# Object-Oriented Programming (OOP)'s concepts in Python

**Object-oriented programming (OOP)** is a method of structuring a program by bundling related properties and behaviors into individual objects.

> Defining a Class in Python

## Class Definition

`class ClassName:

```
    # Statement`
```

## Object Definition

```
 obj = ClassName()
print(obj.atrr)
```

> Syntax

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

**example**

`class Dog:

```
    # Class attribute
    species = "Canis familiaris"

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

In [3]:
```
# Python3 program to
# demonstrate instantiating
# a class


class Dog:

    # A simple class
    # attribute
    attr1 = "mammal"
    attr2 = "dog"

    # A sample method
    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)


# Driver code
# Object instantiation
Rodger = Dog()

# Accessing class attributes
# and method through objects
print(Rodger.attr1)
Rodger.fun()
```

```
mammal
I'm a mammal
I'm a dog
```

In [5]:
```
# Sample class with init method
class Person:

    # init method or constructor
    def __init__(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)


p = Person('Shiva')
p.say_hi()
```

```
Hello, my name is Shiva
```

## Class and Instance Variables

> Instance variables are for data, unique to each instance and class variables are for attributes and methods shared by all instances of the class. Instance variables are variables whose value is assigned inside a constructor or method with self whereas class variables are variables whose value is assigned in the class.

**Defining instance variables using a constructor.**

In [6]:
```
# Python3 program to show that the variables with a value
# assigned in the class declaration, are class variables and
# variables inside methods and constructors are instance
# variables.

# Class for Dog

class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed, color):

        # Instance Variable
        self.breed = breed
        self.color = color


# Objects of Dog class
Rodger = Dog("Pug", "brown")
Buzo = Dog("Bulldog", "black")

print('Rodger details:')
print('Rodger is a', Rodger.animal)
print('Breed: ', Rodger.breed)
print('Color: ', Rodger.color)

print('\nBuzo details:')
print('Buzo is a', Buzo.animal)
print('Breed: ', Buzo.breed)
print('Color: ', Buzo.color)

# Class variables can be accessed using class
# name also
print("\nAccessing class variable using class name")
print(Dog.animal)
```

```
Rodger details:
Rodger is a dog
Breed:  Pug
Color:  brown

Buzo details:
Buzo is a dog
Breed:  Bulldog
Color:  black

Accessing class variable using class name
dog
```

**Defining instance variables using the normal method.**

In [7]:
```
# Python3 program to show that we can create
# instance variables inside methods

# Class for Dog


class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed):

        # Instance Variable
        self.breed = breed

    # Adds an instance variable
    def setColor(self, color):
        self.color = color

    # Retrieves instance variable
    def getColor(self):
        return self.color


# Driver Code
Rodger = Dog("pug")
Rodger.setColor("brown")
print(Rodger.getColor())
```

```
brown
```

**Constructors** are generally used for instantiating an object.

- The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. In Python the **init**() method is called the constructor and is always called when an object is created.

> Syntax of constructor declaration :

` def **init**(self):

    # body of the constructor `

In [8]:
```
#default constructor :
class dummy:

    # default constructor
    def __init__(self):
        self.geek = "dummy"

    # a method for printing data members
    def print_Geek(self):
        print(self.geek)


# creating object of the class
obj = dummy()

# calling the instance method using the object obj
obj.print_Geek()
```

```
dummy
```

In [9]:
```
# parameterized constructor :

class Addition:
    first = 0
    second = 0
    answer = 0

    # parameterized constructor
    def __init__(self, f, s):
        self.first = f
        self.second = s

    def display(self):
        print("First number = " + str(self.first))
        print("Second number = " + str(self.second))
        print("Addition of two numbers = " + str(self.answer))

    def calculate(self):
        self.answer = self.first + self.second

# creating object of the class
# this will invoke parameterized constructor
obj = Addition(1000, 2000)

# perform Addition
obj.calculate()

# display result
obj.display()
```

```
First number = 1000
Second number = 2000
Addition of two numbers = 3000
```

# Inheritance in py

**inheritance is the capability of one class to derive or inherit the properties from another class.**

> Benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides the reusability of a code. We don’t have to write the same code again and again. Also, it allows 3. us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

## Inheritance Syntax

```
Class BaseClass:
    {Body}
Class DerivedClass(BaseClass):
    {Body}
```

In [10]:
```
# A Python program to demonstrate inheritance

class Person(object):

  # Constructor
  def __init__(self, name, id):
    self.name = name
    self.id = id

  # To check if this person is an employee
  def Display(self):
    print(self.name, self.id)


# Driver code
emp = Person("Satyam", 102) # An Object of Person
emp.Display()
```

```
Satyam 102
```

In [12]:
```
class Emp(Person):

  def Print(self):
    print("Emp class called")

Emp_details = Emp("shiva", 103)

# calling parent class function
Emp_details.Display()

# Calling child class function
Emp_details.Print()
```

```
shiva 103
Emp class called
```

In [14]:
```
# A Python program to demonstrate inheritance

# Base or Super class. Note object in bracket.
# (Generally, object is made ancestor of all classes)
# In Python 3.x "class Person" is
# equivalent to "class Person(object)"


class Person(object):

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

    # To get name
    def getName(self):
        return self.name

    # To check if this person is an employee
    def isEmployee(self):
        return False


# Inherited or Subclass (Note Person in bracket)
class Employee(Person):

    # Here we return true
    def isEmployee(self):
        return True


# Driver code
emp = Person("heyyy")  # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("heyyy11")  # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

```
heyyy False
heyyy11 True
```

In [16]:
```
# Python code to demonstrate how parent constructors
# are called.

# parent class
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

# child class


class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)


# creation of an object variable or an instance
a = Employee('shiva', 886012, 200000, "Intern")

# calling a function of the class Person using its instance
a.display()
```

```
shiva
886012
```

In [18]:
```
class A:
    def __init__(self, n='shiva'):
        self.name = n


class B(A):
    def __init__(self, roll):
        self.roll = roll


object = B(23)
print(object.name)
```

```
---------------------------------------------------------------------
---------
AttributeError                            Traceback (most recent c
all last)
/var/folders/nj/5n17tpm16j1dszf8dzt_qfj00000gn/T/ipykernel_1388/27
42791510.py in <module>
     10
     11 object = B(23)
---> 12 print(object.name)

AttributeError: 'B' object has no attribute 'name'
```

In [ ]: