**Section 8 Lesson 37: Python Code Walkthrough by Ahmed Sami**

**Introduction**   
Welcome, students. Today we will carefully examine a short Python snippet. In this session, I will explain each line of code in detail so you understand how Python classes and objects are used. We'll see how we define a class, create an instance, and attach attributes to that instance. Let us now explore the code line by line.

**Body**

1. **Line: "# new class"**   
   This is a comment. In Python, any text following the hash symbol is ignored during execution. Comments serve as notes for developers and help in understanding the structure or purpose of the code.
2. **Line: "class Plueprent:"**   
   Here we are declaring a new class called **Plueprent**. A class is like a blueprint for creating objects (instances). This statement tells Python to allocate a new namespace for the class, in which we will later define variables (attributes) and functions (methods).
3. **Line: " # vars"**   
   This is another comment inside the class. It is indicating that variable declarations, also known as class attributes, could be placed here. At this point, no variables are defined.
4. **Line: " #functions"**   
   This comment suggests that functions (or methods) that define the behavior of objects of this class might be added here. Once again, it is just a placeholder and does not execute any code.
5. **Line: " pass"**   
   The keyword **pass** is used as a placeholder in Python. It tells the interpreter that no action is performed. In this case, it allows us to define the class without adding any attributes or methods immediately. Think of it as a way to say, "I know a class should have contents, but I will fill it in later."
6. **Line: "# instance"**   
   This comment marks the beginning of the code section where we create an instance of the class. An instance is an object produced using the blueprint defined by the class.
7. **Line: "blokc1 = Plueprent()"**   
   Here we create a new object named **blokc1** of the class **Plueprent**. The parentheses () indicate that the class's constructor is being called. Since we haven't defined an initialization method, Python uses a default constructor.
8. **Line: "blokc1.name = "b1""**   
   In this line, we assign a new attribute **name** to our instance **blokc1**. The value **"b1"** is a string assigned to this attribute. Notice that this attribute was not declared within the class itself; Python's dynamic nature lets us add attributes directly to an object.
9. **Line: "blokc1.value = "4*4""*** *Similarly, we add another attribute* ***value*** *to the instance* ***blokc1*** *and assign it the string* ***"4*4"**. Although it appears to represent a mathematical expression, it is stored as a sequence of characters rather than being computed as 4\*4.

**Conclusion**   
To summarize, this code demonstrates the basics of Python’s object-oriented programming:

* A new class **Plueprent** is defined, with placeholders for attributes and methods.
* An instance **blokc1** of this class is created.
* Dynamic attributes **name** and **value** are assigned to the instance after its creation.

This simple example highlights Python's flexibility in defining classes and managing their attributes. As you progress in your studies, you will learn how to use more formal methods like the **<strong>init</strong>** constructor for initializing attributes at the time of instance creation. Thank you for your attention.

section 8 lesson 37 python code

# new class

class Plueprent:

# vars

#functions

pass

# instance

blokc1 = Plueprent()

blokc1.name = "b1"

blokc1.value = "4\*4"

**Section 8 Lesson 38 – Lecture by Ahmed Sami**

**Introduction:**   
Good morning, everyone. Today we will study a series of Python code examples that illustrate object-oriented programming for a car company. I will walk you through each line of the code, explaining how classes and objects work and how we can use methods to display information. Let’s explore the structure and logic behind these examples.

**Body:**

1. **Defining the Car Class:**   
   Every example begins by defining a class called Car.
   * **Line:**   
     class Car:   
     This line declares a new class named Car. A class in Python is like a blueprint for creating objects. In our context, each object will represent a car with specific attributes.
2. **The Constructor Method:**   
   Next, we see the constructor method, which initializes the attributes of each car instance.
   * **Line:**   
     def \_\_init\_\_(self, brand, wheels, doors, horsepower):   
     Here, the special method \_\_init\_\_ is defined. It is automatically called when an object is created. The parameters brand, wheels, doors, and horsepower are provided to set up the car’s properties.
   * **Lines:**
3. self.my\_brand = brand
4. self.my\_wheels = wheels
5. self.my\_doors = doors

self.my\_horsepower = horsepower

These lines assign the passed parameters to instance variables (attributes) of the object. The prefix self. indicates that these variables belong to the specific instance of Car. For example, if we pass the brand as "Audi C420", it gets stored in self.my\_brand.

1. **Defining a Method to Display Car Information:**   
   A method named get\_my\_car is defined to print the car’s details in a formatted message.
   * **Line:**   
     def get\_my\_car(self):   
     This defines a new method within the Car class.
   * **Line:**

print(f"Your car brand is {self.my\_brand} and it has {self.my\_wheels} wheels, {self.my\_doors} doors, and {self.my\_horsepower} horsepower")

Here, a formatted string is printed. This message uses the attributes stored in the object to display information clearly. Note that the method does not return any value; it simply prints the message to the console.

1. **Creating an Object and Accessing Attributes:**

**First Code Block:**

* + **Line:** audi = Car("Audi C420", 4, 2, 450) An object named audi is created by calling the Car class constructor. The properties passed are:
    - Brand: "Audi C420"
    - Wheels: 4
    - Doors: 2
    - Horsepower: 450
  + **Line:**   
    print(audi.my\_horsepower)   
    This line directly accesses and prints the my\_horsepower attribute of the audi object, thus displaying 450 on the console.

**Second Code Block:**   
In this block, we create two objects and directly access their attributes.

* + **Lines:**

audi = Car("Audi C420", 4, 2, 450)

safira = Car("safira", 4, 4, 300)

Two objects are instantiated: one for Audi and another for Safira.

* + **Lines:**

print(audi.my\_doors)

print(safira.my\_doors)

The program prints the number of doors for each car. For audi, this value is 2, and for safira, it is 4.

**Third Code Block:**   
Here, we use the class method to display the details of the audi car.

* + **Line:**

audi.get\_my\_car()

Invoking the get\_my\_car method on the audi object prints all details (brand, wheels, doors, horsepower) for the Audi car in a single, formatted message.

**Fourth Code Block:**   
Finally, the method is used to print the details of the safira car.

* + **Line:**

safira.get\_my\_car()

By calling this method on the safira object, the information specific to "safira" is output to the console. This demonstrates the reusable nature of class methods across different objects created from the same class.

**Conclusion:**   
Today’s lecture demonstrated the concept of object-oriented programming in Python using a simple Car class. We learned how the constructor method initializes attributes, how class methods can encapsulate behaviors such as printing detailed information, and how to create multiple objects with different properties. This modular approach to programming not only keeps code organized but also enhances reusability. I encourage you to review each code segment carefully and experiment by creating your own car objects to solidify your understanding of Python classes and object instantiation.

section 8 lesson 38 python code

# Program for a car company

class Car:

def \_\_init\_\_(self, brand, wheels, doors, horsepower):

self.my\_brand = brand

self.my\_wheels = wheels

self.my\_doors = doors

self.my\_horsepower = horsepower

def get\_my\_car(self):

print(f"Your car brand is {self.my\_brand} and it has {self.my\_wheels} wheels, {self.my\_doors} doors, and {self.my\_horsepower} horsepower")

audi = Car("Audi C420", 4, 2, 450)

print(audi.my\_horsepower)

# Program for a car company

# safira brand

# printing audi.my\_doors and safira.my\_doors

class Car:

def \_\_init\_\_(self, brand, wheels, doors, horsepower):

self.my\_brand = brand

self.my\_wheels = wheels

self.my\_doors = doors

self.my\_horsepower = horsepower

def get\_my\_car(self):

print(f"Your car brand is {self.my\_brand} and it has {self.my\_wheels} wheels, {self.my\_doors} doors, and {self.my\_horsepower} horsepower")

audi = Car("Audi C420", 4, 2, 450)

safira = Car("safira", 4, 4, 300)

print(audi.my\_doors)

print(safira.my\_doors)

# Program for a car company

# Print details of audi brand

class Car:

def \_\_init\_\_(self, brand, wheels, doors, horsepower):

self.my\_brand = brand

self.my\_wheels = wheels

self.my\_doors = doors

self.my\_horsepower = horsepower

def get\_my\_car(self):

print(f"Your car brand is {self.my\_brand} and it has {self.my\_wheels} wheels, {self.my\_doors} doors, and {self.my\_horsepower} horsepower")

audi = Car("Audi C420", 4, 2, 450)

safira = Car("safira", 4, 4, 300)

audi.get\_my\_car()

# Program for a car company

# Print details of safira brand

class Car:

def \_\_init\_\_(self, brand, wheels, doors, horsepower):

self.my\_brand = brand

self.my\_wheels = wheels

self.my\_doors = doors

self.my\_horsepower = horsepower

def get\_my\_car(self):

print(f"Your car brand is {self.my\_brand} and it has {self.my\_wheels} wheels, {self.my\_doors} doors, and {self.my\_horsepower} horsepower")

audi = Car("Audi C420", 4, 2, 450)

safira = Car("safira", 4, 4, 300)

safira.get\_my\_car()