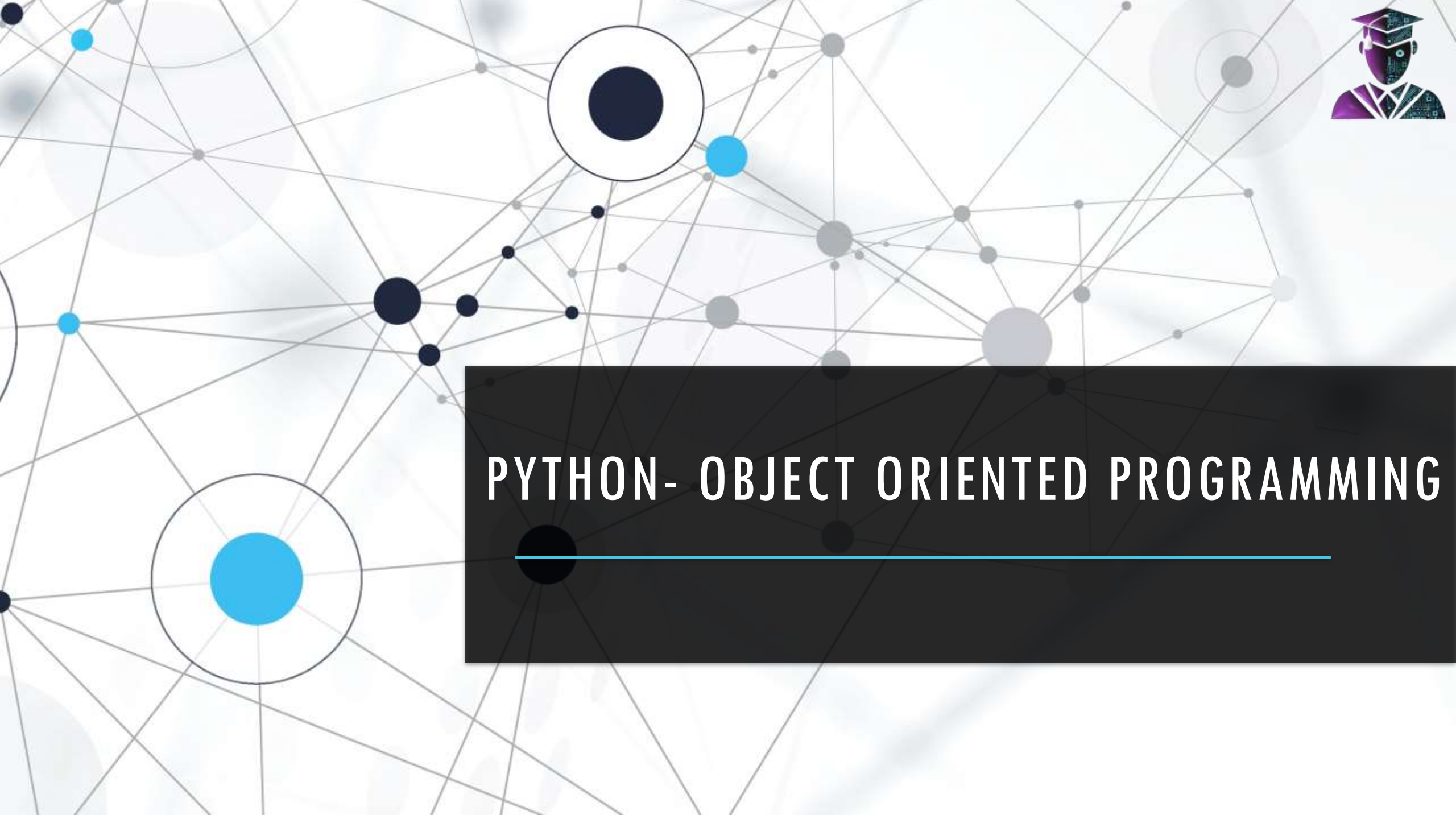
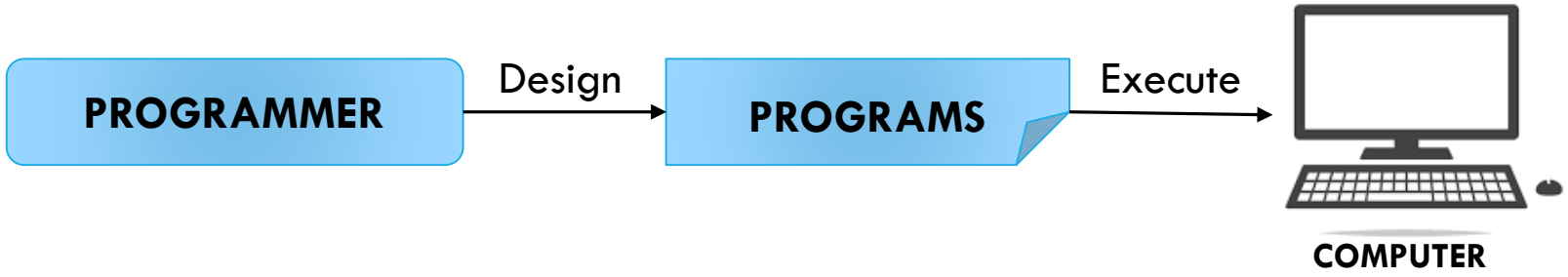


PYTHON- OBJECT ORIENTED PROGRAMMING

TYPES OF PROGRAMMING LANGUAGE

Programming is a process, executed by a Programmer, for creating set of instructions (Programs) to instruct a computer on how to perform a task.

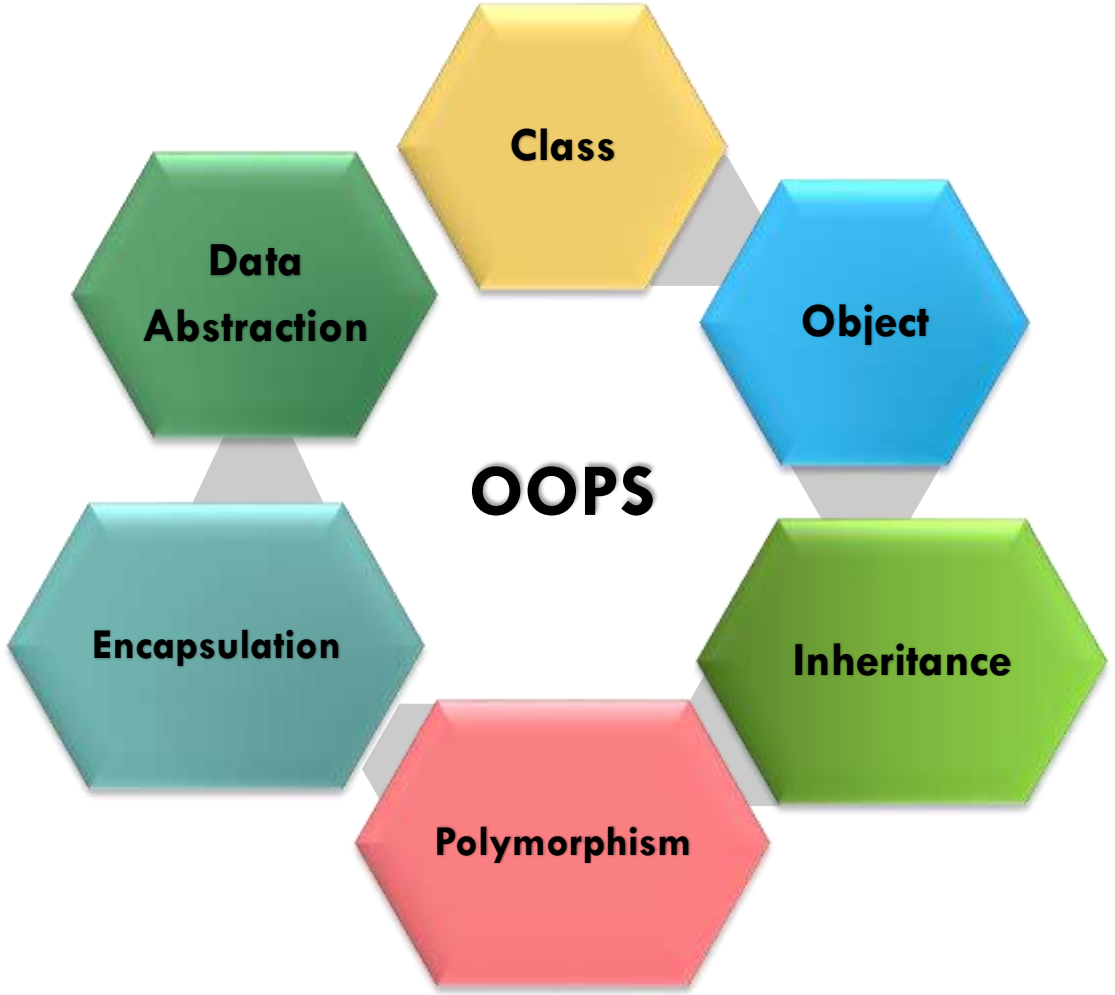


There are different ways to approach a task. There are major 2 types of Programming Paradigm:

1. **Procedural Programming Language:**
Procedural programming implements a set of instructions to inform the computer on what to do in a step-by-step manner.
Example- C is a Procedural Programming Language.

2. **Object-oriented Programming Language:**
Object-oriented programming is the problem-solving approach and used where computation is done by using objects.
Example- Python is an Object-oriented Programming Language.

FEATURES OF OBJECT-ORIENTED PROGRAMMING(OOPS)



CLASS

- The building block of Python that leads to Object-Oriented Programming is a Class.
- The **Class** can be defined as a **collection of Objects.**
- It is a logical entity that has some specific attributes and methods.

LOGIC

```
#Initialize the Class
class Dog:
    Attributes+Methods
```

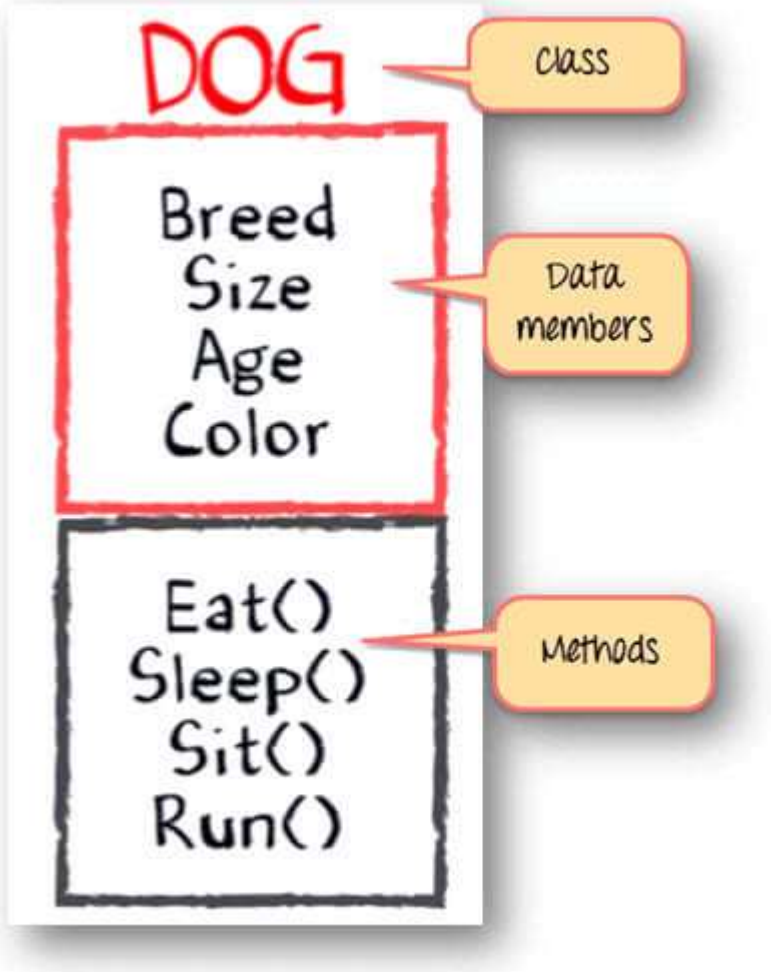
DOG CLASS

OBJECT

- Object is a collection of data members (attributes) with associated behaviors (methods).
- An Object is an instance of a Class.
- For example “Dog” is a class, which has some attributes like Breed, Size, Age, colour and behaviors like Sit, Run, Sleep, and Eats.

LOGIC

```
#Initialize the Class
class Dog:
        Breed, Size, Age, color
        Eat(),Sleep(),Sit(),Run()

#Create object “Dog1” with following attributes
Dog1=Dog(“Bulldog”,31,1,”brown”)

#call a method for Dog1 object
Dog1.Eat()
```



PYTHON SYNTAX

For instance- **Tom** is an object for **Person** class with properties like a name and age and behaviors such as eating and speak.

All classes have a function called **__init__()**, which assign attributes to different objects.

There must be a special first argument self in all of method definitions which gets bound to access variables in class.

Attributes
Name
Age

TOM (Object)
Person (Class)

Behaviors
Speak
Eating

SYNTAX

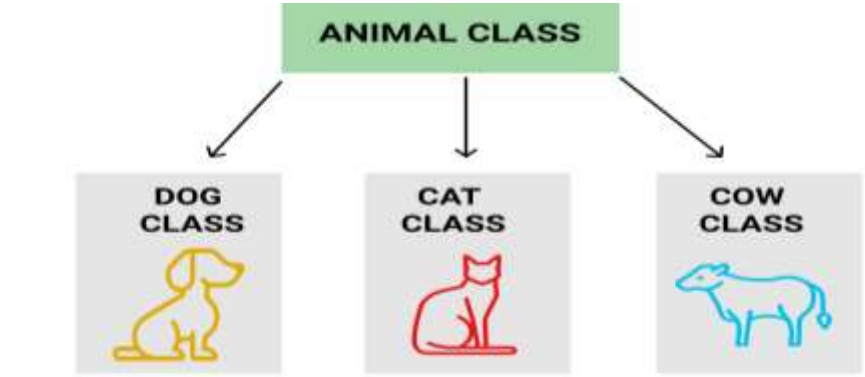
```
#A class representing a person
class Person:

  #Intialize the attributes
  def __init__(self, name, age):
    self.name = name
    self.age = age

  #define a method
  def Speak(self):
    print("Hello my name is " + self.name)

  def Eating(self):
     print("Let's Eat")

Tom = Person("Tom",30)
Tom.myName()
```

INHERITANCE

Inheritance allows us to define a class that inherits all the methods and properties from another class.

- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.

Dog, Cat, Cow are Derived Class of Animal Base Class.



SYNTAX

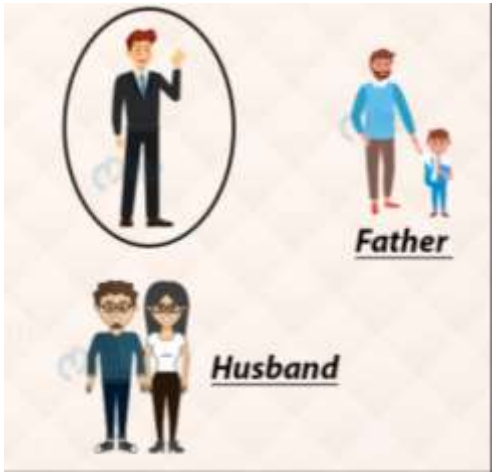
```
class Animal:

  def __init__(self, breed, color):
    self.breed = breed
    self.color = age

  def  myBreed(self):
    print("Hello my name is " + self.breed)

#define child class which inherit the properties
And methods from Animal class
class Dog(Animal):
  pass

dog1 = Dog(“Bulldog","”Brown”)
dog1.myBreed()
```

*Note- **pass** keyword is used when any other methods are not added to the class.

POLYMORPHISM

- Polymorphism contains two words "poly" and "morphs". Poly means many, and morph means shape.
- By polymorphism, we understand that one task can be performed in different ways. Python allows different classes to have methods with the same name.

Example- A man at the same time is a father and husband. So the same person posses different role in different situations



SYNTAX

```
class Father:

  def __init__(self, name):
    self.name = name

  def  myRole(self):
    print(self.name+“ is a Father”)

class  Husband:
  def __init__(self, name):
    self.name = name

  def  myRole(self):
    print(self.name+“ is a Husband”)

f1 =Father(“Tom”)
f1.myRole()

h1 =Husband(“Tom”)
h1.myRole()
```

ENCAPSULATION

Encapsulation is the mechanism for restricting the access to some variables, this means, that the internal representation of an object can't be seen from outside of the objects definition

Syntax of Variable:

```
class Encapsulation:
      def __init__(self, name):
          self.name=name (Public)
          self._name=name (Protected)
          self.__name=name (Private)
```

Class member access specifier	Access from own class	Accessible from derived class	Accessible from object
Private member	Yes	No	No
Protected member	Yes	Yes	No
Public member	Yes	Yes	Yes

DATA ABSTRACTION

- Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonyms because data abstraction is achieved through encapsulation.

- Abstraction means hiding the complexity and only showing the essential features of the object.

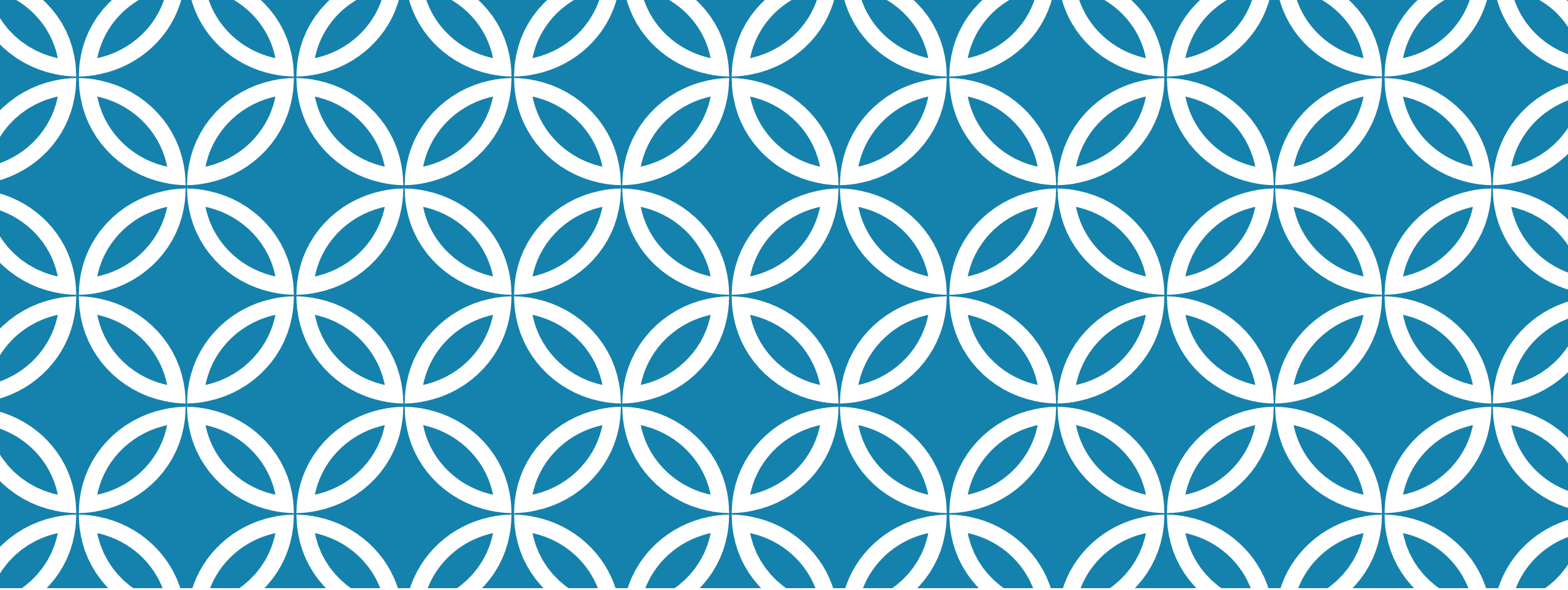
ABSTRACTION EXAMPLE

- Consider, a man goes to ATM machine for money withdrawal, he pressed withdraw button and enter the amount, but he does not know about how on pressing the withdraw the money is coming. This is what abstraction is.

- Here, create an abstract class Machine that has an abstract method withdraw() from Bank. There are two child classes FatherPayment and MotherPayment derived from Machine that implement the abstract method withdraw() as per their functionality.

SYNTAX

```
from bank import Bank,withdraw
class Machine(Bank):

  def __init__(self,amount):
    self.amount= amount

  def withdraw(self,amount):
    pass

#define child class
class FatherPayment(Machine):
 def withdraw(self,amount):
    print("Total withdrawal is" + self.amount)

class MotherPayment(Machine):
 def withdraw(self,amount):
    print("Total withdrawal is" + self.amount)

obj = FatherPayment()
obj.withdraw(100)
obj1 = MotherPayment()
obj1.withdraw(200)
```



THANK YOU