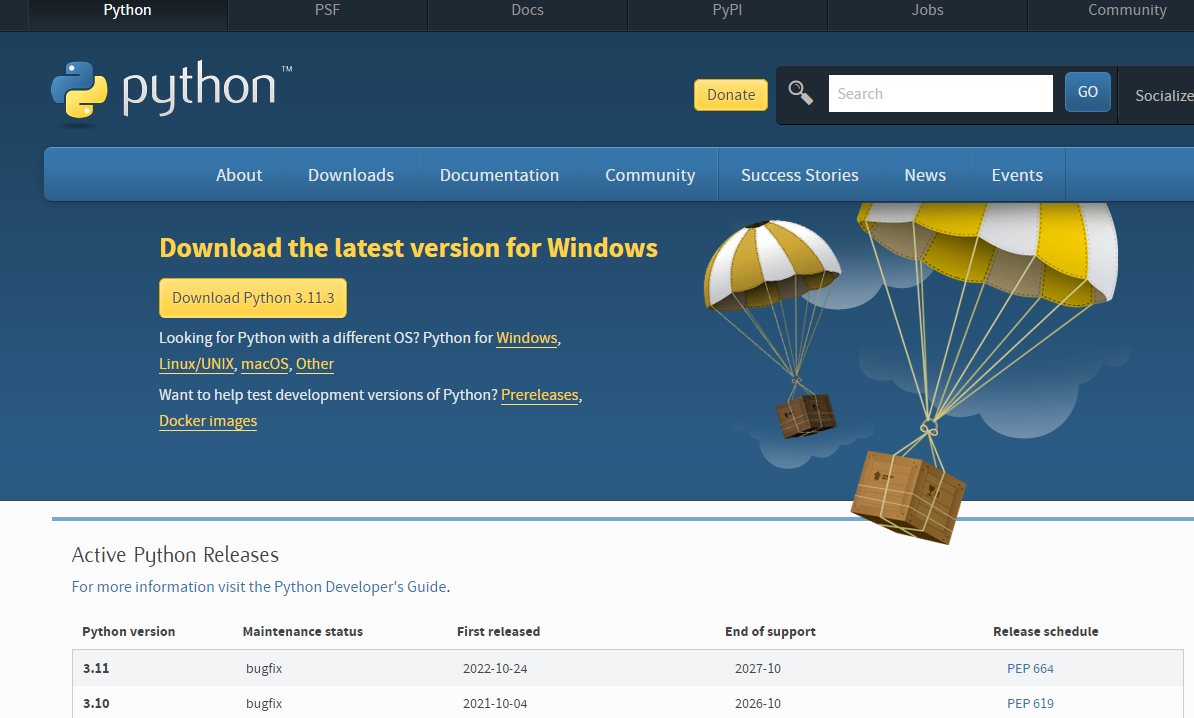
**Bytewise fellowship: Data Engineering**

**Week 4 progress:**

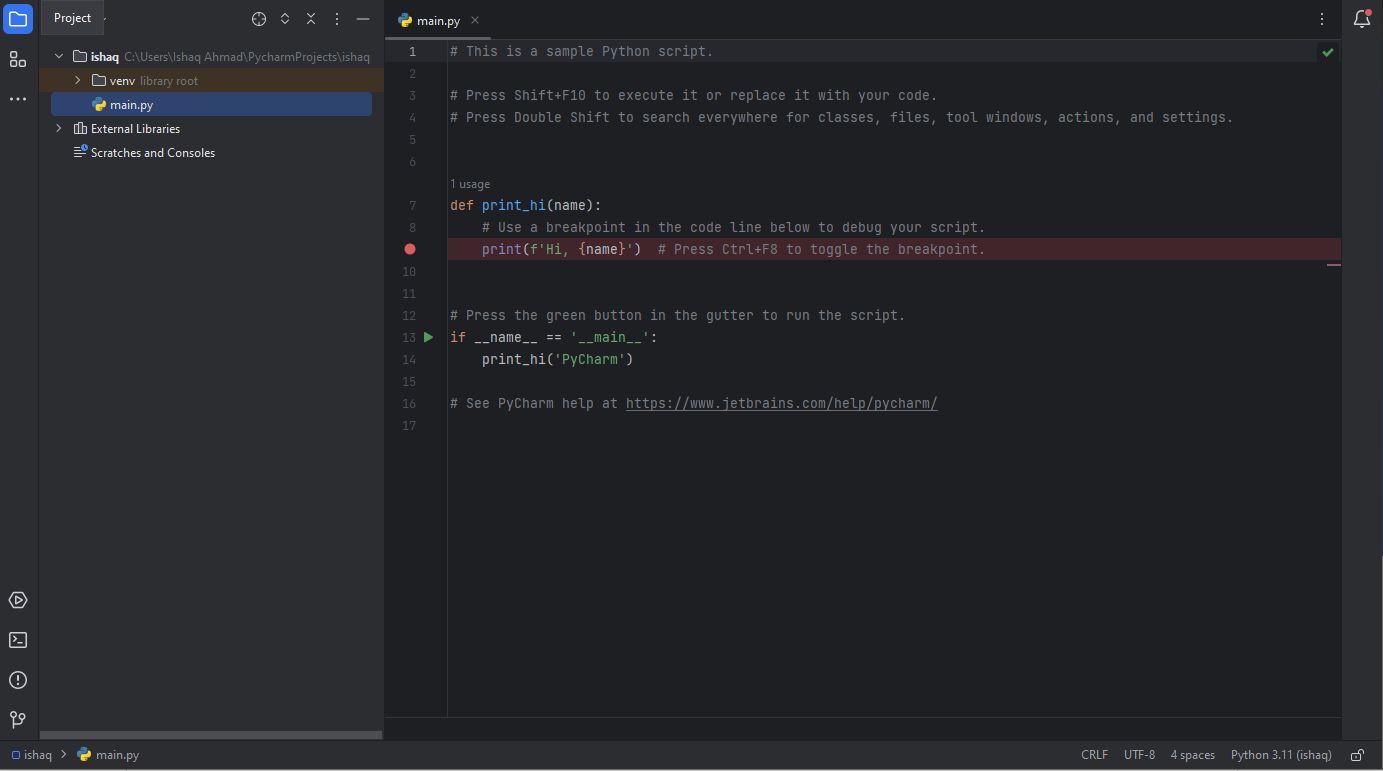
**Introduction to python:**

1. **Install Python:**

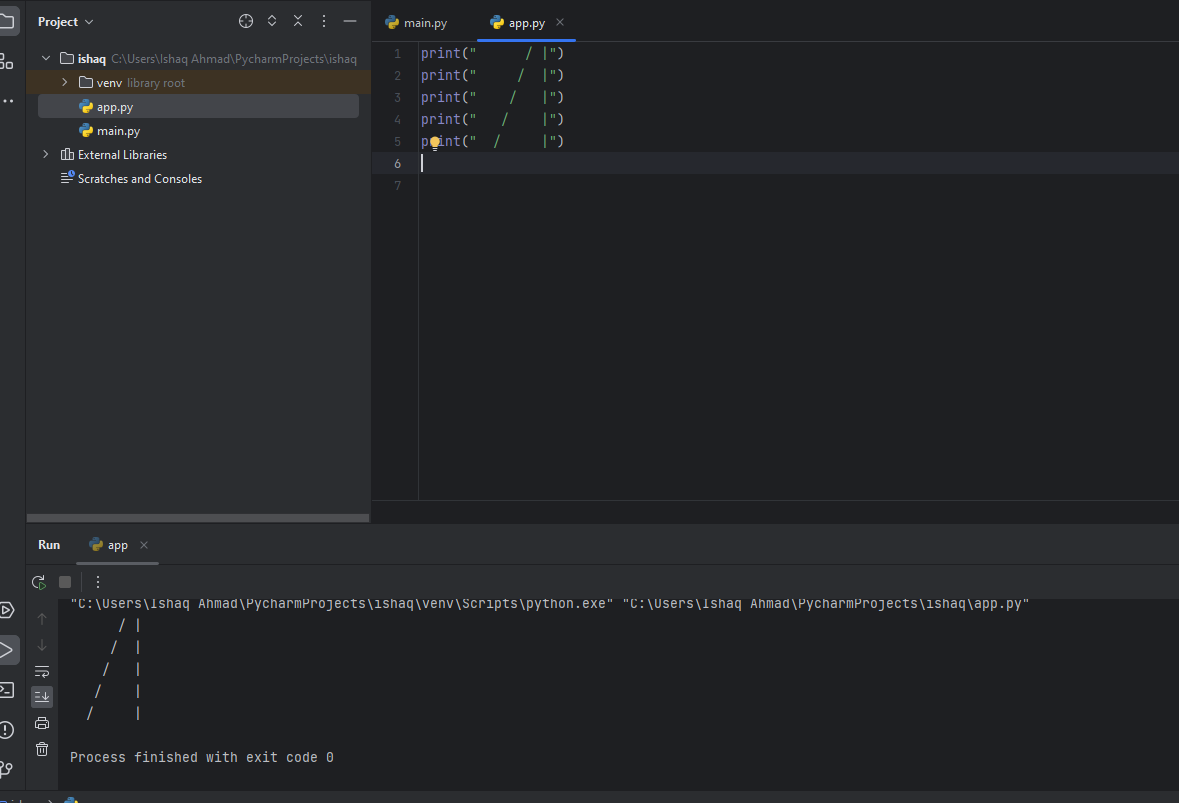
I have downloaded python in my PC.



I downloaded pycharm as well.



**Run shape:**



1. **Declare a Variable:**

Variables are one of the concepts found in all programming languages. You might as well say that without a variable, you can't program, and that's not an exaggeration.

As mentioned in the video, think of a variable as a kind of box containing a value. This box is itself stored on a shelf among many others, in a gigantic warehouse. The location of each box is very precisely recorded, just as your computer records the exact location of your variable in its memory.

A value is what you will store in a variable. To return to the warehouse analogy, there are several boxes for storing different values. For example, if you work in a bank, you might want to store information about a customer in different boxes, such as their checking account balance and their savings account balance. We will also need to perform different operations on these boxes like emptying them, adding money, transferring the contents from one to another, etc. Variables will let you do this!

To get to the contents of each box, you will need to label them. The process is similar in programming: each variable is given a name.

**Steps for declaring variable:**

Use clear variable names

Use explicit variable names

Follow a typographic convention

**Create a Variable**

I have declared two variables here, **checkingAccount** and  **savingsAccount** , by storing the values 500 and 1,000, respectively.

The rules of arithmetic apply in Python, in particular the order of operations, but as in ordinary mathematics, you can use brackets to rearrange the order of the calculations. See how to do this in Python:

1. **Handle Variable Types:**

**Understand Why We Need Variable Types**

You looked at different types of variables in the previous chapter, but there is much more to know about the subject!

The types encountered so far are called primitive types. They exist in Python—a bit like atoms. These are the simplest types of variables; they are the foundation of all computer operations and programs. In the same way that atoms can be combined to make more complex molecules, you can combine primitive types to create much more complex variable types, as you will see in the next parts of this course. For now, you will explore numeric types and strings in a little more depth. Let's go!

**Numeric Variables**

Numeric variables can be broken down into two distinct types:

Integers, which correspond to the set of positive or negative integers (1, 2, 0, 123, -3, etc.)

Decimals, which, in addition to integers, include all decimal numbers (2.50, 5.99, -1.20, etc.)

Start with the one you are already familiar with: integers. Integers are declared like any other variable, by associating a value to a variable name.

**Mix Several Numeric Variables**

It is important to keep in mind how the different numeric types can be mixed together and what the potential consequences are. If you mix different types, the most complex will be the one kept for the final result. For example, an integer value can be stored as a float, as seen above with the width variable, but the opposite is not possible if there are numbers after the decimal point! The float is therefore the most complex type: if you mix an int with a float , the result will always be a float , whatever operation is performed or whatever the result is.

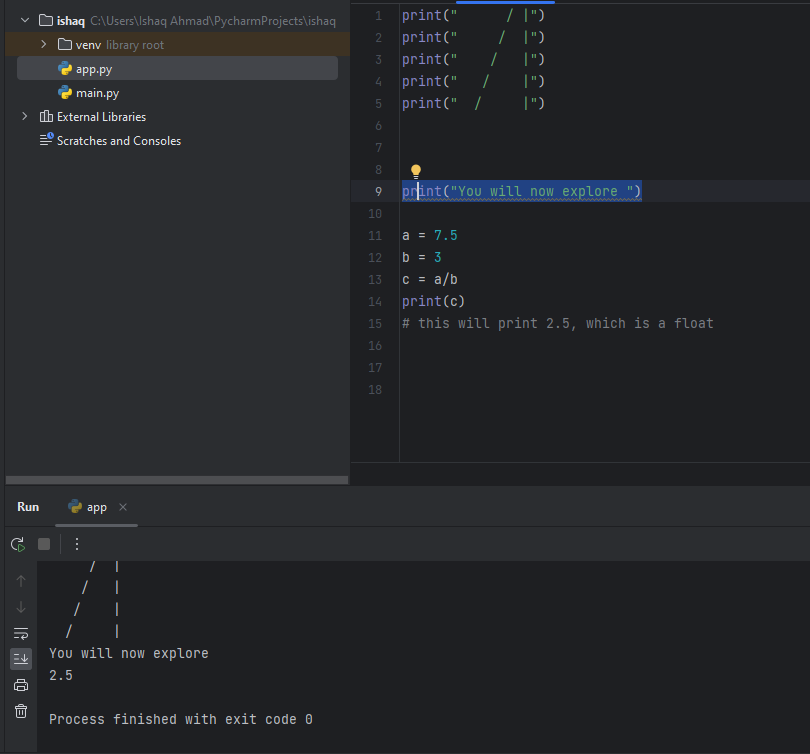
a = 7.5

b = 3

c = a/b

print(c)

# this will print 2.5, which is a float



**Character Strings**

You will now explore a little more about character strings, which let you store text in your variables. First, a bit of semantics: we call them character strings, because Python does not consider these variables as text, as such, but as a set of characters put together. This is how you can define character strings in Python (you can use either single or double quotes):

city = 'New York'

film = 'taxi driver'

emptyString = ''

city = 'New York'

film = 'taxi driver'

emptyString = ''

1. **Write Your Own Functions**

Since the beginning of this course, you have used different functions, such as the print() function of the different cast functions like int() or str() . We will now take the time to define what a function is, what it is used for and how you can create your own functions—you are going to find out everything there is to know about functions!

During your data analysis, you will regularly have to use groups of statements several times for a very specific purpose. One of the fundamental principles for any computer programmer is to get maximum results for minimum effort (there is even a saying that a good programmer is a lazy programmer!). It is thanks to this somewhat "lazy" but very effective principle that the idea of functions came about. Functions can group several statements in a block which will be called using a name.

Functions are not specific to Python; they are present in all computer languages. They can:

reuse a portion of code already written just by stating the function name—so you don't have to rewrite the whole portion of code each time.

simplify code and make it more readable!

def functionName():

# statements

# that can go

# on several

# lines

Another example

def printPerimeter():

dimension1 = 6

dimension2 = 4

dimension3 = 3

perimeter = dimension1 + dimension2 + dimension3

print(perimeter)

printPerimeter() # => 13

**Define the Parameters**

To overcome this limitation, you must make your function accept external numbers. You can do this by defining parameters.

In Python, parameters, just like the name of the function, are defined when the function is written. This is how it would look with the above example:

def printPerimeter(dimension1, dimension2, dimension3):

perimeter = dimension1 + dimension2 + dimension3

print(perimeter)

Parameters are variables defined during the declaration of the function, specified inside brackets. Now you can use your function with any existing triangle:

printPerimeter(10, 11, 4) # => 25

printPerimeter(2, 2, 3.5) # => 7.5

Python is an object-oriented programming language—this means that in Python, everything is an object! In this chapter, you will see what this means and how you can use it in a practical way. Start by looking at a few objects from everyday life, such as pens, books, smartphones, computers, etc.

Objects can have many different shapes and characteristics, but you can classify different versions of the same object into a category or group. That's why it's easy to recognize a chair in a store, for example, although its appearance (shape, color, etc.) can vary greatly from one model to another.

It is by observing the common points between different objects that you are able, mentally, to classify the objects in the same group or category!

For example, there are different types of books, but they all have a title, an author, a back cover, etc. All books share different attributes that let you classify them in a well-identified category: books.

**Classes: Object Models**

In programming, this concept of a group or category of objects is called a class. A class can be considered as the construction diagram for an object that will define the characteristics of all objects of this type and their features. From this class, you will be able to create different models of an object.

Let's take a concrete example with a Car class. The plan of a car can be defined by:

its characteristics, called attributes: it must have four wheels, a color, a shape, an engine power, etc.

its functionalities, called methods: it can drive, brake, etc.

So, from this plan, you can create different car models:

An ordinary family car, green, medium power (110 hp)

A sports car, red, relatively powerful (180 hp)

A small blue city car, not very powerful (90 hp)

etc.

And no matter what the car model, they are all capable of driving or braking, but not with the same performance!

In summary, a class is the outline of an object, defining its attributes and methods. From the same class, you can therefore create several objects of the same type, but with different attributes—these are called class instances.

**Focus on Methods**

As we said before, in Python, everything is an object. This means that, without knowing it, since the beginning of this course, you have been manipulating objects! Consider the following lines of code to illustrate this:

var1 = 14

var2 = 1031