TITLE- CLASSES AND OBJECTS IN PYTHON

What is Python?

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python supports multiple programming paradigms, including Procedural, Object Oriented and Functional programming language. Python design philosophy emphasizes code readability with the use of significant indentation.

This tutorial gives a complete understanding of Python programming language starting from basic concepts to advanced concepts. This tutorial will take you through simple and practical approaches while learning Python Programming language.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Data analysis and machine learning

Python has become a staple in data science, allowing data analysts and other professionals to use the language to conduct complex statistical calculations, create data visualizations, build machine learning algorithms, manipulate and analyze data, and complete other data-related tasks.

Python can build a wide range of different data visualizations, like line and bar graphs, pie charts, histograms, and 3D plots. Python also has a number of libraries that enable coders to write programs for data analysis and machine learning more quickly and efficiently, like TensorFlow and Keras.

Python, one of the most popular programming languages in the world, has created everything from Netflix’s recommendation algorithm to the software that controls self-driving cars.

CLASSES

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

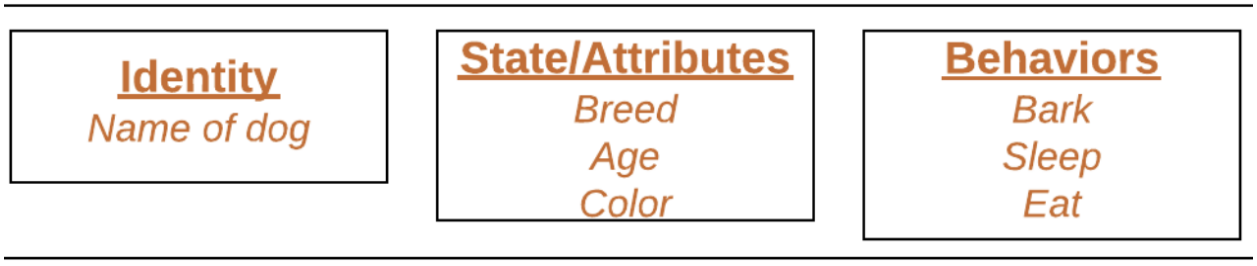
- A class is a user-defined blueprint or prototype from which objects are created.
- Classes provide a means of bundling data and functionality together.
- Creating a new class creates a new type of object, allowing new instances of that type to be made.

- Each class instance can have attributes attached to it for maintaining its state.
- Class instances can also have methods (defined by their class) for modifying their state.
- Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.
- Some points on Python class:

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

- Defining a class:

```
# Python3 program to
# demonstrate defining
# a class
class Dog:
    pass
```



OBJECTS

- An Object is an instance of a Class.
- A class is like a blueprint while an instance is a copy of the class with actual values.
- An object consists of:
- State: It is represented by the attributes of an object. It also reflects the properties of an object.
- Behaviour: 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.
- Declaring Objects (Also called instantiating a class):
- This will create a new object instance named harry. We can access the attributes of objects using the object name prefix.
- Attributes may be data or method. Methods of an object are corresponding functions of that class.

```
# Python3 program to
# demonstrate instantiating
# a class


class Dog:

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

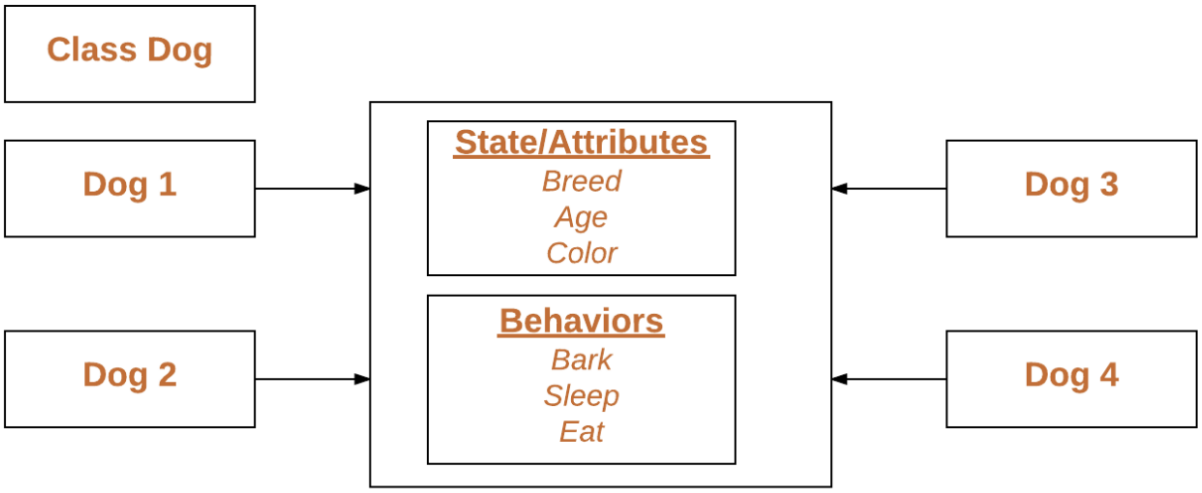
```
    # A sample method
    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)


# Driver code
# Object instantiation
Rodger = Dog()

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

OUTPUT

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



METHODS

- Objects can also contain methods.
- Methods in objects are functions that belong to the object.
- These methods are defined inside a class.
- These methods are the reusable piece of code that can be invoked/called at any point in the program.
- Python offers various types of these methods. These are crucial to becoming an efficient programmer and consequently are useful for a data science professional. Defining a class:

TYPES OF METHODS

There are basically three types of methods in Python:

1. Instance Method
2. Class Method
3. Static Method

Instance Methods

The purpose of instance methods is to set or get details about instances (objects), and that is why they are known as instance methods. They are the most common type of methods used in a Python class.

They have one default parameter- self, which points to an instance of the class. Although you do not have to pass that every time. You can change the name of this parameter but it is better to stick to the convention i.e., self.

Any method you create inside a class is an instance method unless you specially specify Python otherwise. Let us see how to create an instance method:

```
class My_class:
  def instance_method(self):
    return "This is an instance method."
```

Class Methods

The purpose of the class methods is to set or get the details (status) of the class. That is why they are known as class methods. They can't access or modify specific instance data. They are bound to the class instead of their objects. Two important things about class methods:

- In order to define a class method, you have to specify that it is a class method with the help of the @classmethod decorator
- Class methods also take one default parameter- cls, which points to the class. Again, this not mandatory to name the default parameter "cls". But it is always better to go with the conventions

Now let's look at how to create class methods:

```
class My_class:

  @classmethod
  def class_method(cls):
    return "This is a class method."
```

Static Methods

Static methods cannot access the class data. In other words, they do not need to access the class data. They are self-sufficient and can work on their own. Since they are not attached to any class attribute, they cannot get or set the instance state or class state.

In order to define a static method, we can use the @staticmethod decorator (in a similar way we used the @classmethod decorator). Unlike instance methods and class methods, we do not need to pass any special or default parameters. Let us look at the implementation:

```
class My_class:

  @staticmethod
  def static_method():
    return "This is a static method."
```

The self

- Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it.
- If we have a method that takes no arguments, we still have one argument.
- This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) – this is all the special self is about.

__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.

```
# 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
```

Problem Statement:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x += dx
        self.y += dy

    def distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
```

```
        return (dx**2 + dy**2) ** 0.5

p1 = Point(0, 0)
p2 = Point(3, 4)

print(p1.distance(p2))  # Output: 5.0
```

In this example, we define a Point class that represents a point in two-dimensional space. The Point class has an __init__() method that initializes the x and y attributes of the point, as well as a move() method that moves the point by a certain amount in the x and y directions, and a distance() method that calculates the distance between two points.

We create two Point objects, p1 and p2, and use the distance() method to calculate the distance between them.

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

    def get_grade(self):
        return self.grade

    def is_passing(self):
```

```
        return self.grade >= 60

class Course:
    def __init__(self, name, max_students):
        self.name = name
        self.max_students = max_students
        self.students = []

    def add_student(self, student):
        if len(self.students) < self.max_students:
            self.students.append(student)
            return True
        return False

    def get_average_grade(self):
        total_grades = 0
        for student in self.students:
            total_grades += student.get_grade()
        return total_grades / len(self.students)

math_course = Course("Math", 25)

john = Student("John Doe", 85, 15)
jane = Student("Jane Smith", 75, 16)
bob = Student("Bob Johnson", 50, 17)

math_course.add_student(john)
math_course.add_student(jane)
math_course.add_student(bob)

print(math_course.get_average_grade())  # Output: 70.0
```

RESULT-

1. Class variables are typically used to define attributes that are common to all objects of the same class. A class variable is a variable that is shared by all instances of a class. It is a variable that is defined at the class level, rather than at the instance level. In the program product is the variable defined outside of any method so it is a class variable.

2. This creates an object that has the given properties initialized to the given values. We can then access these properties and get their values. The __init__ method is a crucial part of a class, because it allows you to create objects with the desired initial state. Without a constructor, you would have to create an object and then set its properties manually, which can be tedious and error-prone. The __init__ method makes it easy to create objects with the right initial state, and it ensures that all objects of the class are properly initialized.

3. In Python, a constructor is a method named __init__ that is defined in a class. The __init__ method is called automatically whenever a new object is created from the class. It is typically used to set up or initialize the object's properties.

4. Class variables can be accessed using the class name in Python. Class variables are variables that are defined at the class level, rather than at the instance level.

5. The print statement is used to output text to the console in Python. It is a built-in function that takes one or more values as arguments and prints them to the standard output (usually the console).

CONCLUSION-

Object-Oriented Programming – is a paradigm that relies on objects and classes to create functional programs. OOPs work on the modularity of code, and classes and objects help in writing reusable, simple pieces of code that can be used to create larger software features and modules.

A class in Python is a user-defined prototype using which objects are created. Put simply, a class is a method for bundling data and functionality together. The two keywords are important to note. Data means any variables instantiated or defined, whereas functionality means any operation that can be performed on that data. Together with data and functionality bundled under one package, we get classes.

Classes help you create a user-defined data structure that has its own data members (variables) and member functions. You can access these variables and methods simply by creating an object for the class (we'll talk more about it later). So, in a sense, classes are just like a blueprint for an object.

Further, creating classes automatically creates a new type of objects – which allows you to further create more objects of that same type. Each class instance can have attributes attached to it in order to maintain its state. Class instances can themselves have methods (as defined by their class) for modifying the state.