OBJECT ORIENTED PROGRAMMING

- Class
- Object
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- Attributes
- Methods

PRESENTED BY:

ONYINYE OGUEJIOFOR

Class

Class is a *blueprint* for creating an object

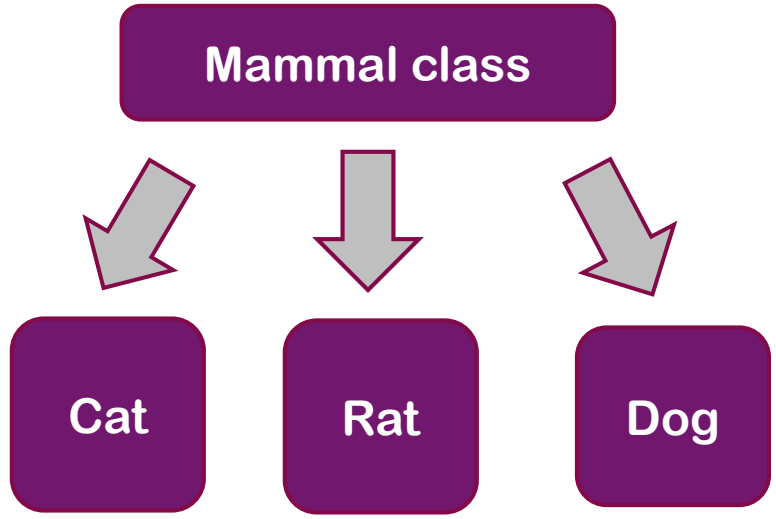
Class is declared using : '*class keyword*', 'name of the class' and block of attributes and methods.

Class Name: is capitalize to distinguish it from variable name.

Self parameter: it ensures that all actions will be performed by an object.

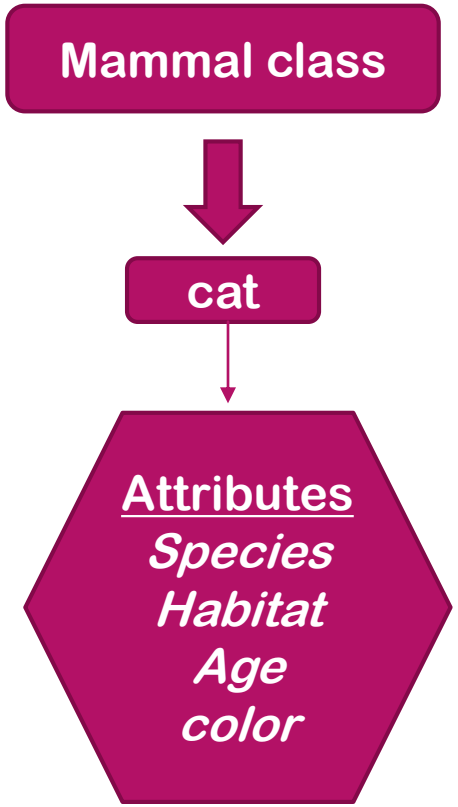
__init__: initializes data attribute when a method is executed.

```
class Mammal:
    def __init__(self, Species):
        self.__species = species

    def show_species(self):
        print (" I am a",self.__ Species)

    def make_sound(self):
        print(" Grrrr")
}
```



Attributes

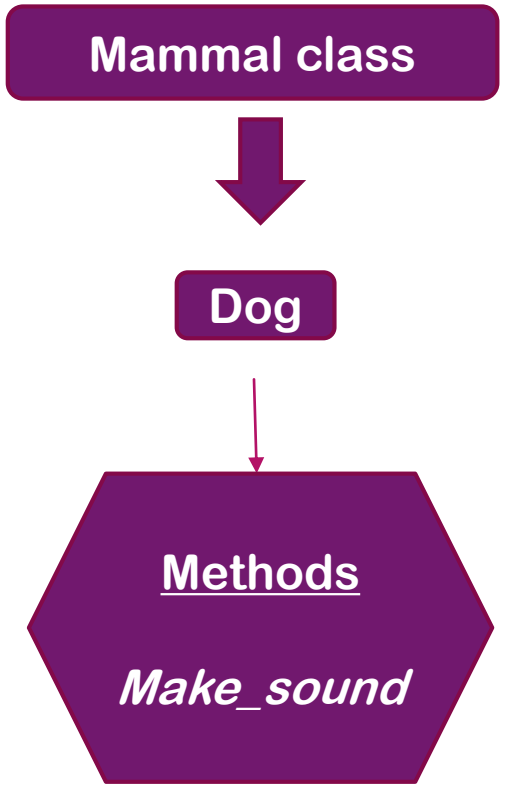
- An Attribute is a variable attached to a class and object .
- They are things an object has.
- It can also be said to be data or properties that describes an object.
- An attribute code has an equal sign.
- Example(Mammals): color, name, age etc




```
class mammal:
    def __init__(self,
Species,habitat,age):
        self.species = species
        self.habitat = habitat
        self.age = 0
```

Methods

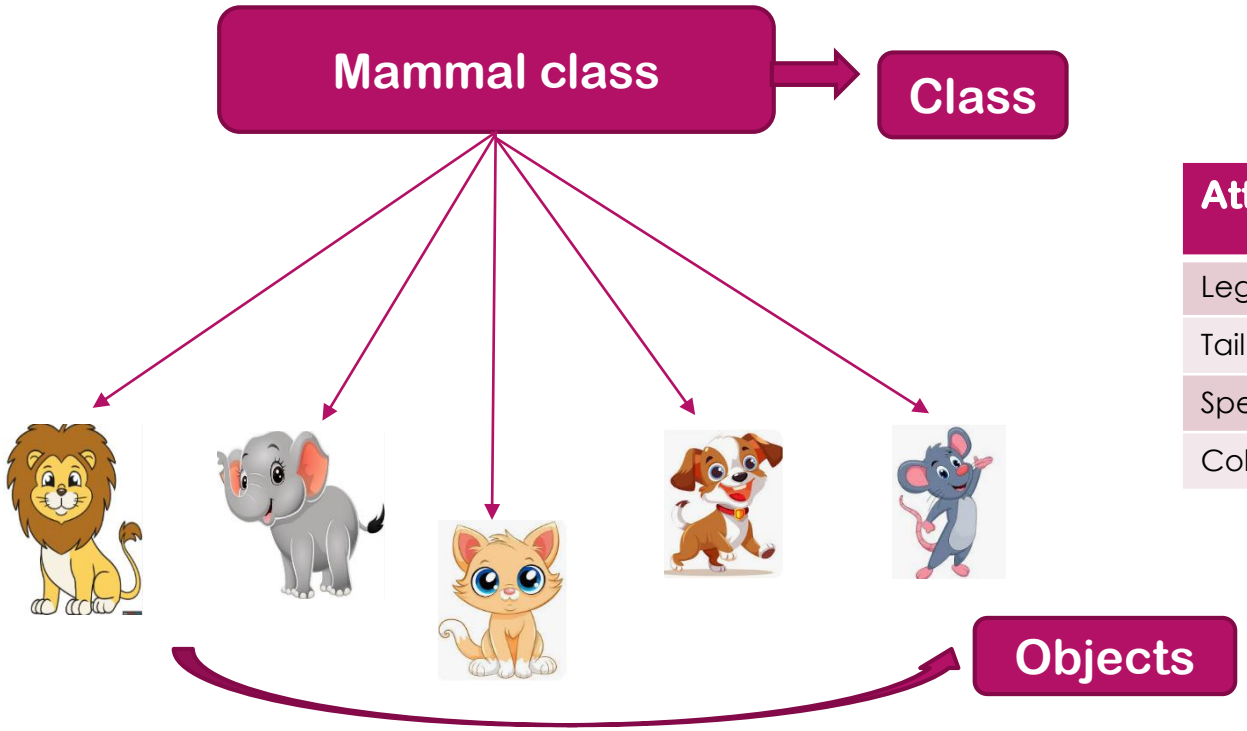
- Methods are functions defined within a class that defines the objects behavior.
- They are things an object can do.
- Methods are functions or actions.
- Methods can be identified by dot notion.
- Examples(Mammal): moves, sound, fights etc.




```
class mammal:
    def __init__(self, Species):
        self.__species = species

    def make_sound(self):
        print(" Woof Woof")
```

Objects

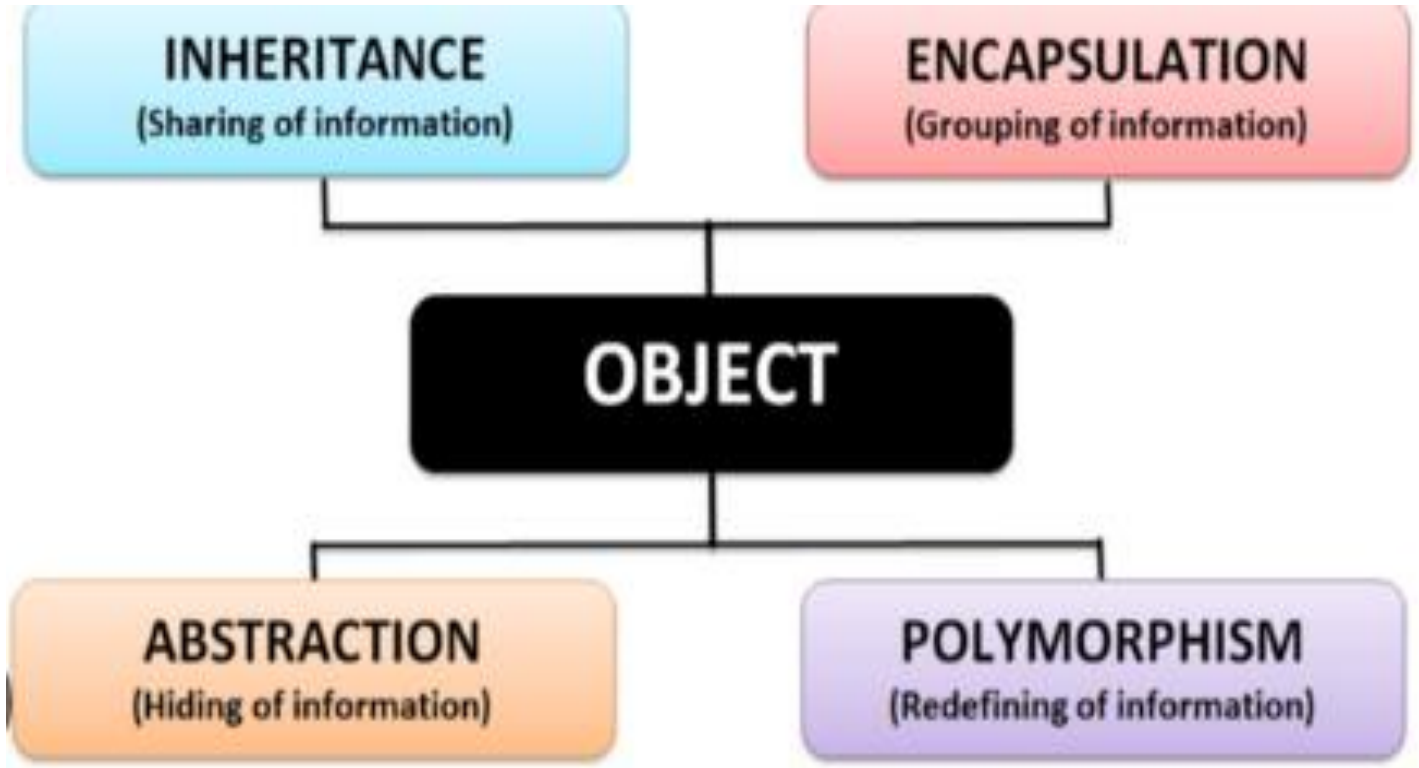
- An Object is an instance of a class.
- Objects is made up of attributes(data) and methods(functions)



Attributes	Methods
Leg	Make_sound
Tail	Sleep
Species	Eat
Color	

```python
class Mammal
  def  __init__(self,legs,tail,species,color):
      #Attributes
      self.legs = legs
      self.tail = tail
      self.species = species
      self.color = color

    # Methods
    def make_sound(self):
        print('woof!')
    def sleep(self):
        print('Is sleeping')
    def eat(self,food):
        print('Is eating')

    #Object
     Object1= Mammal(4,True,"Lion","Brown")
     Object2= Mammal(4,True,"elephant","purple")
     Object3= Mammal(4,True,"Cat","Brown")
     Object4= Mammal(4,True,"Dog","Brown")
     Object5= Mammal(4,True,"Rat","purple")
```

Pillars of Object Oriented Programming



Encapsulation

- The bundling of data(attributes) and methods that operate on that data into a single unit or class.
- It is a form data security
- It also refers to combining of data and code into a single object.
- Protects your class from accidental changes and deletions.
- Single preceding underscore = off limit while double preceding underscore = private member & returns error when accessed.
- To retrieve use the Get method
- To modify use the Set method

```python
class Mammal:
    def __init__(self, species, sound):
        self._species = species  # Encapsulated attribute
        self._sound = sound      # Encapsulated attribute

    def make_sound(self):
        print(f"A {self._species} makes the sound: {self._sound}")

    # Getter method for species
    def get_species(self):
        return self._species

    # Setter method for species
    def set_species(self, species):
        self._species = species
```

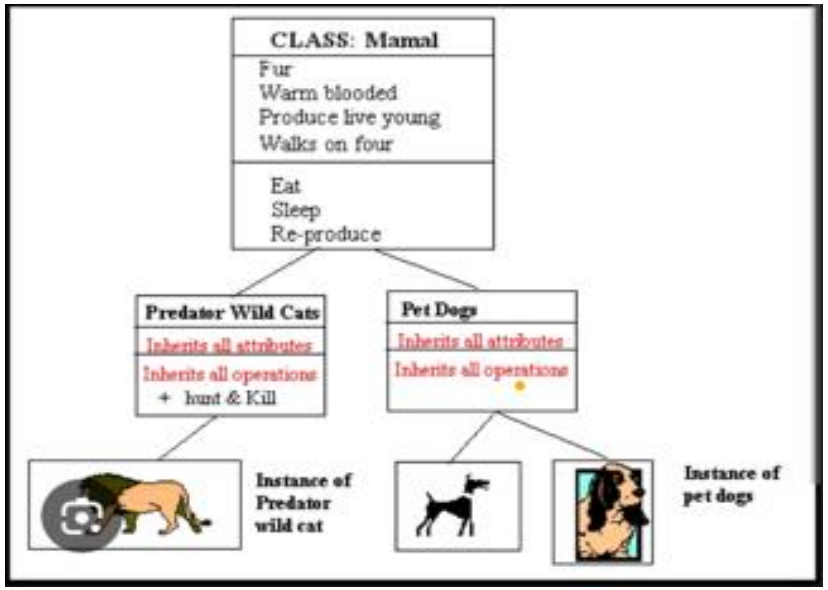
Abstraction

- Abstraction allows the creator to hide the complex details of an object.
- It shows the necessary features of an object.
- For Instance turning on a laptop or starting a car.

```python
from abc import ABC, abstractmethod

class Mammal(ABC):
    def __init__(self, species):
        self._species = species

    @abstractmethod
    def make_sound(self):
        pass

class Dog(Mammal):
    def make_sound(self):
        return "Woof"

# Instantiate a Dog
dog = Dog("Canine")
print(f"A {dog._species} says: {dog.make_sound()}")
```

Inheritance

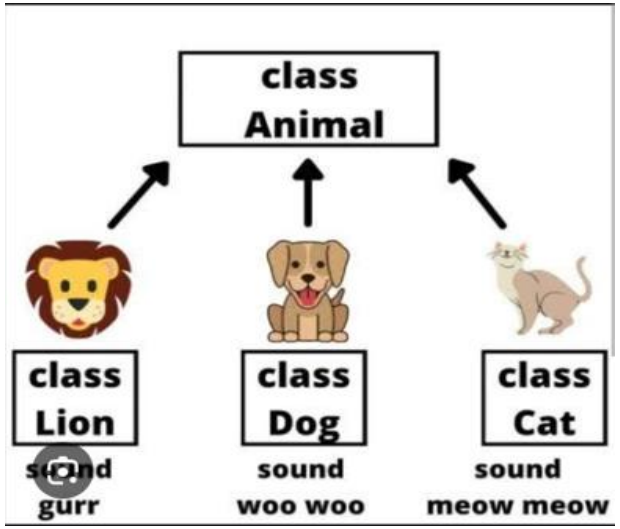
- Inheritance allows a new class to possess the members of the class it extends.
- Inheritance involves a Super/ general/parent class and a sub/specialized/ child class.




```python
class Mammal:
    def __init__(self, species, sound):
        self.species = species
        self.sound = sound

    def make_sound(self):
        print(f"{self.species} makes a {self.sound} sound")


class Dog(Mammal):
    def __init__(self, breed):
        super().__init__("Dog", "bark")
        self.breed = breed

class Cat(Mammal):
    def __init__(self, breed):
        super().__init__("Cat", "meow")
        self.breed = breed

dog.make_sound()
cat.make_sound()
```

Polymorphism

- The ability of objects to take on different forms.
- It allows subclass to have methods with the same name as methods in the superclass.
- There is usually an inheritance where polymorphism exist.




```python
class Mammal:
    def __init__(self, Species):
        self.__species = species

    def show_species(self):
        print (" I am a",self.__ Species)

    def make_sound(self):
        print(" Grrrr")

class Dog(mammal):
  def __init__ (self):
      mammal.__init__(self,'Dog'):

  def make_sound (self):
      print('woof! woof!')
```

Benefits of OOP

- Allows Reuse of codes
- Offers Security
- Ensures flexibility
- Locates and fixes problems effortlessly
- Makes changes seamlessly

Conclusion

Object oriented programming is a programming methodology that centers on objects, it is a computer programming that focuses on how to make codes simple and clean.

It requires thinking about the structure of the object before writing down the codes.

Examples of programming language that follows OOP are Python, C++ and Java.

Handle

Github handle: https://github.com/oguejioforO

Linkedin handle: https://www.linkedin.com/in/onyinye-oguejiofor-817299197