Class

1. A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods. 2. To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes. class is a keyword is used to create a class

init Method

In [ ]:

```
__ init__() - This method is used to initilize the variables. This is a special method.
we do not call this method explicitly
Self is a variable which refers to current class instance/object
Object is a class type variable or class instance. To use a class, we should create an o
bject to the class
```

Some points on Python class:

1. Classes are created by keyword class. 2. Attributes are the variables that belong to a class. 3. Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

Example: Creating an empty Class in Python

In [1]:

```
class Dog:
    pass
```

In the above example, we have created a class named dog using the class keyword.

Objects

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string "Hello, world" is an object, a list is an object that can hold other objects, and so on. You've been using objects all along and may not even realize it.An object consists of : State: It is represented by the attributes of an object. It also reflects the properties of an object. Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects. Identity: It gives a unique name to an object and enables one object to interact with other objects. To understand the state, behavior, and identity let us take the example of the class dog (explained above). The identity can be considered as the name of the dog. State or Attributes can be considered as the breed, age, or color of the dog. The behavior can be considered as to whether the dog is eating or sleeping.

In [31]:

```
class Myclass:
    def show(self):
        print("I am a Method")

x = Myclass()
x.show()
```

```
I am a Method
```

In [32]:

```
class Mobile:
    def __init__(self):
        self.model = 'RealMe X'
    def show_model(self):
        print("Model:", self.model)

realme = Mobile()
realme.show_model()
```

```
Model: RealMe X
```

In [33]:

```
class Mobile:
    def __init__(self):
        self.model = 'Realme X'

    def show_model(self):
        print("Model:",self.model)

realme = Mobile()
realme.model = 'RealMe Pro2'
realme.show_model()
```

```
Model: RealMe Pro2
```

In [34]:

```
class Mobile:
    def __init__(self):
        self.model = 'Realme X'

    def show_model(self):
        print("Model:",self.model)

realme = Mobile()
realme.model = 'RealMe Pro2'
realme.show_model()
```

```
Model: RealMe Pro2
```

In [35]:

```
class Mobile:
    def __init__(self):
        self.model = 'RealMe X'
    def show_model(self):
        price = 1000
        print("Model:", self.model, "price:",price)

realme = Mobile()
realme.show_model()
```

```
Model: RealMe X price: 1000
```

In [36]:

```
class Mobile():
    def __init__(self,m):
        self.model = m

    def show_model(self,p):
        self.price = p
        print("Model:", self.model, "Price:", self.price)

realme = Mobile('Narzo 20A')
realme.show_model(10000)
print(id(realme))
print()

samsung = Mobile('samsung 2s')
samsung.show_model(20000)
```

```
print(id(samsung))
print()

geek = Mobile('Python')
geek.show_model(30000)
print(id(geek))
```

```
Model: Narzo 20A Price: 10000
2247135036752

Model: samsung 2s Price: 20000
2247135037280

Model: Python Price: 30000
2247134871904
```

In [37]:

```
class MyClass:
    x = 5

p1 = MyClass()
print(p1.x)
```

```
5
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

In [39]:

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

p1 = Person("pravin",25)

print(p1.name)
print(p1.age)
```

```
pravin
25
```

In [41]:

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

p1 = Person("John", 36)

print(p1)
```

```
<__main__.Person object at 0x0000020B323A41F0>
```

In [42]:

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

    def __str__(self):
        return f"{self.name}({self.age})"

p1 = Person("Yash", 36)
print(p1)
```

```
Yash(36)
```

In [43]:

```
# Insert a function that prints a greeting, and execute it on the p1 object:

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

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

p1 = Person("aww",36)
p1.myfunc()
```

```
Hello my name is : aww
```

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:Use the words mysillyobject and abc instead of self:

In [46]:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("Dipak", 25)
p1.myfunc()
```

```
Hello my name is Dipak
```

In [48]:

```
p1.age
```

Out[48]:

```
25
```

In [51]:

```
p1.age = 26
```

In [52]:

```
p1.age
```

Out[52]:

```
26
```

In [53]:

```
# Delete Object Properties

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

  def myfunc(self):
    print("Hello my name is " + self.name)
```

```
p1 = Person("John", 36)

del p1.age

print(p1.age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [53], in <cell line: 15>()
     11 p1 = Person("John", 36)
     13 del p1.age
---> 15 print(p1.age)

AttributeError: 'Person' object has no attribute 'age'
```

In [54]:

```
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
```

__init__ method The __init__ method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

In [55]:

```
# 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('Nikhil')
p.say_hi()
```

```
Hello, my name is Nikhil
```

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 [56]:

```
# 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 [57]:

```
# 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
```

In [62]:

```
class Dog:
    species = "Canis familiaris"
    def __init__(self, name, age):
        self.name = name
        self.age = age

buddy =   Dog('egle',12)
```

In [63]:

```
buddy.age
```

Out[63]:

```
12
```

In [64]:

```
class SchoolForm:
    def printDetails(self):
            print("Your name is : ", self.name)
            print("Your roll no. is : ", self.roll no)

rahul = SchoolForm()
rahul.name = 'rahul'
rahul.rollno = '101'
rahul.printDetails()
```

```
  Input In [64]
    print("Your roll no. is : ", self.roll no)
                                               ^
SyntaxError: invalid syntax
```

In [2]:

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

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")
```

In [3]:

```
person = Person("Dipak", 25)
```

In [4]:

```
person.age
```

Out[4]:

```
25
```

In [5]:

```
person.greet()
```

```
Hello, my name is Dipak and I am 25 years old.
```

Car Class

In [6]:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start(self):
        print(f"{self.make} {self.model} ({self.year}) is starting.")

    def stop(self):
        print(f"{self.make} {self.model} ({self.year}) is stopping.")
```

In [7]:

```
car = Car("Toyota", "Camry", 2021)
car.start()
car.stop()
```

```
Toyota Camry (2021) is starting.
Toyota Camry (2021) is stopping.
```

Rectangle class

In [8]:

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)
```

In [9]:

```
rect = Rectangle(5, 10)
print(rect.area())
print(rect.perimeter())
```

```
50
30
```

Creating an object

In [10]:

```
obj = Dog()
```

Example 1: Creating a class and object with class and instance attributes

In [11]:

```
class Dog:

     # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))

# Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

```
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
```

Example 2: Creating Class and objects with methods

In [12]:

```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My name is {}".format(self.name))

        # Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()
```

```
My name is Rodger
My name is Tommy
```

In [13]:

```
def reverse(s):
    str = ""
    for i in s:
        str = i + str
    return str

s = "Geeksforgeeks"

print("the original string is : ", end = "")
print(s)

print("The reversed string(using loops) is : ", end="")
```

```
print(reverse(s))
```

```
the original string is : Geeksforgeeks
The reversed string(using loops) is : skeegrofskeeG
```

In [14]:

```
def reverse(s):
    str = ""
    for i in s:
        str = i + str
    return str

s = str(input("Enter any word:"))

print("the original string is : ", end = "")
print(s)

print("The reversed string(using loops) is : ", end="")
print(reverse(s))
```

```
Enter any word:abcd
the original string is : abcd
The reversed string(using loops) is : dcba
```

Program description:- Python program to print numbers from 1 to 10 using while loop

In [17]:

```
#|print('Numbers from 1 to 10:')
n = 1
while n <= 10:
    print(n, end = '')
    n = n + 1
print(n)
```

```
1234567891011
```

In [18]:

```
class car:
    pass
```

In [19]:

```
class car:
    def __init__(self, brand_name, fueltype, body_type):

        self.brand_name = brand_name
        self.fueltype = fueltype
        self.body_type = body_type

    def desc_car(self):
        print(self.brand_name, self.fueltype, self.body_type)
```

In [20]:

```
innova = car("toyota" , "petrol" , "suv")
nexon = car("tata", "petrol", "min suv")
```

In [21]:

```
innova.brand_name
```

Out[21]:

```
'toyota'
```

In [22]:

```
nexon.brand_name
```

Out[22]:

```
'tata'
```

In [23]:

```
innova.desc_car()
```

```
toyota petrol suv
```

In [24]:

```
a = 10
```

In [ ]:

```

```