Variables

Introduction to Variables

To start, let's create a new Python file called "variables." This can be done by going to the project tool window on the left-hand side and right-clicking on your "first-pycharm-project" folder. From here you can then select New > Python file. Just like before, you'll need to specify the desired filename in the window that appears. In this case, I've gone with the name `variables.py.`

With our newly created `variables.py` file we can now delve into the concept of variables.

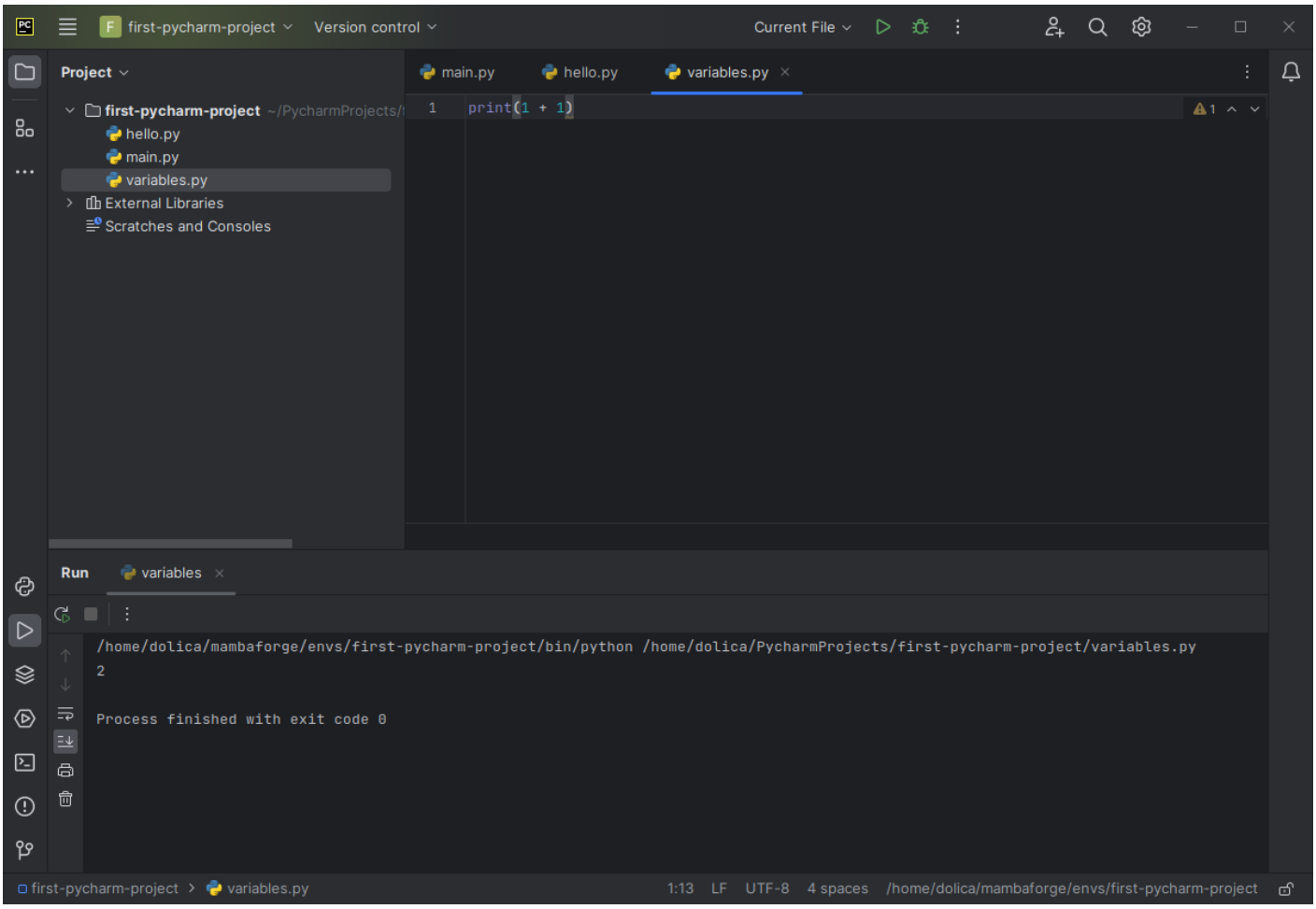
Why Use Variables?

Python can be used as a calculator as it allows us to perform calculators with whole-numbers. In programming, we refer to these whole-number values as **integers** or **ints**.

Consider a basic task: adding two whole numbers using Python. You can achieve this by entering the following code into your "variables.py" file and running it:

```
print(1 + 1)
```

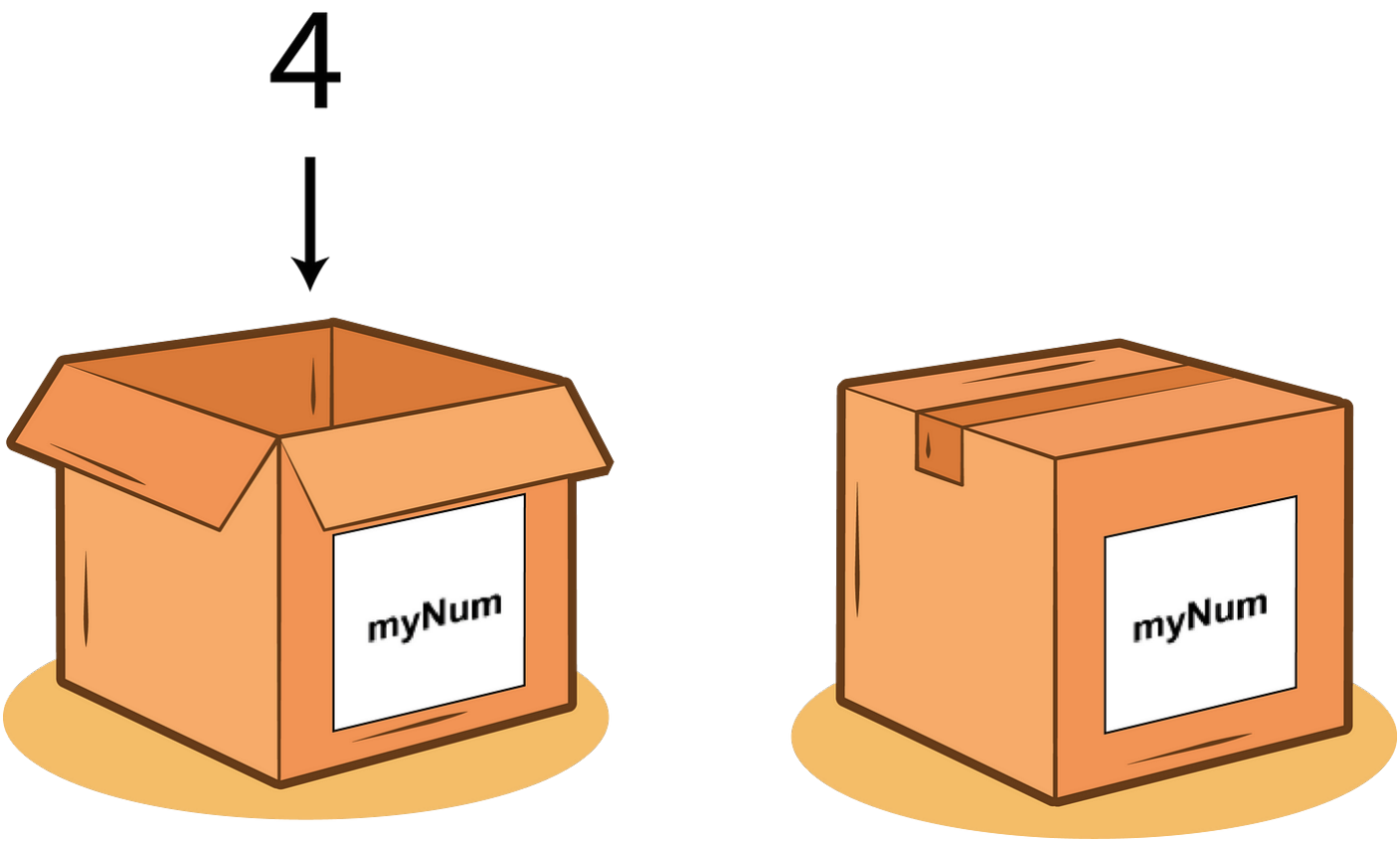
As expected, Python tells us that 1 + 1 gives a result of 2.



However, you may want to use a certain value more than once. It is often useful to store data in our code for repeated use. This is where the concept of **variables** comes into play.

What Are Variables?

In the world of programming, variables are like containers that hold information. Imagine them as labeled boxes where you can store different types of data. This data could be numbers, words, sentences, or something else your program needs to work with.



Variables allow you to give a name to a piece of data, making your code more readable and organised.

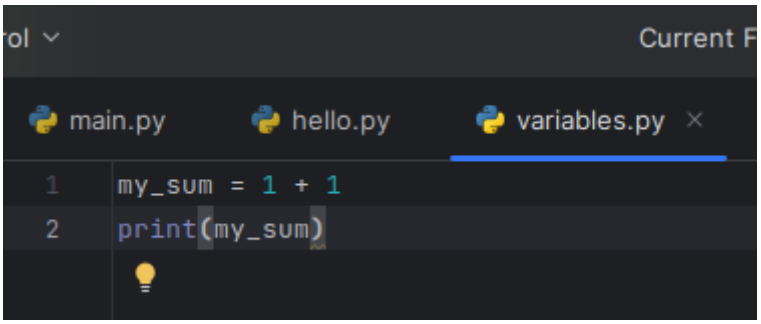
Variable Assignment

We use the assignment operator `=` to put data inta a variable. Now we can find a sum of two integers like we did before, except this time we are able to *store* the result so that it can be accessed again later.

```
my_sum = 1 + 1
```

We have now created a box with the label `my_sum` that contains an integer of the value 2. Now we can display this by giving our `my_sum` variable to the `print()` command.

When combined, these two lines of code appear as follows:



Naming Variables

Python enforces specific rules for naming variables. Variable names must begin with a letter or an underscore (_) and can be followed by letters, numbers, or underscores. However, they cannot begin with a number or contain spaces or special characters.

Self-Documenting Code

It's also a good idea to make sure your variable names are clear and meanginful. For example, `recipe_name` or `ingredient_count` are good variable names, as they help us understand what a variable is for and what a bit of code is doing.

Let's take the example of code used for finding the area for a rectangle:

```
a = x * y
print(a)
```

```
rectangle_area = width * length
print(rectangle_area)
```

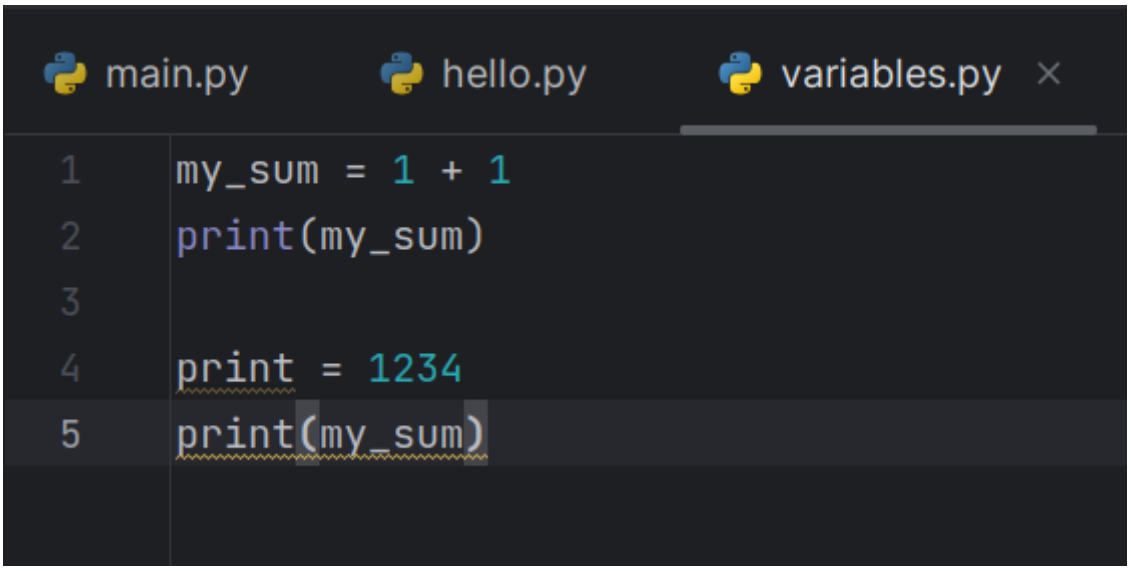
Both code snippets accomplish the same task, but the second one requires less mental effort to understand its purpose. This is an important consideration when writing code and selecting variable names.

When you choose names in your code that make it more "self-explanatory," this is often referred to as **self-documenting code.** This is a *very* good habit to pick up.

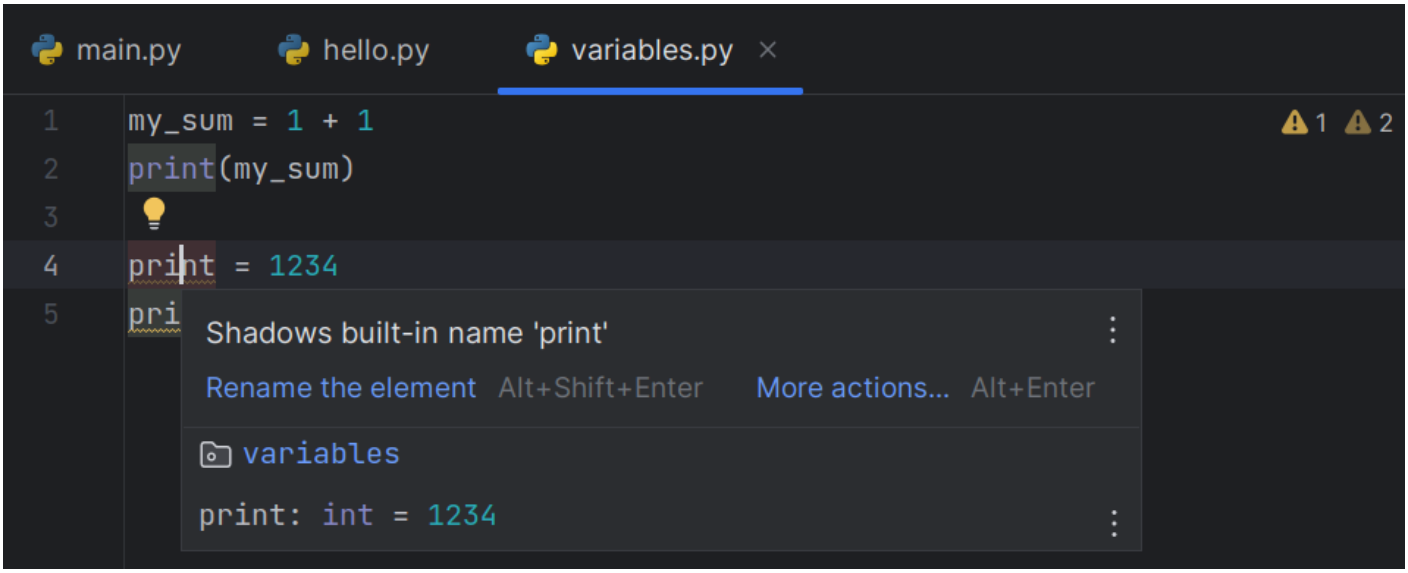
Reserved Words

Avoid using variable names that coincide with **reserved words** in Python. Python reserves certain keywords such as `while`, `for`, `if`, `else`, `import`, `exit`, and others for its built-in functions and tools. By using these keywords for variables, we are in a sense "confusing" Python, as it no longer knows where to find the built-in tools that it associates with those names. This is almost guaranteed to make your program misbehave.

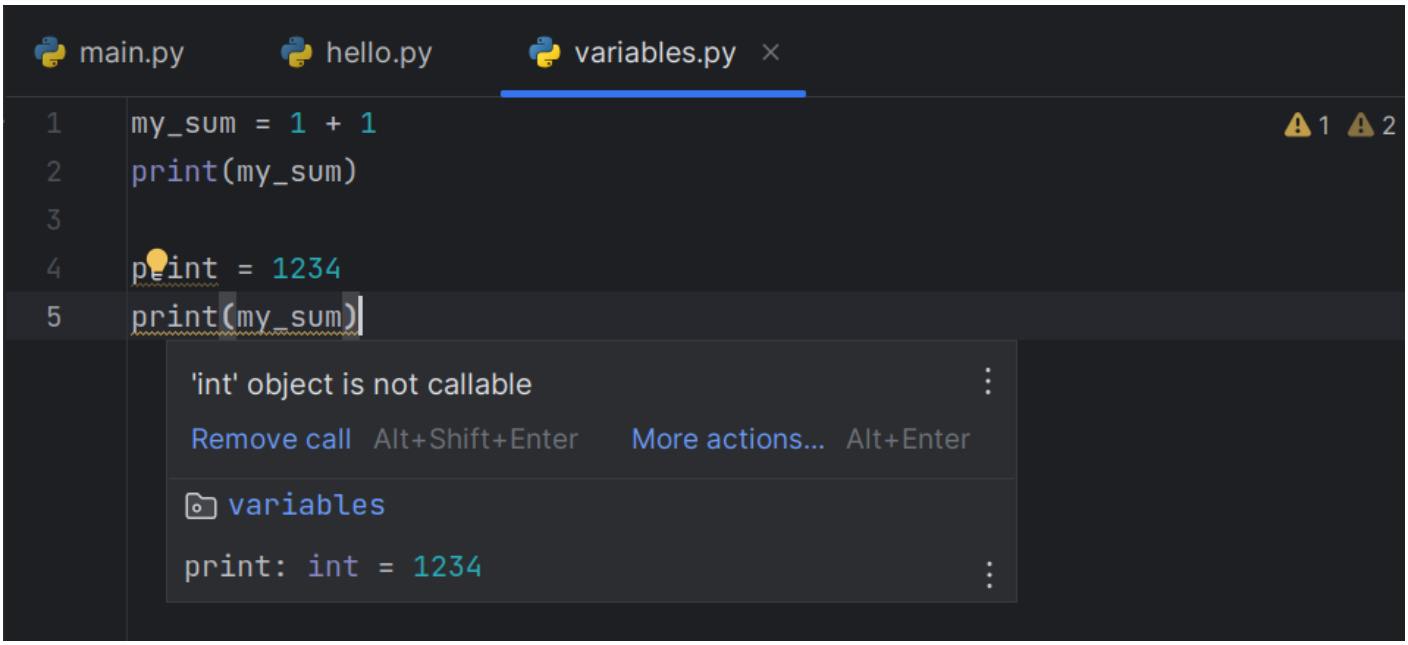
For example, let's talk about the word `print` in Python. In Python, we use the `print` command to show information from our code on the screen. Now, imagine what happens when we first use `print` as a name for a variable in our program, but then also try to use `print` as a command once more later:



You'll see that `print` on line 4 now has a squiggly line beneath it. Moving the cursor over the `print` on line 4 gives us the following message:



This message is telling us that the built-in print command is being "shadowed" by something else. Now let's look at the `print(my_sum)` from line 5:



Like before, moving the mouse over this bit of code gives us another message. This time we're being told that certain things are "callable" in Python, but that integers do **not** have this property of being callable. In essense, we are misuing the integer by attempting to do something with it that cannot be done. While the exact meaning of these messages may not be clear to you now, understand that these are some of the first signs that alert you to potential problems within a piece of code.

One of the useful features of IDEs their ability to detect and highlight potential issues in your code before you've even run it. This helps you avoid the inconvinience of executing your code, only to have it crash before completing its intended task.

We can also see that the IDE has suggested some solutions. Where we assign 1234 to a variable called `print`, it suggests using a comment built into PyCharm that will help rename this element to something other than `print`. It also advises us to delete the call on line 5. While renaming the variable on line 4 is the correct action, we'll keep things this way for now in order to better understand how errors manifest in Python.

Now when we run our code we get the following output:

```
/home/dolica/mambaforge/envs/first-pycharm-project/bin/python /home/dolica/PycharmProjects/first-pycharm-project/variables.py
2
Traceback (most recent call last):
  File "/home/dolica/PycharmProjects/first-pycharm-project/variables.py", line 5, in <module>
    print(my_sum)
TypeError: 'int' object is not callable

Process finished with exit code 1
```

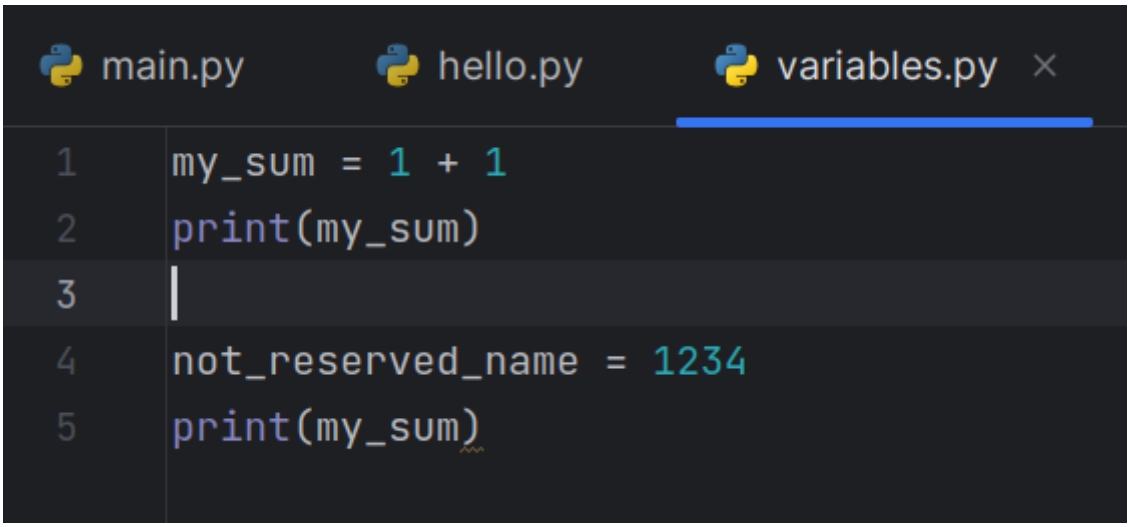
On the first line of our output, we have two paths. The first is the path to a particular version of Python that is being used specifically for our `first-pycharm-project.` The second path is the file that was run with this version of Python: the `variables.py` file that we have just created.

After this, we see the number 2. This is the output created by the second line in our code. Python was able to get to this point in the code without any problems, so we know the issue isn't coming from therem.

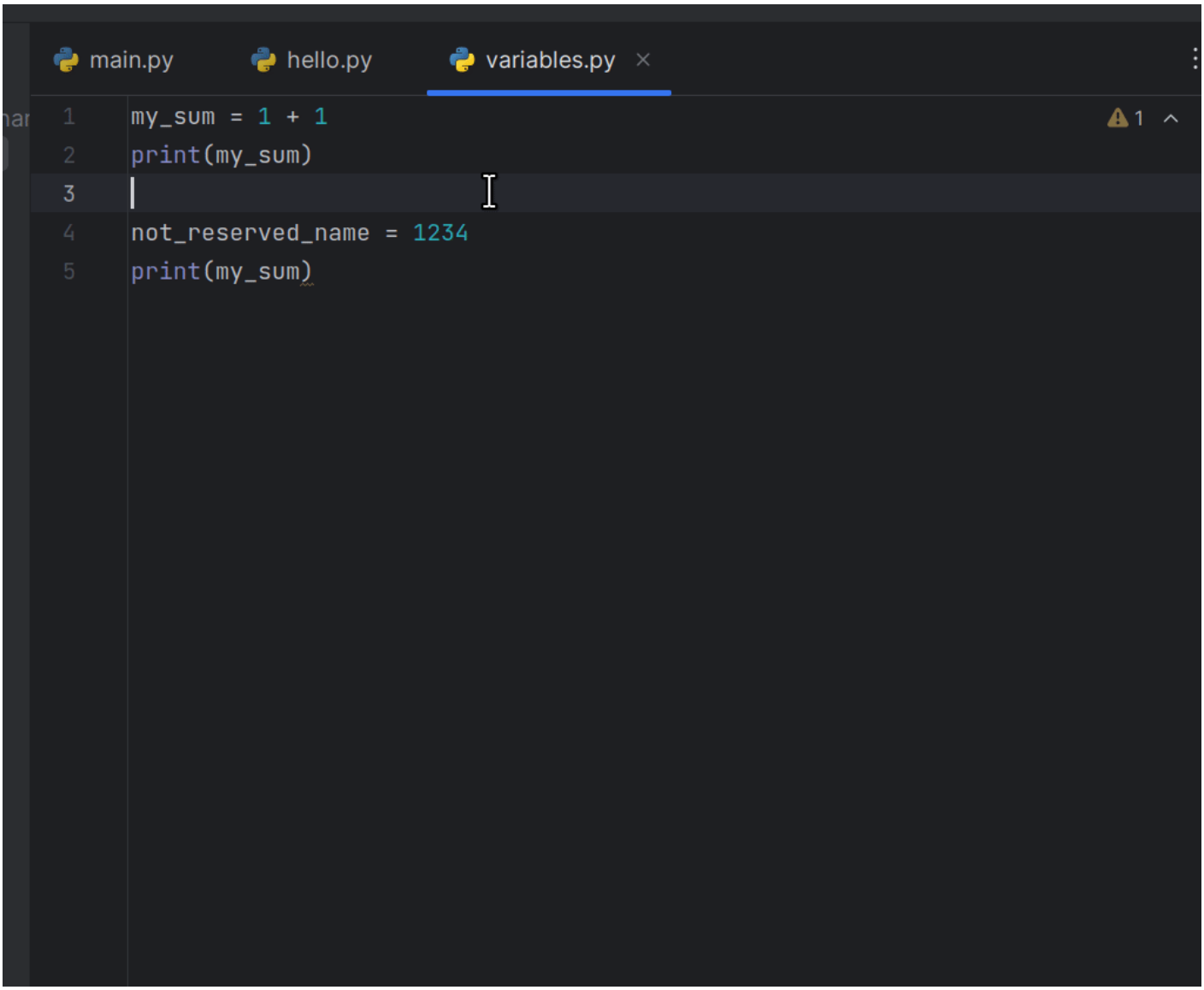
However, further down we get an error. The output tells us that on line 5 we attempted to treat an `int` as a Callable when it is not one. It even shows us that the command on that line we used was `print(my_sum)`. This is what we were warned about earlier in the mouse-over message.

Remember: When you deal with errors the first thing you should do is identify the file and line that it is coming from.

We can fix this by using a different variable name for our other number. Here, I have changed the name of the variable on line 4 to `not_reserved_name`.



Now the squiggly lines have disappeared. Another indication that the issue has been fixed is that the second `print` command now has the same highlighting as the first one. Now let's examine the mouse-ver messages again.



The first mouse-over message simply tells us we have a variable of the type `int` with a value of 1234. This is not a warning - it's simply telling us that a variable has been created on this line.

The mouse-over message shows us some information about how to use the `print` command. This is another handy feature of IDEs: showing us a bit about how commands work without having to go to its declaration or the documentation. Chances are you won't use any of the "advanced" features of `print` and will simply stick to using it with basic data.

Now you should understand why reserved words should *not* be used for naming variables.

Reusing Variables

Variables can hold different values over time. If you later find out you need 12 ingredients for the cake, you can simply change the value:

```
ingredient_count = 12
```

No need to create a new variable; the old one updates with the new value.

Dynamic Nature of Variables

Unlike some other programming languages, Python is dynamically typed. This means you don't have to specify the type of data a variable will hold. Python figures it out on its own.

Revision #11
Created 18 August 2023 11:16:43 by Dolica Akello-Egwel
Updated 22 September 2023 13:29:36 by Dolica Akello-Egwel