Python Classes/Objects

Python is an object oriented programming language.
Almost everything in Python is an object, with its properties and methods.
A Class is like an object constructor, or a "blueprint" for creating objects.

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.

➢ **Some points on Python class:**

Classes are created by keyword class.

Attributes are the variables that belong to a class.

Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

Class Definition Syntax:

```
class ClassName:
    # Statement-1
    .
    .
    .
    # Statement-N
```

Create a Class

To create a class, use the keyword `class`:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

Class 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*. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.

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.

Declaring Objects (Also called instantiating a class)
When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

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, then we still have to have one argument.

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

Constructors are used to initializing 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.

```
A 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()
```

FILE HANDLING IN PYTHON

Python File Open

File handling is an important part of any web application.
Python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the open() function.
The open() function takes two parameters; *filename*, and *mode*.
There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist
"a" - Append - Opens a file for appending, creates the file if it does not exist
"w" - Write - Opens a file for writing, creates the file if it does not exist
"x" - Create - Creates the specified file, returns an error if the file exists
In addition you can specify if the file should be handled as binary or text mode
"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because `"r"` for read, and `"t"` for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Open a File on the Server

Assume we have the following file, located in the same folder as Python:
demofile.txt

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

To open the file, use the built-in open() function.
The open() function returns a file object, which has a read() method for reading the content of the file

Example

```
f = open("demofile.txt", "r")
print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this:

Example

Open a file on a different location:

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Read Lines

You can return one line by using the `readline()` method:

Example

Read one line of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

Example

Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

It is a good practice to always close the file when you are done with it.

Example

Close the file when you are finish with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Write to an Existing File

To write to an existing file, you must add a parameter to the `open()` function:
`"a"` - Append - will append to the end of the file
`"w"` - Write - will overwrite any existing content

Example

Open the file "demofile2.txt" and append content to the file

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the `open()` method, with one of the following parameters:

`"x"` - Create - will create a file, returns an error if the file exist
`"a"` - Append - will create a file if the specified file does not exist
`"w"` - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function:

Example

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

Example

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```