Class attributes and updating those

In [1]:
```
class Employee:
    employee_id = 111
```

In [2]:
```
employee = Employee()
employee.employee_id
```

Out[2]: 111

In [3]:
```
Employee.employee_id = 222
```

In [4]:
```
employee2 = Employee()
employee2.employee_id
```

Out[4]: 222

Instance attributes

In [5]:
```
employee2.name = 'abc'
```

In [6]:
```
employee2.name
```

Out[6]: 'abc'

In [8]:
```
employee.name = 'dasd'
```

In [9]:
```
employee.name
```

Out[9]: 'dasd'

In [10]:
```
#Instance attributes are specific to the object, class attributes are specific to the c
```

In [11]:
```
employee.employee_id = 444
```

In [12]:
```
employee.employee_id
```

Out[12]: 444

In [13]:
```
employee2.employee_id
```

Out[13]: 222

In [14]:
```
#Python first searches for instance attributes and then if no match comes, it searches
#Instance attributes->Class attributes
```

Understanding Self parameter

In [15]:
```
class Employee:

    def employeeDetails():
        pass
```

In [17]:
```
employee = Employee()
#employee.employeeDetails()
#If you run this you will get this error:
#TypeError: employeeDetails() takes 0 positional arguments but 1 was given
#Bcz python calls the method like this->Employee.employeeDetails(employee)->error comes
```

In [18]:
```
class Employee:

    def employeeDetails(self):
        self.name = 'Souparna'
        print(self.name)
```

In [20]:
```
employee = Employee()
employee.employeeDetails()
print('\n')
Employee.employeeDetails(employee)
```

```
Souparna


Souparna
```

In [21]:
```
#If you dont use the objectname.instancename , then the lifespan of a an attribute is o
```

In [24]:
```
class Employee:

    def employeeDetails(self):
        self.name = 'Souparna'
        print(self.name)
        age = 30
        print(age)

    def printEmployeeDetails(self):
        print(self.name)
        print(age)
```

In [25]:
```
employee2 = Employee()
employee2.employeeDetails()
```

```
Souparna
30
```

In [27]:
```
#employee2.printEmployeeDetails()->NameError: name 'age' is not defined
```

Static methods and instance methods

In [30]:
```
#Instance methods are methods of the class that make use of the self parameter
#,to access and modify the instance attributes of the class
#All the methods used above are instance methods
```

In [32]:
```
#Static methods donot take the default self parameter
#Question is how will it avoid the error which python will throw is self is not passed
```

```
#Using 'DECORATOR', we distinguish between static and instance methods
```

In [37]:
```
class Employee:

    def employeeDetails(self):
        self.name = 'Souparna'
        print(self.name)

    @staticmethod
    def welcomeMessage():
        print('Hello World')
```

In [38]:
```
employee = Employee()
employee.employeeDetails()
```

```
Souparna
```

In [39]:
```
employee.welcomeMessage()
```

```
Hello World
```

In [41]:
```
#We need to have a way to intialize all the attributes of our object/class before they
#Python helps in doing that witht the help of a special method called the init method
#Special methods in python start and end with __
```

In [42]:
```
class Employee:

    def employeeDetails(self):
        self.name = 'Souparna'
        print(self.name)

    def welcomeMessage(self):
        print(self.age)
```

In [44]:
```
employee = Employee()
#employee.welcomeMessage() ->AttributeError: 'Employee' object has no attribute 'age'
```

In [45]:
```
#Lets use __init__ method now
```

In [46]:
```
class Employee:

    def __init__(self):
        self.name = 'Souparna'

    def welcomeMessage(self):
        print(self.name)
```

In [47]:
```
employee = Employee()
employee.welcomeMessage()
```

```
Souparna
```

In [49]:
```
#Make sure to initialize all attributes within init method, then the object becomes a f
```

In [50]:
```
#We need to have a way in which the init method takes in a parameter and assigns the at
```

In [52]:
```
class Employee:
```

```
    def __init__(self,name):
        self.name = name
        #self.name implies instance attribute name, name implies the parameter passed i

    def welcomeMessage(self):
        print(self.name)

employeeTwo = Employee('Bose')
```

In [53]:
```
employeeTwo.welcomeMessage()
```

```
Bose
```

In [55]:
```
#CLASS ATTRIBUTE->either inside class or classname.attributeName
#INSTANCE ATTRIBBUTE->objectname.attributeName
#Self parameter handling->objectname.methodName() is handled as classname.MethodName(ob
#init() method is an INITIALIZER in python, called when an object is instantiated
```

In [ ]:
```

```