OBJECT ORIENTED PROGRAMMING

Language Classes	Categories	Languages
Programming Paradigm	Procedural	C, C++, C#, Objective-C, Java, Go
	Scripting	CoffeeScript, JavaScript, Python, Perl, Php, Ruby
	Functional	Clojure, Eralang, Haskell, Scala
Compilation Class	Static	C, C++, C#, Objective-C, Java, Go, Haskell, Scala
	Dynamic	CoffeeScript, JavaScript, Python, Perl, Php, Ruby, Clojure, Eralang
Type Class	Strong	C#, Java, Go, Python, Ruby, Clojure, Erlang, Haskell, Scala
	Weak	C, C++, Objective-C, CoffeeScript, JavaScript, Perl, Php
Memory Class	Managed	Others
	Unmanaged	C, C++, Objective-C

What is Object Oriented Programming

- Object Oriented Programming (OOP) is based on the concept of objects rather than actions, and data rather than logic. In order for a programming language to be object oriented, it should have a mechanism to enable working with classes and objects as well as the implementation and usage of the fundamental object-oriented principles and concepts namely inheritance, abstraction, encapsulation and polymorphism

WHY USE OBJECT ORIENTED PROGRAMMING

- Classes help us logically group our data and functions in a way that's easy to reuse and easy to build upon if needed.
- Provides a clear program structure, which makes it easy to map real world problems and their solutions.
- Facilitates easy maintenance and modification of existing code.
- Enhances program modularity because each object exists independently and new features can be added easily without disturbing the existing ones.
- Presents a good framework for code libraries where supplied components can be easily adapted and modified by the programmer.
- Imparts code reusability

Procedural vs. Object Oriented Programming

- Procedural based programming is derived from structural programming based on the concepts of functions/procedure/routines. It is easy to access and change the data in procedural oriented programming. On the other hand, Object Oriented Programming (OOP) allows decomposition of a problem into a number of units called objects and then build the data and functions around these objects. It emphasis more on the data than procedure or functions. Also in OOP, data is hidden and cannot be accessed by external procedure

Encapsulation

- This property hides unnecessary details and makes it easier to manage the program structure. Each object's implementation and state are hidden behind well-defined boundaries and that provides a clean and simple interface for working with them. One way to accomplish this is by making the data private.

Inheritance

- Inheritance, also called generalization, allows us to capture a hierarchal relationship between classes and objects. For instance, a ‘fruit’ is a generalization of ‘orange’. Inheritance is very useful from a code reuse perspective.

Abstraction

- This property allows us to hide the details and expose only the essential features of a concept or object. For example, a person driving a scooter knows that on pressing a horn, sound is emitted, but he has no idea about how the sound is actually generated on pressing the horn.

POLYMORPHISM

- Poly-morphism means many forms. That is, a thing or action is present in different forms or ways. One good example of polymorphism is constructor overloading in classes.

Modules vs. Classes and Objects

- Modules are like "Dictionaries" When working on Modules, note the following points:
- A Python module is a package to encapsulate reusable code.
- Modules reside in a folder with a __init__.py file on it.
- Modules contain functions and classes.
- Modules are imported using the import keyword.

Modules continued

- Consider a module named employee.py with a function in it called employee. The code of the function is given below: # this goes in employee.py

```
def EmployeID():
    print ("Employee Unique Identity!")
```

Now import the module and then access the function EmployeID:

```
import employee
employee. EmployeID()
```

Modules continued

- Notice that there is common pattern in Python:
- Take a key = value style container
- Get something out of it by the key's name When comparing module with a dictionary, both are similar, except with the following:
- In the case of the dictionary, the key is a string and the syntax is [key].
- In the case of the module, the key is an identifier, and the syntax is .key.

Classes

- Classes are like Modules Module is a specialized dictionary that can store Python code so you can get to it with the '.' Operator. A class is a way to take a grouping of functions and data and place them inside a container so you can access them with the '.'operator.
- Note: Classes are preferred over modules because you can reuse them as they are and without much interference. While with modules, you have only one with the entire program.

Class Example

```
class employee(object):
    def __init__(self):
        self. Age = "Employee Age is ##"
    def EmployeeID(self):
        print ("This is just employee unique identity"
```

Object

- A class is like a mini-module and you can import in a similar way as you do for classes, using the concept called instantiate. Note that when you instantiate a class, you get an object.
- You can instantiate an object, similar to calling a class like a function, as shown:

Obj = employee()

Example for use case

- If we have a company , each employee will have different attributes and methods. (name , email, salary, actions).
- Instead of saving the employee data over and over again we create a blueprint to use for all employees at once.

IMPORTANT NOTES

- A FUNCTION IN A CLASS IS ALSO REFERRED TO AS A METHOD.
- THERE ARE TWO KINDS OF VARIABLES , INSTANCE VARIABLES AND CLASS VARIABLES.
- INSTANCE VARIABLES ARE FOR THE INSTANCE WE CREATE FROM THE CLASS.
- WHEN WE CREATE A CLASS WE NEED TO CREATE AN INITIALIZER FUNCTION TO INITIALIZE THE VARIABLES OF THE ATTRIBUTES. (CONSTRUCTOR)

Important Notes Continued

- When you create a function inside the class the first argument is always the instance itself.
- By convention it is called self.
- One of the most common mistakes when writing methods or functions is forgetting the self argument.

Class variables

- Variables that are shared for all instances in the class.
- When we access class variable we need to access them through the class itself or an instance of a class.
- When we change the class variable using an instance it only changes it for this specific instance because it creates the attribute for this instance. (check it using __dict__)

Regular vs Class vs Static Methods

- Regular automatically takes the instance as the first argument.
- Class method takes the class as the first argument. Common convention is cls.
- Class methods changes the class variables.
- You can run a class method from an instance but it doesn't make sense and nobody does it.

Continued

- Class methods can be used as an alternative constructor.
- Static methods don't take and instance or a class as an argument.
- It is connected logically to the class but doesn't need anything. (instance or class variable)

INHERITANCE

- As the name indicates , it allows to inherit methods and attributes from a parent class.
- Useful to create subclasses and make use of the classes we already have and alter what we need without affecting the original class.
- When creating an instance from the subclass it looks for the method inside the subclass, if not found it will look in the parent class.

NOTES

- Special methods usually surrounded by underscores. And they are called dunder methods
- Repr and str are commonly used special functions
- Repr is used to fix printing out vague output when printing the instance. It's used for debugging and logging and read by other developers
- Str is more readable , displayed for the end user.