Object Oriented Programming (OOP) Concepts

- Python is an object-oriented language that allows users to develop applications using an Object-Oriented approach i.e. to use classes and objects to design a program.
- OOPs is a way of computer programming using the idea of “objects” to represents data (variables) and methods (functions)
- Object-oriented programming enables to develop large, modular programs that can instantly expand over time.
- Object-oriented programs hide the implementation from the end-user.
- It enables you to develop any large-scale software and GUIs effectively.

Overview of OOP Terminologies

S. No.	Terms	Description
1.	Class	A user-defined prototype that binds variables and functions.
2.	Object/ Instance	A unique identity of a class (user-defined data structure).
3.	Data member / Attributes	**Variables** that hold data associated with a class and its objects
4.	Member Function/ Methods	**Functions** that are defined in a class definition.
5.	Instantiation	Creation / Initialization of a class
6.	Class Variable	A variable that is defined within a class but outside any of the class's methods.
7.	Instance Variable	A variable that is defined inside a method of a class.

Class

- A class is a blueprint/prototype/template that binds data members (variables) and member functions (methods) together as a single unit so that no other part of the code can access this data.
- These variables and functions are accessed by creating the object of the class.

Creating a Class

- All class definitions start with the **class** keyword, which is followed by the name of the class and a colon.
- Any code that is indented below the class definition is considered part of the class’s body.

Class Definition Syntax:

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

Creating an empty Class in Python

- The **pass** keyword. pass is often used as a placeholder when no definition of class is specified.
- It allows you to run this code without Python throwing an error.

```
class class_name:
    pass
```

Built-in Classes

There are several built-in classes (data types) in Python: integer, float, string, Boolean, list, range, tuple, set, dictionary, and some other, rarely used ones:

Example:

```
a = 5.2
b = 'Hello World'
c = [1, 2, 3]
d = False
e = range(4)
f = (1, 2, 3)
g = complex(1, -1)

for var in [a, b, c, d, e, f, g]:
   print(type(var))
```

Output:

```
<class 'float'>
<class 'str'>
<class 'list'>
<class 'bool'>
<class 'range'>
<class 'tuple'>
<class 'complex'>
```

Objects

- The object is an entity that is used to access the members of a class.
- It has the following properties:
  - ✓ **identity** – the unique name
  - ✓ **state** – to access variables/attributes/properties of the class
  - ✓ **behaviour** – to access methods/functions/actions of the class
- A class only the description, no memory allocation is done until we create its **object**.
- A class can have many objects.
- Creating a new object from a class is called **instantiating** an object.

(General example, a car can be an object. If we consider the car as an object then its properties/state would be – its color, its model, its price, its brand, etc. And its behavior/function would be acceleration, slowing down, gear change, etc. Therefore, an object of a class is called as a real-world entity.)

Creating Object:

An object of a class is creating by assigning a variable (object_name) a the name of the class, followed by opening and closing parentheses:

Syntax:

```
<object-name> = <class-name>( )

OR

obj1 = class_name()
```

Example:

```
class A:
  x = 5
obj = A()
print(obj.x)
```

Output:

```
5
```

Example:

```
class A:
 x = "Python"
 y = "DS"
obj = A()
print(obj. x)
print(obj.y)
```

Output:

```
Python
DS
```

Class Methods

- Class Methods are the **functions defined inside any class.**
- Class methods **must have an extra first parameter in the method definition (generally called as self)**
- **We do not give a value for this parameter when we call the method**, Python interpreter provides it.
- If we have a method that takes no arguments, then we still have to have one argument.
- The methods defined inside a class other than the constructor method are known as the instance methods.

Example:

```
class A:
 x = "Python"
 y = "DS"
 def func(self):
    print("Welcome to:", self.x)
    print("Happy Learning:", self.y)

obj = A()
print(obj. x)
print(obj.y)
obj.func()
```

Output:

```
Python
DS
Welcome to: Python
Happy Learning: DS
```

Example:

```
class Employee:
    id = 10
    name = "John"
    def display (self):
        print("ID: %d \nName: %s"%(self.id,self.name))
# Creating a emp instance of Employee class
emp = Employee()
emp.display()
```

Output:

```
ID: 10
Name: John
```

Note:

When a method is called using object as object_name.method_name(arg1, arg2), this is automatically converted by into class_name.method_name(object_name, arg1, arg2) – this is all the special self is about.

‘self’ parameter

- **The “self” parameter is the reference of the object of a class.**
- The “self” is **the first parameter in any function defined inside a class.**
- The self-parameter **is used to access variables and methods that belongs to the class.**
- **To access these functions and variables inside the class, their name must be preceded with ‘self’ and a full-stop (e.g. self.variable_name).**

Note:

It does not have to be named self, you can call it whatever you like.

Example:

```
class Employee:              #Creating Class
   id = 10                   #Class variable
   name = "SD"               #Class variable
   def display (self):       #Class method with first argument a self parameter
      print(self.id,self.name)   #Accessing class variables using self
a=Employee()                 #Creating Object a using class_name Employee
a.display()                  #Accessing class method using object of class Employee
```

Output:

```
10
SD
```

Additional Points:

Object-Oriented Programming (OOP)	**Procedural-Oriented Programming (Pop)**
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of *Access modifiers* 'public', private', protected'	Doesn't use *Access modifiers*
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance

Index	**Object-oriented Programming**	**Procedural Programming**
1.	Object-oriented programming is the problem-solving approach and used where computation is done by using objects.	Procedural programming uses a list of instructions to do computation step by step.
2.	It makes the development and maintenance easier.	In procedural programming, It is not easy to maintain the codes when the project becomes lengthy.
3.	It simulates the real-world entity. So real-world problems can be easily solved through oops.	It doesn't simulate the real world. It works on step-by-step instructions divided into small parts called functions.
4.	It provides data hiding. So it is more secure than procedural languages. You cannot access private data from anywhere.	Procedural language doesn't provide any proper way for data binding, so it is less secure.
5.	Example of object-oriented programming languages is C++, Java, .Net, Python, C#, etc.	Example of procedural languages are: C, Fortran, Pascal, VB etc.