1. A Python class is like an outline for creating a new object. An object is anything that you wish to manipulate or change while working through the code. Every time a class object is instantiated, which is when we declare a variable, a new object is initiated from scratch

Whereas in Python, Modules are simply files with the **.py** extension containing Python code that can be imported inside another Python Program. In simple terms, we can consider a module to be the same as a code library or a file that contains a set of functions/Classes that you want to include in your application.

2. For creating a class instance, we call a class by its name and pass the arguments which its **\_\_init\_\_** method accepts.

**Example:** Kunal = employee ('Male', 50000), Here Kunal is an instance of class employee with attributes 'Male' and 50000.

Whereas for creating a class, we use the **Class** keyword. Class keyword is followed by classname and semicolon.

**Example:** Here Employee is a class created with class keyword with arguments gender and salary.

class Employee: def \_\_init\_\_(self, gender, salary):

self.gender = gender

self.salary = salary

3. Class attributes belong to the class itself. These attributes are shared by all the instances of the class. Hence these attributes are usually created in the top of class definition outside all methods.

Example: class Bike: no\_of\_wheels = 2; # this is a class attribute

def \_\_init\_\_(self, colour, price, engine):

self.colour = colour

self.price = price

self.engine = engine

#These all are instance attributes

4. Instances attributes are passed to the class when an object of the class is created. Unlike class attributes, instance attributes are not shared by all objects of the class. Instead each object maintains its own copy of instance attributes at object level. Whereas in case of class attributes all instances of class refer to a single copy. Usually instance attributes are defined within the **\_\_init\_\_** method of class.

Example: class Car: def \_\_init\_\_(self, colour, price, engine): self.colour = colour # All this are instance attributes

self.price = price

self.engine = engine

BMW = Car ('Black', 4500000, 'Petrol')  
i20 = Car('Grey',800000, 'Diesel')

5. **self** represents the instance of the class (it represents the object itself). By using the “self” keyword we can access the attributes and methods of the class within the class in python. It binds the attributes with the given arguments.

6. Python Classes handle operator overloading by using special methods called **Magic methods**. These special methods usually begin and end with **\_\_**.

Some of the Magic methods for basic arithmetic operators are:

* + -> \_\_add\_\_()
* - -> \_\_sub\_\_()
* \* -> \_\_mul\_\_()
* / -> \_\_div\_\_()

7. When we want to have different meaning for the same operator according to the context we use operator overloading.

8. The most popular form of operator overloading in python is by special methods called **Magic methods** which are usually begin and end with double underscore **\_\_<method name>\_\_**.

Example: **class** X:

**def** \_\_init\_\_(self, x):

self**.**x **=** x

**def** \_\_add\_\_(self,y):

**return** self**.**x+y.x

obj1 **=** X(4)

obj2 **=** X(5)

obj3 **=** X('Kunal')

obj4 **=** X(' Ishank')

print(f'Sum: {obj1**+**obj2}')

print(f'String Concatenation: {obj3**+**obj4}')

Output: Sum: 9

String Concatenation: Kunal Ishank

9. **Classes** and **objects** are the two concepts to comprehend python OOP code as more formally objects are entities that represent instances of general abstract concept called class.