OOPS in PYTHON

1]. **CLASS :-** A class is a blueprint for the object. We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colours, size etc.

Example :-

```
Class parrot:
pass
```

Class keyword to define an empty class parrot.

2]. **OBJECT :-** An object (instance) is an instantiation of a class. When class is defined, only description for object is defined, no memory or storage is allocated.

Example :-

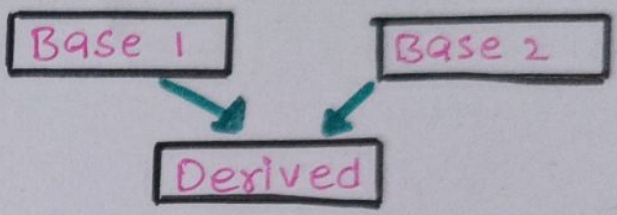
```
Class Vehicle:
def__init__(self, brand, model, type):
Self.brand = brand
Self.model = model
Self.type = type
Self.gas_tank_size = 14
Vehicle_object = Vehicle('Honda', 'truck')
```

3]. **INHERITANCE :-** Inheritance is a way of creating a new class for using details of an existing class without modifying it.

Example:-

```
Class Parent ( ):
def first( self ):
Print('first function')
Class child(Parent ):
def Second(self ):
Print(' Second function')
Ob = Child ( )
Ob. First( )
Ob. Second( )
```

Output:- First function
Second function



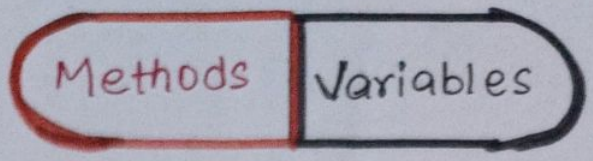
4]. **ENCAPSULATION** :- Using OOP in python, we can restrict access to methods and variable. This prevent data from direct modification which is called as Encapsulation.

Example:-

```
Class Employee:
def__init__(self, name ,salary , Project ):
self.name = name
self.salary = salary
self. Project = Project
def show ( self ):
Print("Name:" , self.name, 'Salary':, self salary)
def work (self):
Print(self.name, ' is working on', self. Project)
# Creating object of a class.
emp = Employee('Ram', 10,000, 'Python')
# Calling Public method.
emp. show ( )
emp. work( )
```

Output:-

Name : Ram Salary : 10,000
Ram is working on python.



5]. **ABSTRACTION :-** Abstraction is used to hide the internal functionality of the function from the users. Abstraction can be achieved by using abstract classes and interfaces.

Example:-

```
From abc import ABC , abstractmethod
class Absclass (ABC):
def Print(self,x):
Print("Passed value :", x)
def task ( self ):
Print("We are inside Absclass task")
class test_class (Absclass ):
def task (self ):
Print("We are inside test_class task")
# Object of test_class Created.
test_obj = test_class( )
test_obj.task ( )
test.obj.Print(100)
```

Output :- We are inside test_class task
Passed value : 100

6]. **POLYMORPHISM :-** The literal meaning of polymorphism is condition & accurance in different forms.
Polymorphism means a use of single type entity (method, Operator, or object) to represent different types in different scenarios.

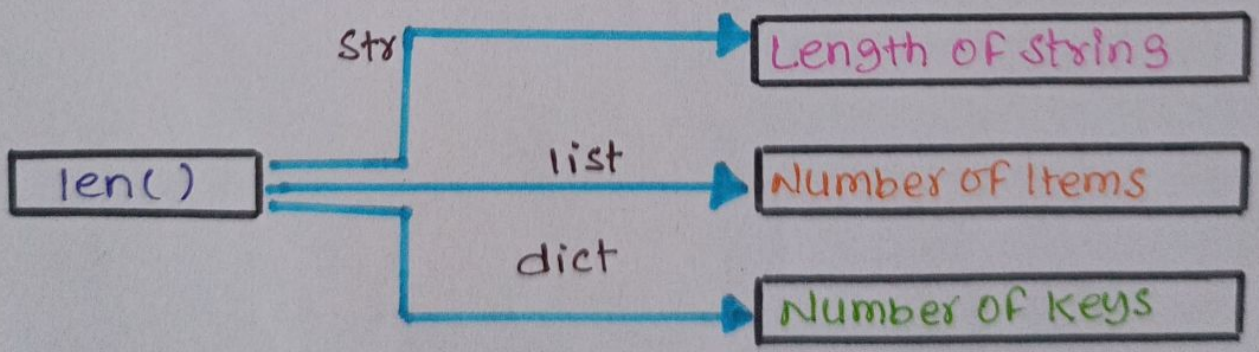
Example:-

```
class Rabbit( ):
def age(self ):
Print("determines age of rabbit ")
def colour(self ):
Print ("determines colour of rabbit")
class Horse ( ):
defage (self ):
Print("determines age of horse")
def colour (self ):
Print("determines colour of horse")
obj 1 = Rabbit ( )
obj2 = Horse ( )
For type in (obj 1 , obj 2 ):
type.age ( )
type.colour ( )
```

Output:-

determines age of rabbit.
determines colour of rabbit.

determines age of horse.
determines colour of horse.



Python OOPs Concepts

Object Oriented Programming.

Python is a multiparadigm programming language. It supports different programming approches.
One of the most popular approches to solve programming problem by creating objects. This is known as Object Oriented programming. (OOP).

- OOP has two characteristics
  1). Attributes
  2). Behavior

Example :- A parrot is an object, as it has following properties.

- name, age, color as attributes.
- Singing, dancing as behavior.

The concepts of OOPs in Python focuses on creating reusable code. This concept is also known as DPY (Don't Repeat Yourself).

Class.

A class is a blueprint for the object. We can think of class as a skech of parrot with labels. It contains all details about the name, colors, size etc.

ex.

```
Class Parrot:
    Pass.
```

Here class keyword define on empty class parrot from class we construct instances (-)n instance is a specific object created from a particular class.

Object :-

An object (instance) is an instantiation of a class. When class is defined only the discription for the class object is defined. Therefore, no memory or storage is allocated.

ex:-

```
Obj = Parrot ( )
```

Here object is an object of class parrot.
Suppose we have details of parrots. Now we are going to show how to build the class and object of parrots.
we can access the class attribute using _ class _ species.

Inheritance :-

Inheritance is a way of creating new class for using details of an existing class without modifying it. The newly formed class is derived class, simillarly, the existing class is a base class.

ex:- Use of Inheritance in python.

```
Class Bird:
    def __init__ (self):
        Print ("Bird is ready")
    def swim (self):
        Print ("swim faster")
Class Penguin (Bird):
    def __init (self):
        super ( ).__init__ ( )
    def run (self):
        Print ("Run fast")
P= Penguin ( )
P. swim ( )
P. run ( )
```

Output

Swim faster
run fast.

We can use the super() function inside the __init__() method. This allows us to run __init__() method.

Encapsulation :-

Using OOP in python, we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation. In python we donate private attribute using _ as the prefix. i.e single _ or double __

```
Class computer:
    def __init__(self):
        self.__maxprice = 900
    def setmaxprice(self, price):
        self._maxprice = price.
C = computer()
C.__maxprice = 1000
C.setMaxPrice(100)
```

We used __init__() method to store the maximum selling price of computer.

C._maxprice = 1000

Method	Variable.

Class Member access specifier	Access from own Class	Accessible from derived	Accessible from Object.
Private	Yes	No	No
Protected	yes	Yes	No
Public	Yes	Yes	No

Polymorphism :-

Polymorphism is an ability to use a common interface for multiple forms (data types) Polymorphism in python defines methods in the child class that have the same name as the method in the parent class. It is possible to modify a method in a child class that it has inherited from child class .. Parent class.

```
Class parrot :
    def fly(self):
        Print("parrot confly")
    def swim(self)
        Print("parrot can't swim")

Class penguin:
    def fly(self):
        Print("Penguin can't fly")
    def swim(self):
        Print("Penguin can swim")

def flying_test(bird):
    bird.fly()

blue = Parrot()
Peg = Penguin()

flying_test(blu).
flying_test(Peg).
```

Output

Parron can fly.
Penguin can't fly.

PYTHON FUNCTIONS

What are Functions?

A Function is a block of code only runs when is called.
In Python a function is defined using the def keyword.

```
def my_function():
    print("Hello")
```

Arguments in a Function

Arguments are specified after the function name, inside the parentheses.
You can add as many arguments as you want separate them with a comma.

```
def my_function(fname)
    Print(fname + "Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```


Arbitary Arguments

If you do not know how many arguments that will be passed into your function, add a before the parameter name in the function definition.

```
def my_function(*kids):
    print("The youngest child is" + kids[2])
my_function("soul", "god1", "txpark")
```

Keyword Arguments (Kwargs)

You can also send arguments with the key = value syntax.

```
def my_function(child3, child2, child1):
  Print("The Youngest child is" + child3)
my_function(child1 = "mcb", child2 = "bca", child3 = "btc")
```

ATUL KUMAR (LINKEDIN).

Arbitary Keyword Arguments ** kwargs.

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

```
def my_function(**kid):
  Print("His last name is" + kid["lname"])
my_function(fname = "Tobias", lname = "Refsnes")
```