## 

## CLASS

1. SYNTAX is class Car:

When we write the word “class”, this will initiate the class.

Car is the name of the class .

It has to end with a :

1. Now we can create an object using “ **car\_1 = Car()** “ . Make sure that the object function is indented.
2. NOTE : We need to create the name of the class using the PascalCase

Name of the class should have the first word capitalized for example : class CarCamshaftPulley :

Attributes to a CLASS :

**Class User :**

**Pass**

**User\_1 = User()**

**user\_1.id=”001”**

**user\_1.user=”Tarn”**

Id , user are all attributes that we have associated with the User class and all these attributes are held in the class.

Now we wanted to keep creating more users as in User\_2 , User\_4 , User\_3 and we want to add both the ID and the user attribute to each one of these user , the process will be a lot more tedious and in order to avoid that we would :

CONSTRUCTOR : or intialising the object , we can set the variables to their starting value.

**Class Car:**

**Def \_ \_init\_ \_(self):**

**# initializing attributes gets assigned to the function**

**Class User :**

**Def \_ \_ init\_ \_ (self):**

**print(“new user being created”)**

**User\_1 = User()**

**User\_2= User()**

Now because we have assigned an init function to the class each time we create an object from the class or we use the class the init function gets **print(“new user being created”)**triggered .

NOTE : Attributes are the things that the object will have. They are variables that are associated with the object.

If we need to select the number of seats in a car inside the init function we will use

**Class Car:**

**Def \_ \_init\_ \_(self), seats:**

**Self.seats = seats**

So if we were to pass the command to the class

**My\_car = Car(5)**

This will take the value of the seat as 5 and action the code.

This would be identical to if we were to create a object called Car first and then add an attribute called seats to it.

**Class Car:**

**whatever we want the class to be**

Then we create an object.

**Car1 = Car()**

**Car.seats = 5**

Let’s build a proper self initializing class.

**Class User:**

**Def \_ \_ init \_ (self, user\_id, username):**

**self.id = user\_id**

**self.name= username**

o we have created the parameters and assigned a value to each one of them.

Now, lets go back to our constructor. Before it was empty but now we will pass attributes through it:

**User\_1 = User(“001”, “Tarn”)**

**print(User\_1.id)**

So this will output 001.

Need to make sure that we provide user\_id and username i.e the number of attributes that are passed when creating an object from the class have to match the number of attributes defined in the init function.

WHEN WE NEED CONSTANT VALUE FOR A SPECIFC ATTRIBUTE IN THE CLASS

For example if we are coding the instagram app , all the user\_Id and username will change but the number of followers that each account begins at will be 0 and it will grow from there.

It makes no sense if we have to pass the value of 0 each time we create an object because of the rule that the number of attributes passed through the object has to match the number of attributes that are specified in the init function.

So in order to resolve this issue we use,we have to provide a default value to the attribute instead of putting it **self.id = user\_id** equal to a variable.

**Class User:**

**Def \_ \_ init \_ (self, user\_id, username):**

**self.id = user\_id**

**self.name= username**

**self.NumberOfFollowers = 0**

**User\_1 = User(“001”, “Tarn”)**

**print(User\_1.NumberOfFollowers)**

The output of this code will be 0 as that is what the value of the attribute is assigned to be.

ADDING METHODS TO CLASS

1. Attributes is something that a class has.
2. Method is something that the class does.
3. Seats that we created before was an attribute for the class car.
4. Method

Def enter\_race\_mode():

Seats = 2

So when this function will be called the seats will be changed to 2.

1. When a function is attached to an object its called a method.

Class car:

Def enter\_race\_mode():

Self.seats=2

So all we need to do to access this method is.

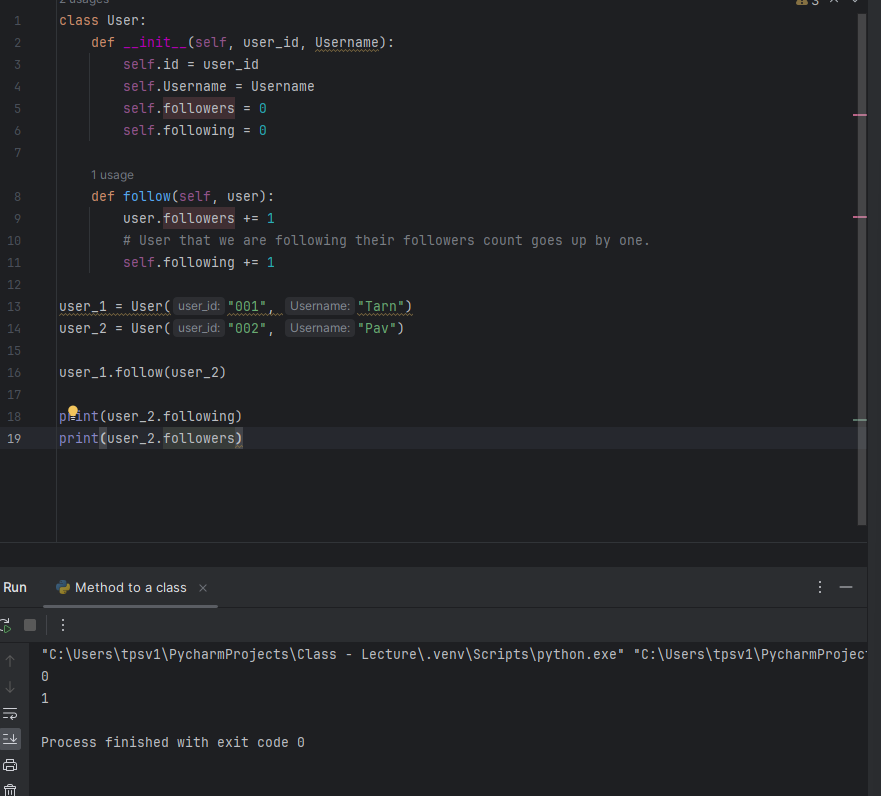
We know the object we created earlier in the code was my\_car

My\_car.enter\_race\_mode()

1. A method unlike a function always needs to have a “self” parameter as the first parameter. So when it is called the function knows that it has been called.

def follow(self):

1. We dont usually see self function when we are creating the objects but we see it a lot when we are creating a class.

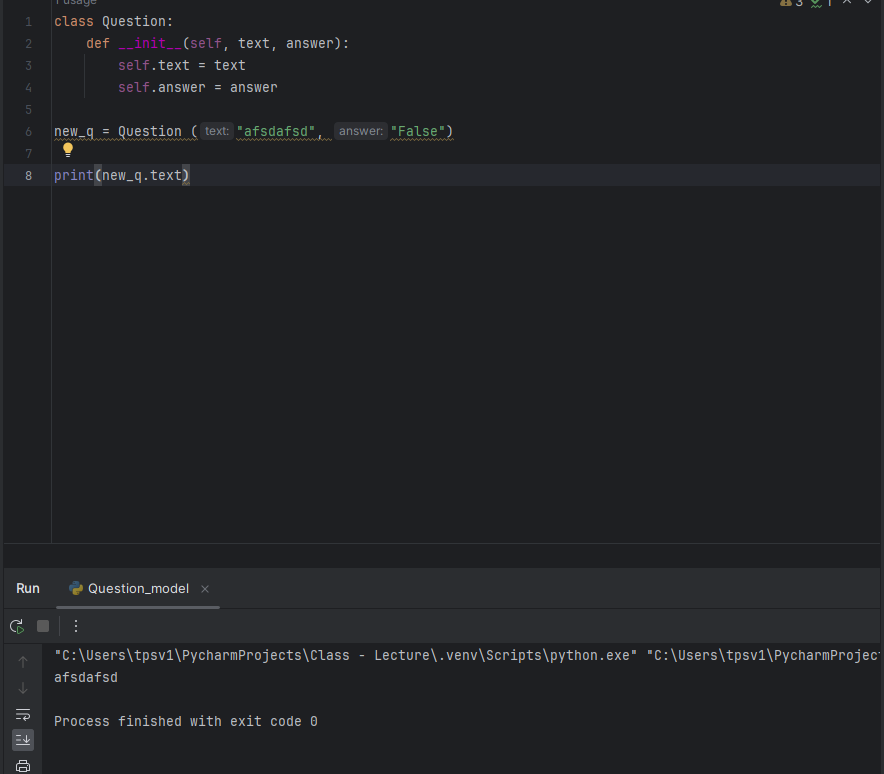


1. For the True/False quiz. We can create new objects. Foe example if we have a question object it might have a Text attribute that holds the question and it might also hold a answer attribute

QUESTION attribute (“text” , “ Answer”)

New\_q = Question (“2+3=5” , “True”)

Now , attributes will be assigned to text and answer because of the constructor and the initializing factor.



We created a class called Question and added two initiating attributes to it called text and answer.

Then we created an object from it called new\_q , passed attributed for the object and finally access printed the attribute out.

question\_bank =[]

for question in question\_data:

q\_text = question["text"]

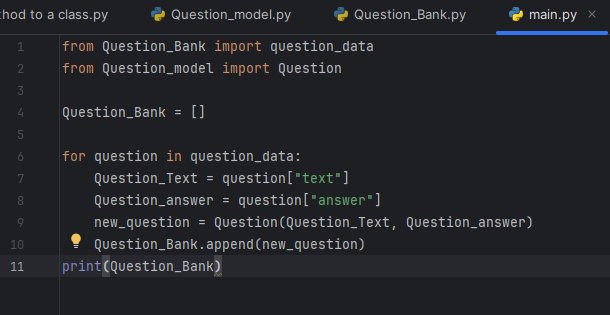
q\_answer = question["answer"]

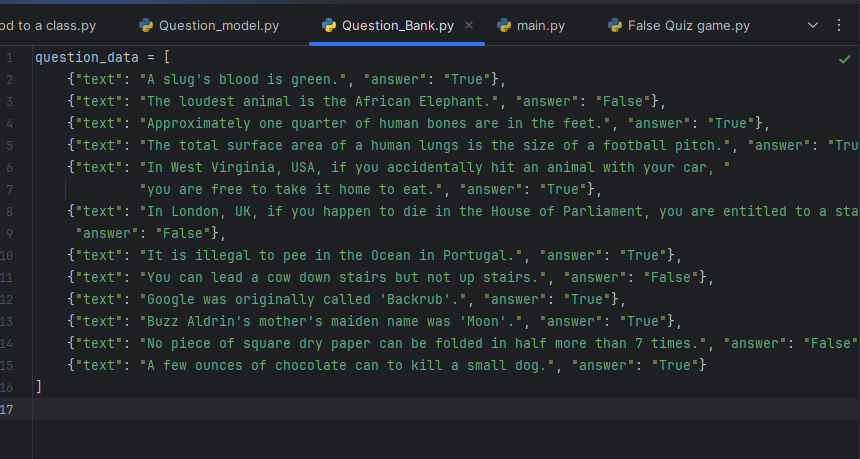
new\_question = Question(q\_text, q\_answer)

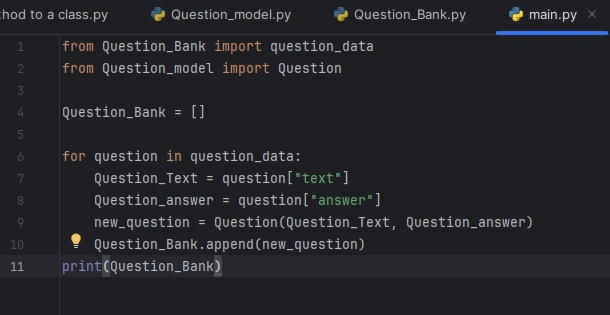
#Question is the class.

question\_bank.append(new\_question)

Print (question\_bank[0].text)







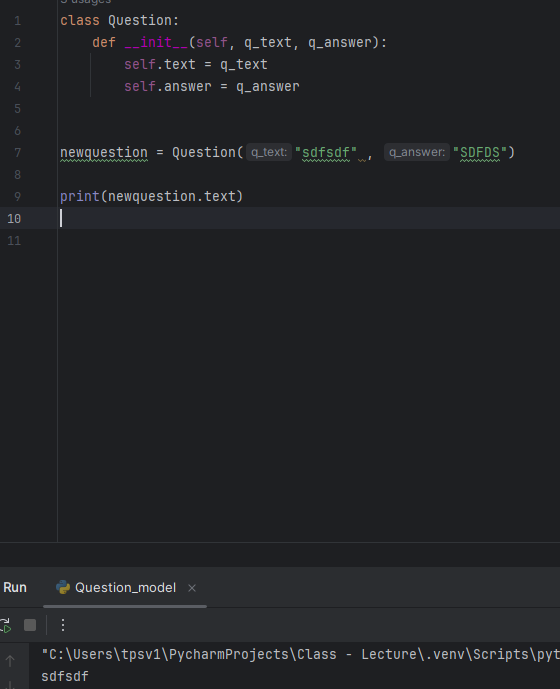
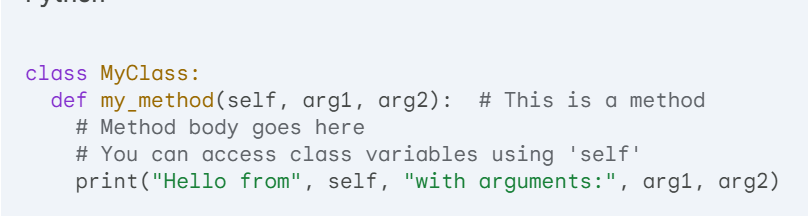
The self argument in Python's \_\_init\_\_ function (also known as the constructor) is essential for working with objects within a class. Here's why it's important:

**Referencing the Object Instance:**

* Classes are blueprints for creating objects. When you create an object (instance) from a class, the \_\_init\_\_ function gets called automatically.
* The self argument within \_\_init\_\_ refers to the specific object instance that's being created. It acts like a placeholder, allowing you to assign attributes and methods to that particular object.

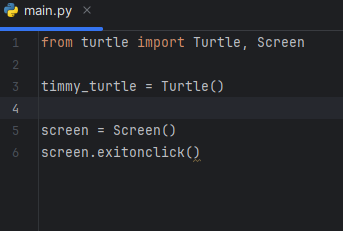
**Initializing Object Attributes:**

* Inside \_\_init\_\_, you can use self to define the object's initial state. You can assign values to attributes (variables that belong to the object).

1. Basically if we need to have multiple attributes within a class we will have to create an init self , so that we can reference a specific attribute from the class and call for the attribute further down the code.
2. HOW TO USE INIT FUNCTION TO ASSIGN values to attributes 
3. 

DAY 18

TURTLE GRAPHICS , TUPLES and IMPORTING MODULES>

1. Python TURTLE : It is a way for us to be able to draw on the screen. 
2. We can set the colour for the pen. It will be based on the colour red , green and blue.
3. If we need to import all the classes and the methods from an class.

We can simply use

import turtle

from turtle import \*

In This case if we need to create an object using the class we have , we can simply use

Timmy = turtle()

But us writing the \* makes the code less expressive.

It is better if we write down the classes we actually we want to download.

1. ALIAS name

What this does is that we can import a module and assign a name to it . So that instead of typing the long module name each time we simply type in out Alais name

We can simply write

Import turtle as t

TUPLES

1. Tuple is similar to list in python. Only difference is tuple are carved in stone as in the values inside of tuple cannot be changed whereas they can be changed inside of a list.

**# Tuple**

**my\_tuple = (1, 2, 3, 4, 5)**

**# This would raise an error since tuples are immutable**

**# my\_tuple[0] = 10**

# List

my\_list = [1, 2, 3, 4, 5]

# This would change the first element to 10

my\_list[0] = 10

Tuple is used to describe the reg green and blue percentage of each of the colors to form a different color.

PLEASE NOTE : In order to use the colormode inside of turtle we need to change the color module for the class that is “turtle” and not the object “Turtle” that we have created.

1. When using a function and passing it into another function we dont have to use the ()

Def add (n1, n2):

Return (n1 + n2)

Def calculator (n1, n2, Function)

Return Function(n1, n2)

Calculator (2, 3, add)

When we use functions within other functions we should attempt to use the keyword arguments and not positional arguments.

def calculate\_area(length, width):

return length \* width

def calculate\_room\_area(room\_length, room\_width):

# Using positional arguments (prone to errors if order changes)

return calculate\_area(room\_length, room\_width)

# Using keyword arguments for clarity and flexibility

def calculate\_room\_area(room\_length, room\_width):

return calculate\_area(length=room\_length, width=room\_width)