1. Basically, a class can be instantiated but a module cannot. A module will never be anything other than a library of methods. A class can be so much more -- it can hold its state (by keeping track of instance variables) and be duplicated as many times as you want. It's all about objects.

2. class Student:

def \_\_init\_\_(self,name):

self.name = name

def \_\_str\_\_(self):

return f"This is {self.name} (an instance of a class Student)"

student1 = Student("Mahak")

print(student1)

3. Class attributes belong to the class itself they will be shared by all the instances. Such attributes are defined in the class body parts usually at the top, for legibility.

class Student:

class\_teacher = 'Johnson' # class attribute

def \_\_init\_\_(self,name, roll\_no): # to bind attributes to class when object is created

self.name = name

self.roll\_no = roll\_no

def \_\_str\_\_(self):

return f"This is {self.name} (an instance of a class Student)"

student1 = Student("Akash Deep",25)

print(student1.class\_teacher)

4. Unlike class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute. They are created in init method. How they are created is Explained below::

class Student:

def \_\_init\_\_(self,name, roll\_no): # to bind attributes to class when object is created

self.name = name

self.roll\_no = roll\_no

def \_\_str\_\_(self):

return f"This is {self.name} (an instance of a class Student)"

student1 = Student("Mahak",25)

print(student1.name)

5. Self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python

6. Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class.

# + and \* operator is used for different purpose

print(1 + 2)

# concatenate two strings

print("abcd"+"efg")

# Product two numbers

print(13 \* 3)

# Repeat the String

print("Ineuron"\*4)

7. Let us assumewe have an object called string1 which is a string object as defined below. Now, when we try to add a string to this string object, the compiler will throw an error. This is because the compiler doesn't know how to add them.

# declare our own string class

class String:

def \_\_init\_\_(self, string):

self.string = string

def \_\_repr\_\_(self):

return 'Object: {}'.format(self.string)

string1 = String('Hello')

# concatenate String object and a string

print(string1 +' world')

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-6-57c4524b4029> in <module>

9

10 # concatenate String object and a string

---> 11 print(string1 +' world')

TypeError: unsupported operand type(s) for +: 'String' and 'str'

This error can be avoided by adding the add method to the String class. This way, we are overloading the + operator to concatenate a string object with a string.

# declare our own string class

class String:

def \_\_init\_\_(self, string):

self.string = string

def \_\_add\_\_(self, other):

return self.string + other

def \_\_repr\_\_(self):

return 'Object: {}'.format(self.string)

string1 = String('Hello')

# concatenate String object and a string

print(string1 +' world')

8. most popular form of operator overloading is of addition (+) operator. When two integers are passed to + operator, it will return the sum of two integers. When two strings are passed to + operator, it will return concatenation of two strings.

9. Inheritance.

Polymorphism.