Python for Beginners

Introduction to Python

Welcome to the world of programming with Python! In this chapter, we will embark on a journey to explore the fundamentals of Python, one of the most versatile and popular programming languages in the world. Whether you're a complete beginner or have some programming experience, this course is designed to give you a strong foundation in Python and help you become a confident programmer.

What is Python?

Python is a high-level, interpreted programming language known for its readability, simplicity, and extensive standard library. Created by Guido van Rossum and first released in 1991, Python was designed to emphasize code readability and ease of use, making it an excellent choice for both beginners and experienced developers.

Why Python?

Python's popularity has surged over the years due to its wide range of applications. It's used in web development, data analysis, artificial intelligence, automation, and more. The language's clear syntax and strong community support make it an ideal starting point for beginner programmers.

Course Goals and Structure

The primary goal of this course is to equip you with the essential skills to write Python code and take advantage of the features provided by an Intergrated Development Environment (IDE). By the end of this course, you will be able to create your own programs and understand the core concepts of programming. Each chapter will build upon the previous one, gradually introducing new concepts and techniques.

Prerequisites

No prior programming experience is required to take this course. All you need is enthusiasm and a willingness to learn. If you have some experience with programming, you'll find that Python's syntax and concepts are relatively straightforward to grasp.

Getting Started

To get started, we'll need to set up your programming environment. We'll cover how to install Python on different operating systems and introduce you to the PyCharm Integrated Development Environment (IDE), which will make your coding experience smoother and more efficient.

Whether your goal is to use Python in your projects, enhance your analytical skills, or simply have fun with coding, learning Python is a valuable and rewarding endeavor. So, let's dive in and begin our exciting journey into the world of Python programming!

Python and Art

Setting Up Your Environment

Installing Python

Mac

Windows

Installing Mamba

As you work on more complex Python projects, you'll start using something called **libraries** to add extra features to your programs. Think of libraries as ready-made sets of code created and maintained by other developers. They're like toolkits that solve specific problems or add new features to Python. Instead of reinventing the wheel, it's much smarter to use these libraries when they fit your needs. The range and number of Python libaries is considered to be one of the best selling-points of the language. However, you may find that sometimes you'll have different projects that need different versions of the same library. This is where virtual environments come in handy. They let you set up separate "sandboxed" Python configurations that don't mess with each other.

In this tutorial, we're going to use a piece of software called Mamba to create and manage virutal environments. Mamba is a package manager and environment manager for Python. It is built on the Conda package management system, and its primary goal is to provide quicker and more efficient operations for managing packages and creating isolated environments.

Mac

Windows

Installing PyCharm

What is an IDE and Why Use One?

A programming Integrated Development Environment (IDE) is a software application that provides a comprehensive set of tools and features to assist you in your programming. It serves as a central platform for programmers to write, edit, test, debug, and manage their code more efficiently and effectively. An IDE typically combines various components, such as a code editor, debugger, compiler, and other development tools, into a single cohesive environment.

Here are some key features and components commonly found in programming IDEs:

1. **Code Editor**: The core component of an IDE, the code editor offers features like syntax highlighting, code auto-completion, indentation, and formatting to enhance code readability and writing speed.
2. **Debugger**: IDEs often include a debugger that helps programmers identify and rectify errors in their code by allowing step-by-step execution, variable inspection, and setting breakpoints to pause the program's execution for analysis.
3. **Compiler/Interpreter Integration**: An IDE can be configured to work with specific compilers or interpreters for various programming languages, enabling code compilation and execution directly from the IDE.
4. **Version Control Integration**: You'll soon find that version control software will become an indispensable part of managing your code. Many IDEs integrate with version control systems like Git, making it easier to manage code changes, collaborate with team members, and track project history.
5. **Error Highlighting**: Instant feedback on code errors, warnings, and potential issues is a common feature of IDEs. This helps catch mistakes as you write code.
6. **Plug-ins and Extensions**: Many IDEs support plug-ins or extensions that allow users to customise the IDE's functionality, adding features and support for additional languages or frameworks.

IDEs play a crucial role in modern software development by providing a cohesive environment that streamlines the coding process, encourages best practices, and enhances productivity.

PyCharm is a popular free IDE for Python development that has all of the features listed above. Follow the steps below in order to get it up and running on your system.
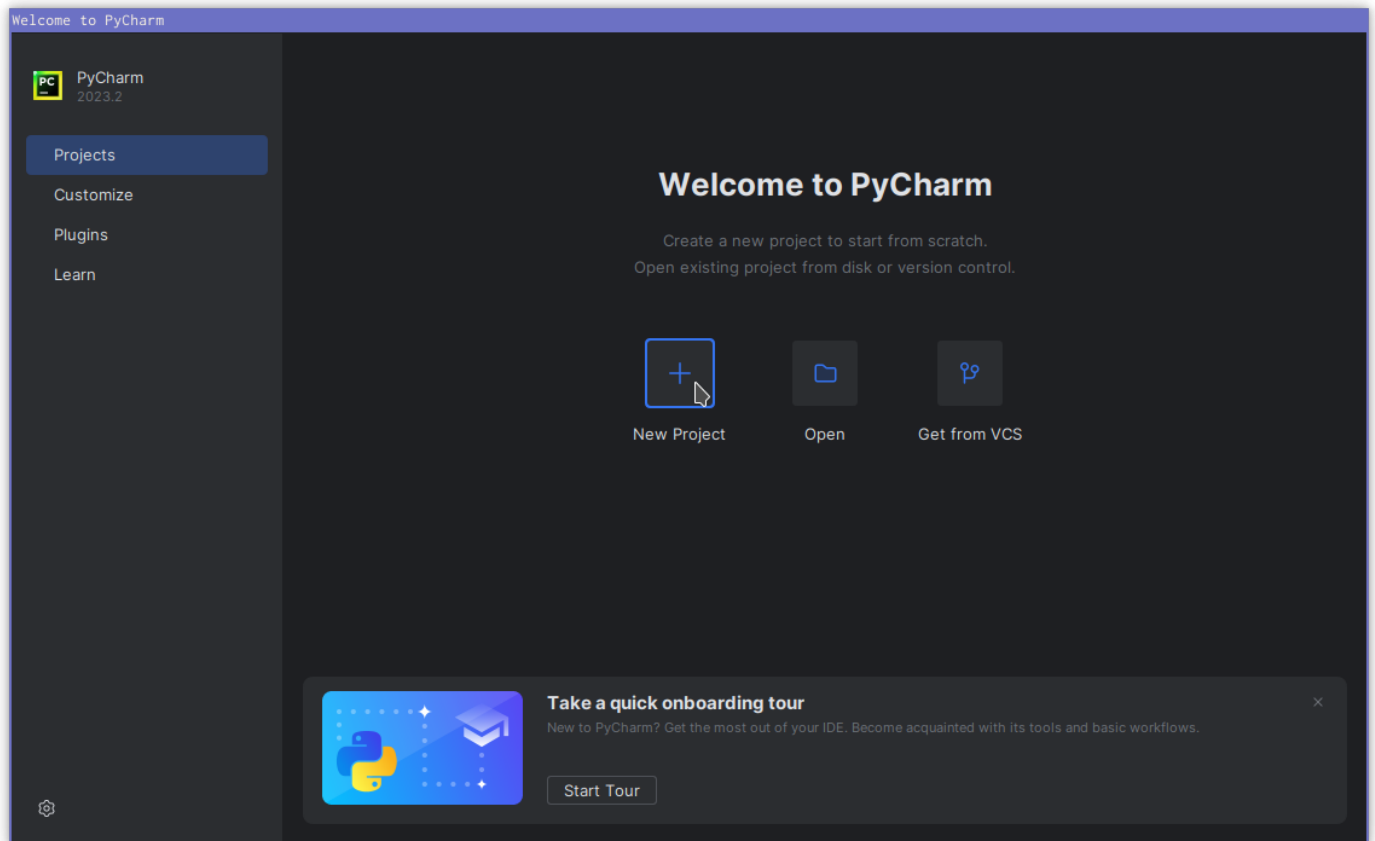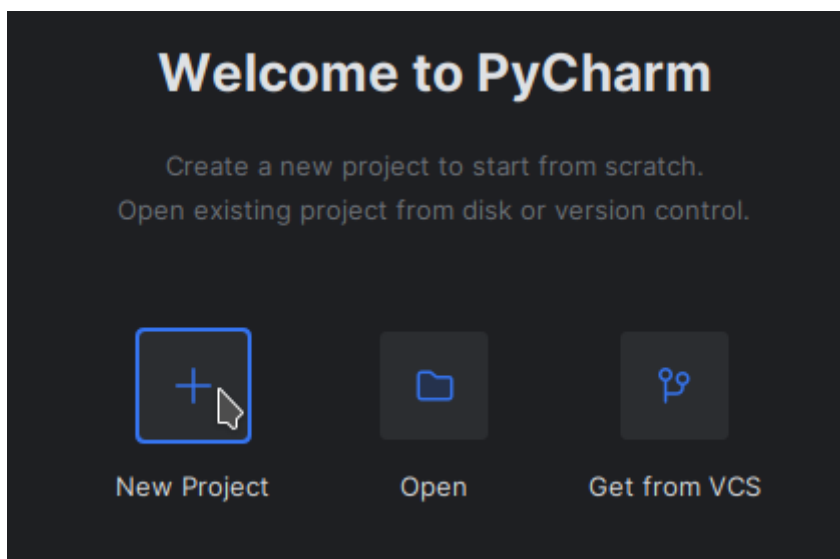
Mac

Windows

Introduction to the PyCharm IDE

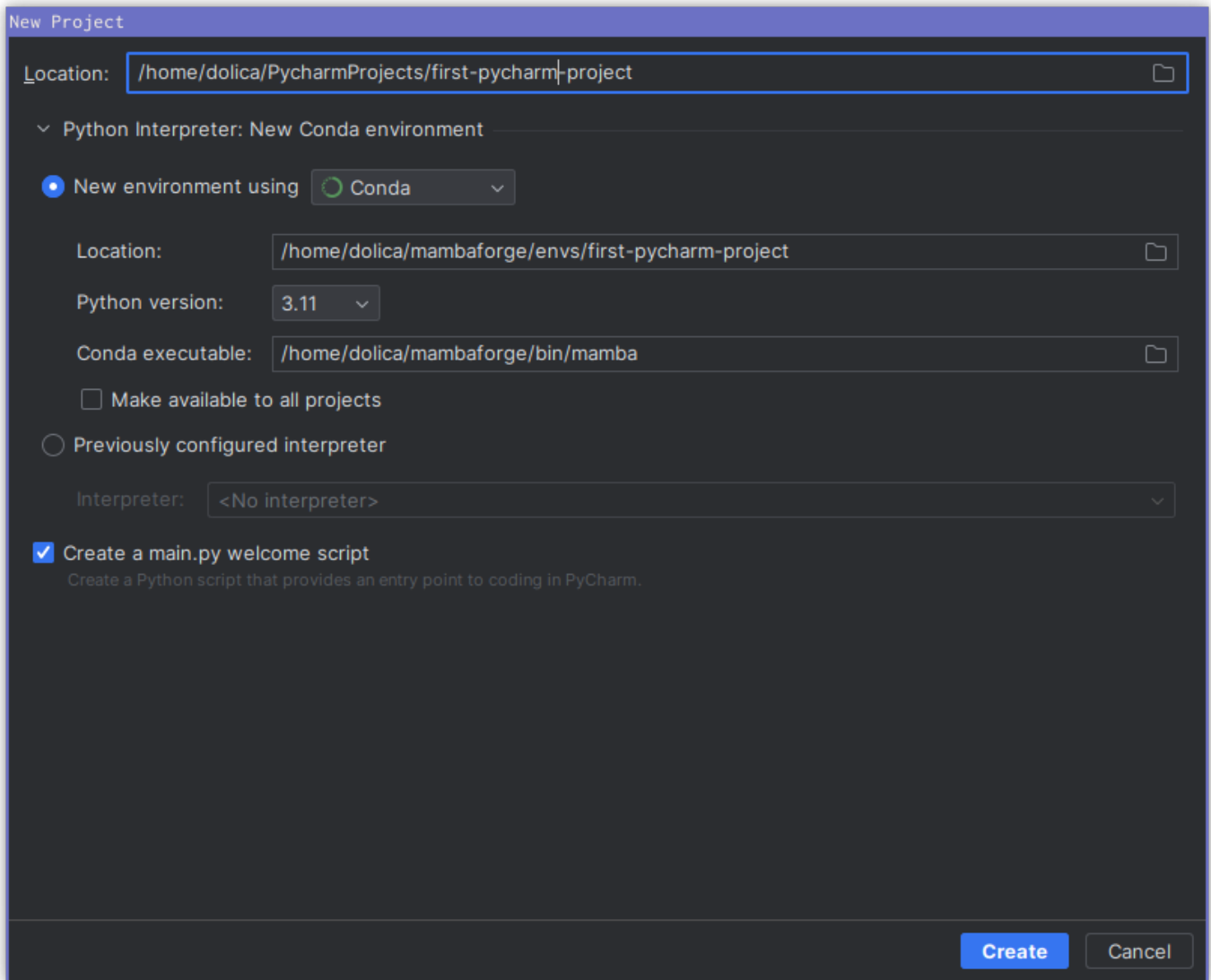# Creating Your First Project in PyCharm

When you first use PyCharm, you'll be welcomed by its startup screen.

To create a blank Python project, click on "New project."

PyCharm will now prompt you to configure the Python project:

In this example, the project files will reside in my `/home/dolica/PyCharmProjects/first-pycharm-project` directory. However, you can choose a location that suits your needs.

For the virtual environment, select Conda. PyCharm automatically makes the virtual environment's name match the project name, and this is something we wish to keep as it makes things less confusing. The setup page also asks if you want to create a `main.py` welcome script; for this project, you should keep this option checked. This will have PyCharm create a basic Python file for you instead of just preparing a blank directory.

Once the setup is complete, click "Create."

The PyCharm Interface

After creating your project, you'll land in the main PyCharm interface.
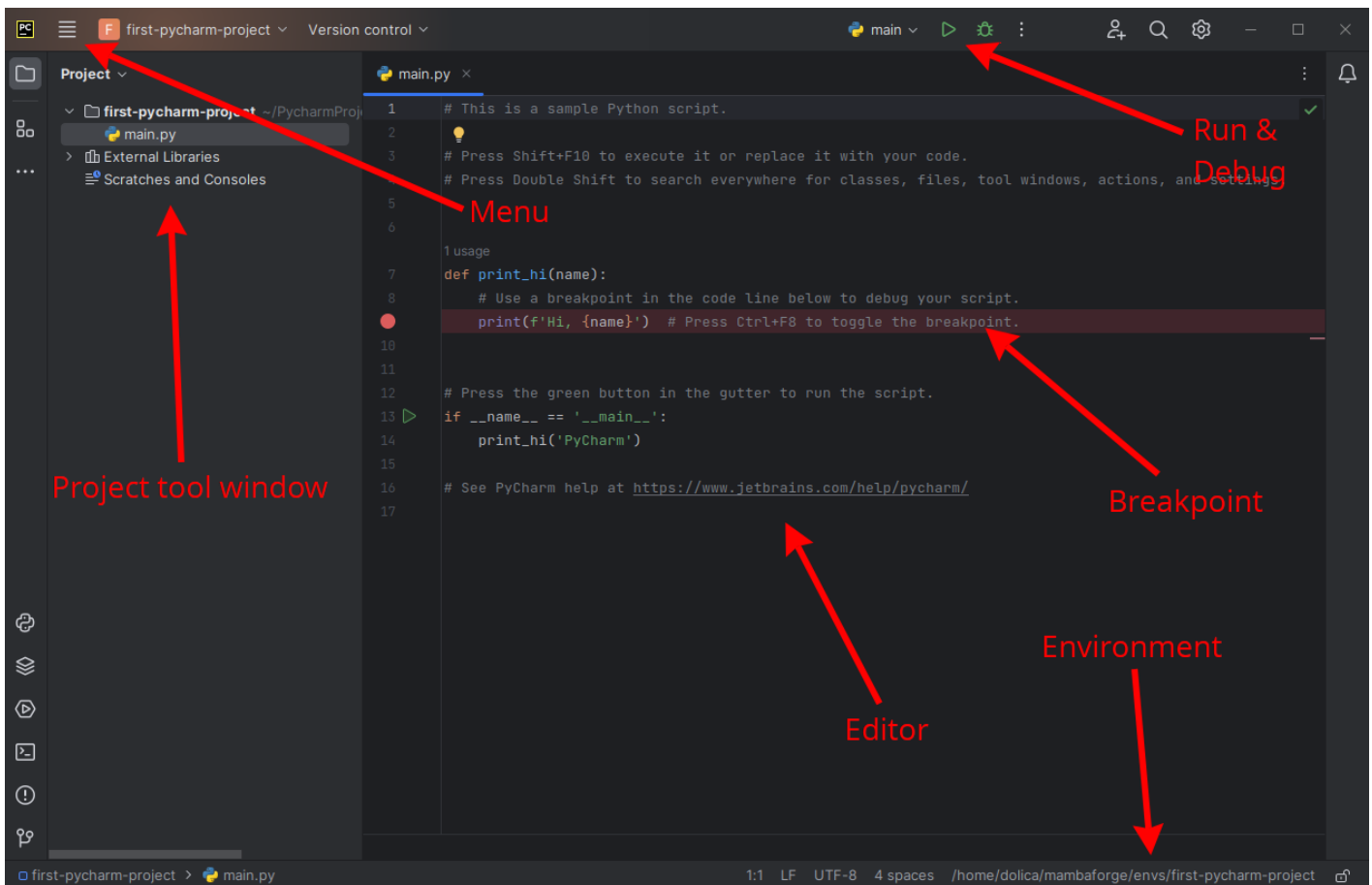
Here's a quick overview:

- **Project Tool Window:** This displays the files comprising your Python project. Right now, there's only one file, `main.py`, but more complex projects will have multiple files and folders.
- **Menu:** Clicking here opens the PyCharm menu.
- **Editor:** This is where you write code. The tab bar at the top lets you switch between different files, though currently, we have only one, `main.py`.
- **Environment:** This shows the environment used to run your Python files for this project, matching the Mamba settings you selected when creating the project.
- **Breakpoint:** Breakpoints are handy for pausing program execution, especially when debugging code to find issues.
- **Run & Debug:** The run button (play symbol) runs your code and displays the output in the console. The debug button (bug symbol) runs the program in debugging mode, pausing at breakpoints.

Hi, PyCharm

Now you should see the PyCharm editor with our `main.py` file open. You'll notice a `print_hi` command created by PyCharm. You'll learn more about how these commands work and how to create your own later. For now, let's run this file:
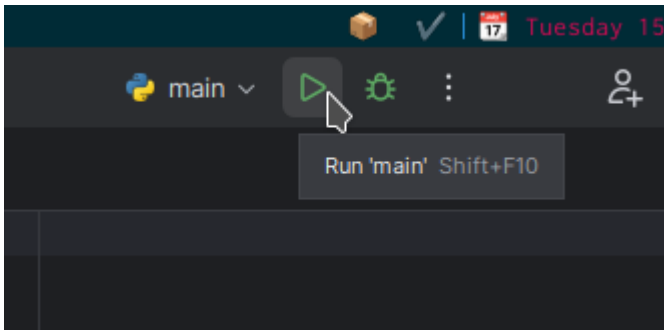
This opens the "Run" panel at the bottom of the window, displaying the output:
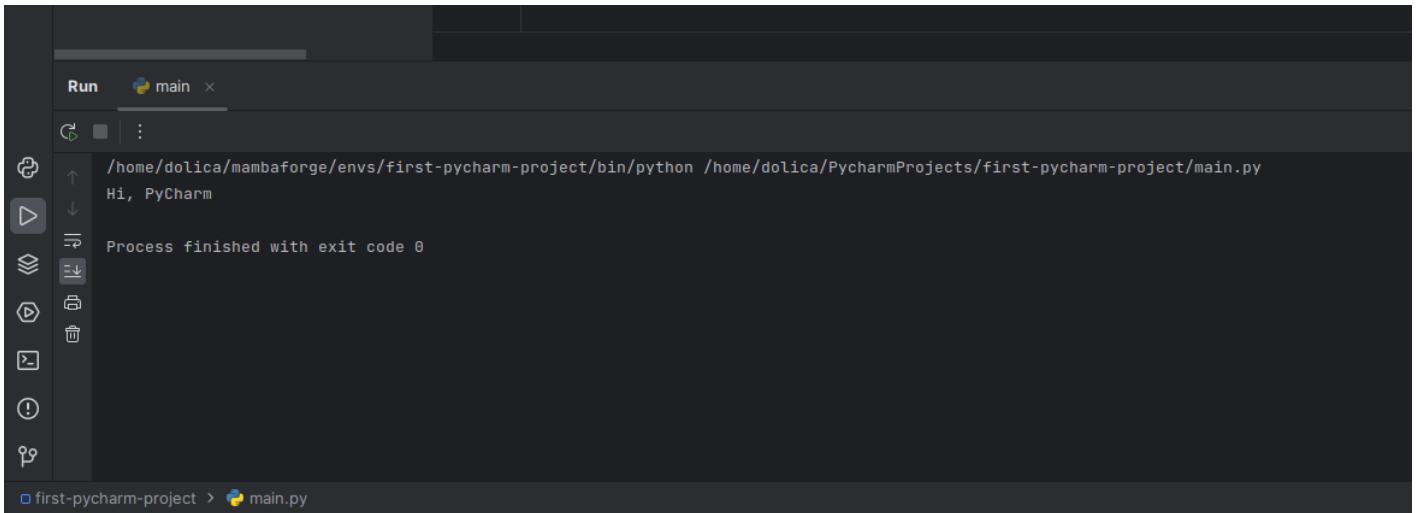

The text "Process finished with exit code 0" indicates that the program ran without errors. Our `main.py` code is designed to display "Hi, PyCharm," and it has executed correctly.

Hello World

Creating a "Hello, World" Program in PyCharm

Now that we have Python, Mamba, and PyCharm set up, let's create a simple "Hello, World" program. To do this, we will create a new Python file using the following steps:

1. Right-click on your `first-pycharm-project` folder.
2. Go to "New" and select "Python File."
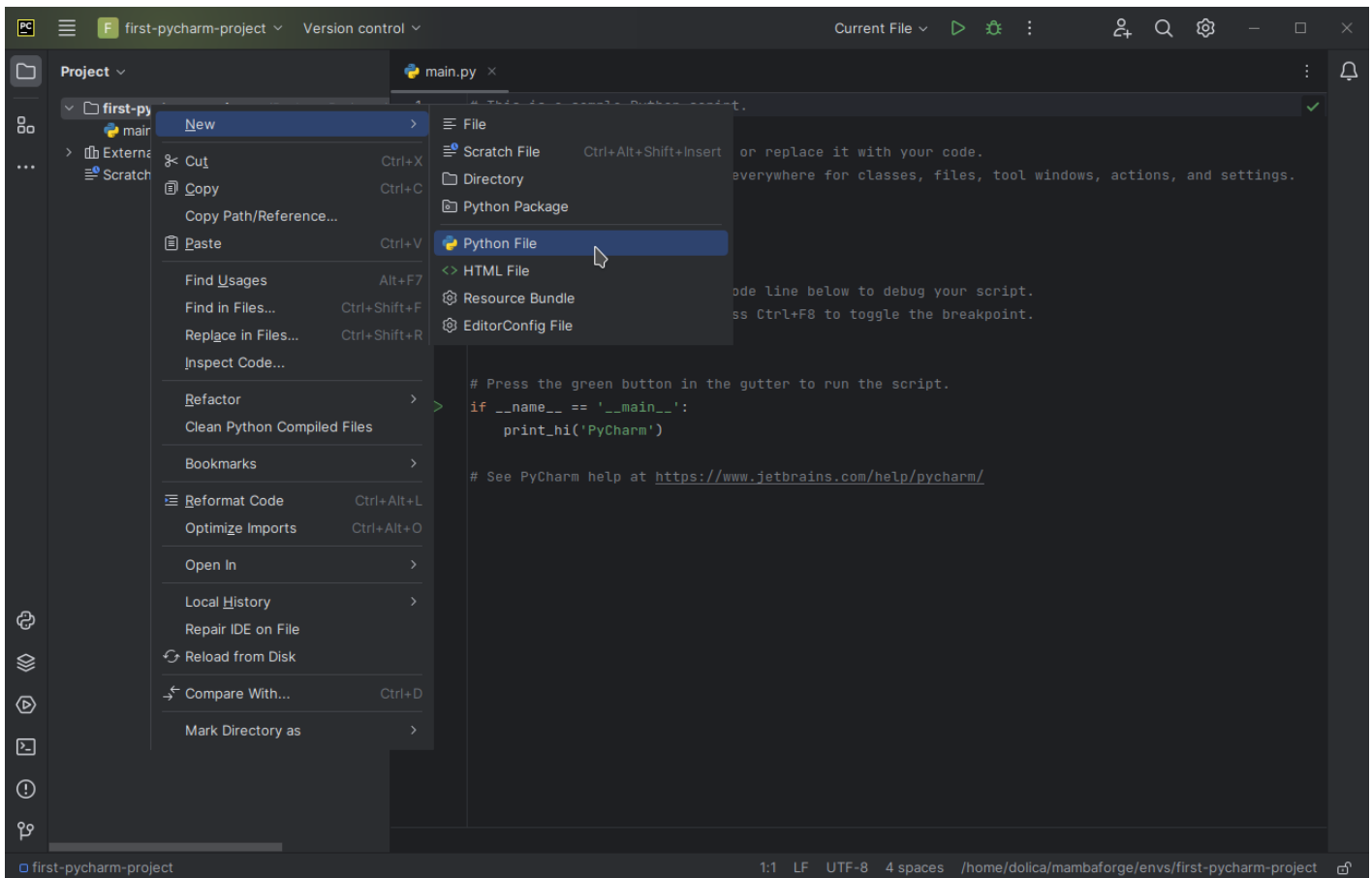
You'll now be prompted to choose a file name and select the "type" of Python file you want PyCharm to create. For this example, let's name the file `hello.py` and choose the file type as "Python file." While this might seem redundant, you'll see that there's a reason PyCharm offers the other file types.
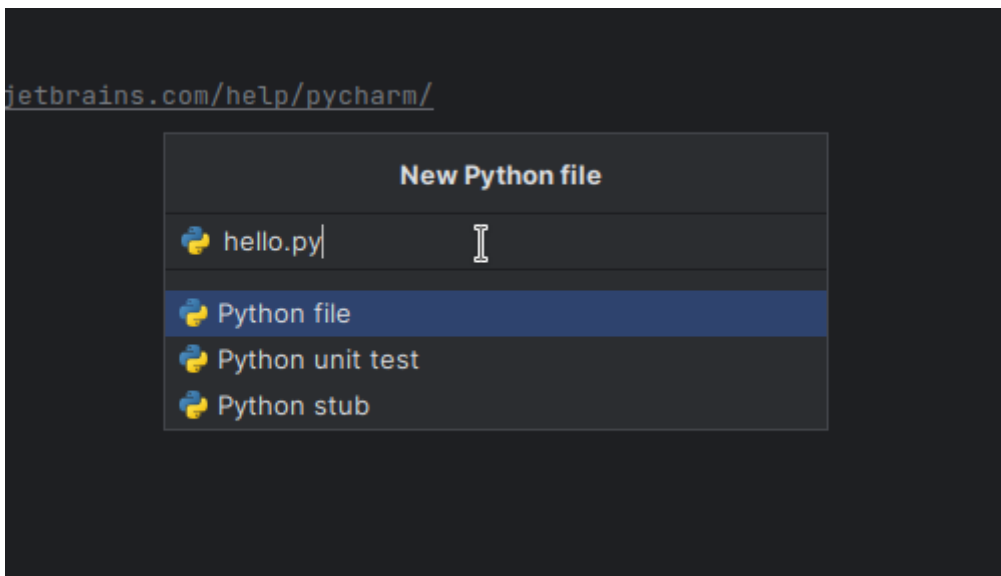
Press Enter, and a new Python file named `hello.py` will be created. You can verify its creation by checking the files listed in the project tool window. The empty `hello.py` file will also open in the editor.
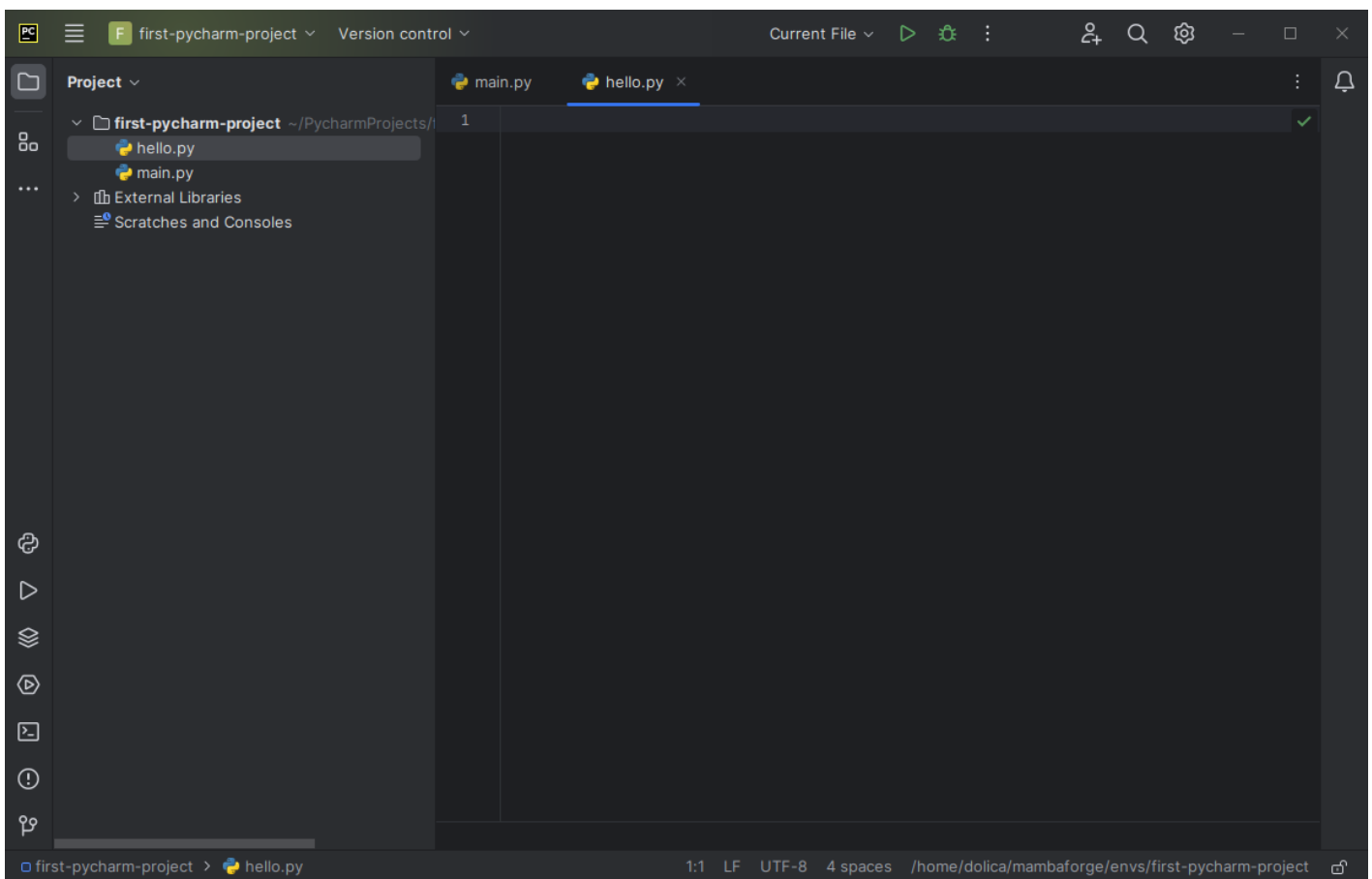
ust like before, you can run the newly created Python file. However, since there's no code in it yet, you won't see any meaningful output. It will simply indicate that the program finished with exit code 0, as we'd expect no errors to occurr seeing as we haven't even writen any code yet.
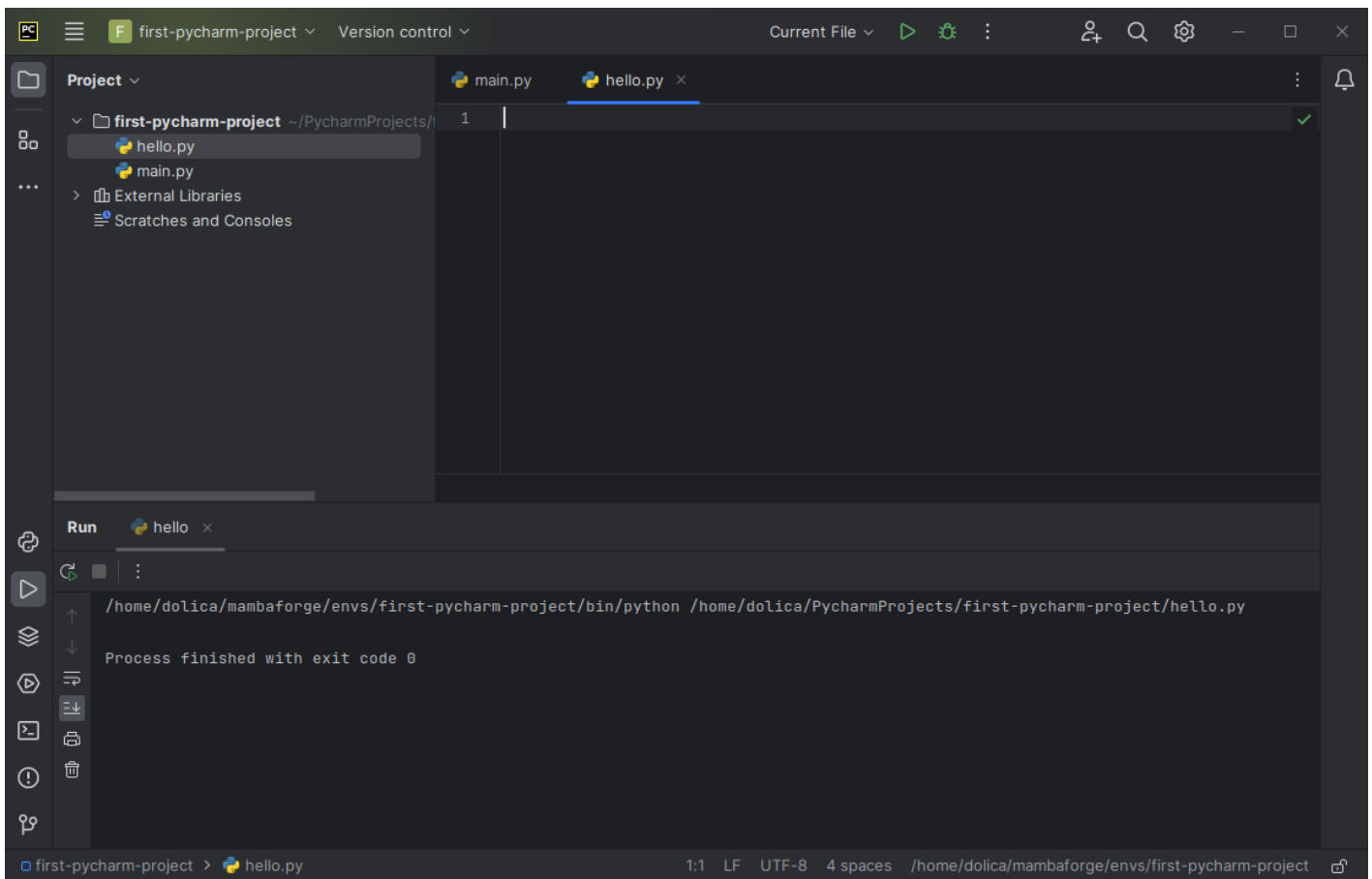
To create a "Hello, World" program, we'll use a `print()` statement. In Python, `print()` is a command that sends output to the console. Inside the brackets of `print()`, we provide the text we want Python to print. This text must be enclosed in quotation marks so that Python knows where it begins and ends. Here's what the code looks like:

```
print("Hello World")
```

If you type out the above code letter-by-letter, you'll notice that PyCharm's **auto-complete** feature can assist you. As you start typing "pri," PyCharm will suggest commands that contain that text. When "print" is selected you can then press Enter and PyCharm will automatically add the remaining characters, including the parentheses, and place your cursor between the brackets, where you can type what it is you want to print.

In this example, I waited until I had typed out "prin" before employing auto-complete's help, but I could have done it sooner. As you type out more, you narrow down PyCharm's list of potential ways for completing your code.
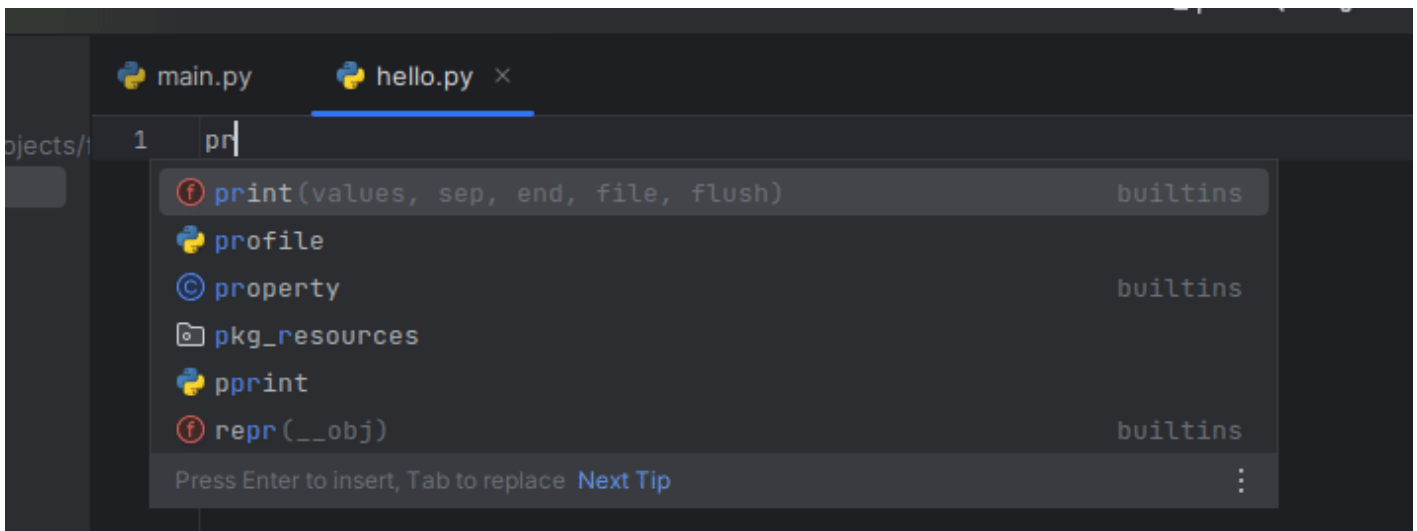
Now let's see what happens when I use auto-complete after just typing the letter "p."
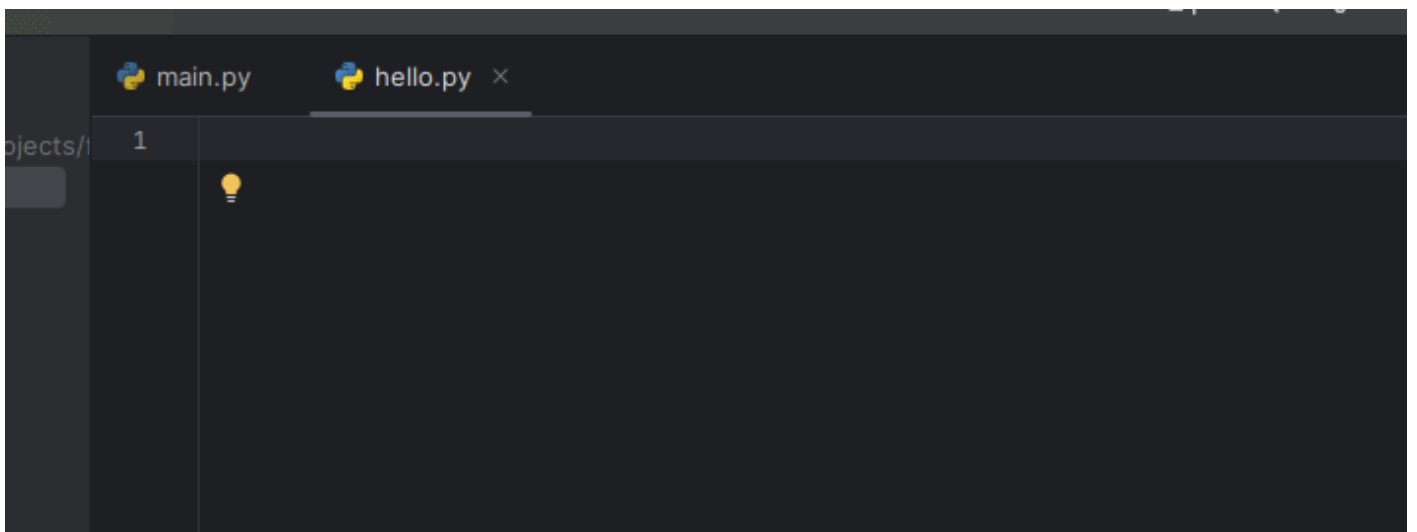
Now I've saved myself a fair few keystrokes as PyCharm rightly guesses that I'm typing out "p" because I wish to use the print command.

Additionally, for writing the text, PyCharm will automatically create a closing quotation marks when you type an opening one, saving you time. This is also a feature frequently found in IDEs.

With the `print("Hello World")` code in place, you can run the program again. This time, you'll see meaningful output.
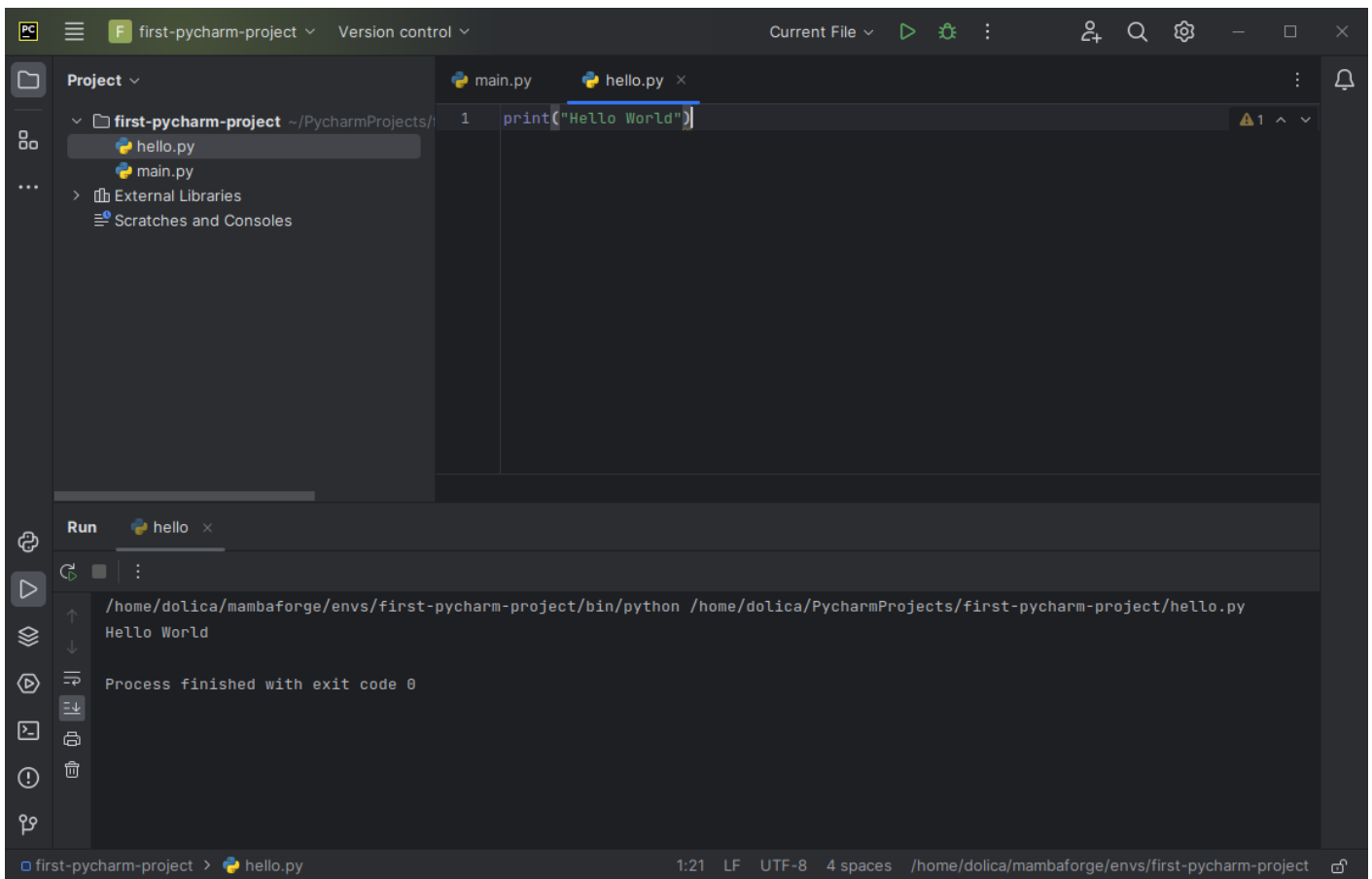
Now you have successfully created and run your "Hello World" program in PyCharm. This is a simple yet essential step in getting started with Python programming. Now let's look at variables.

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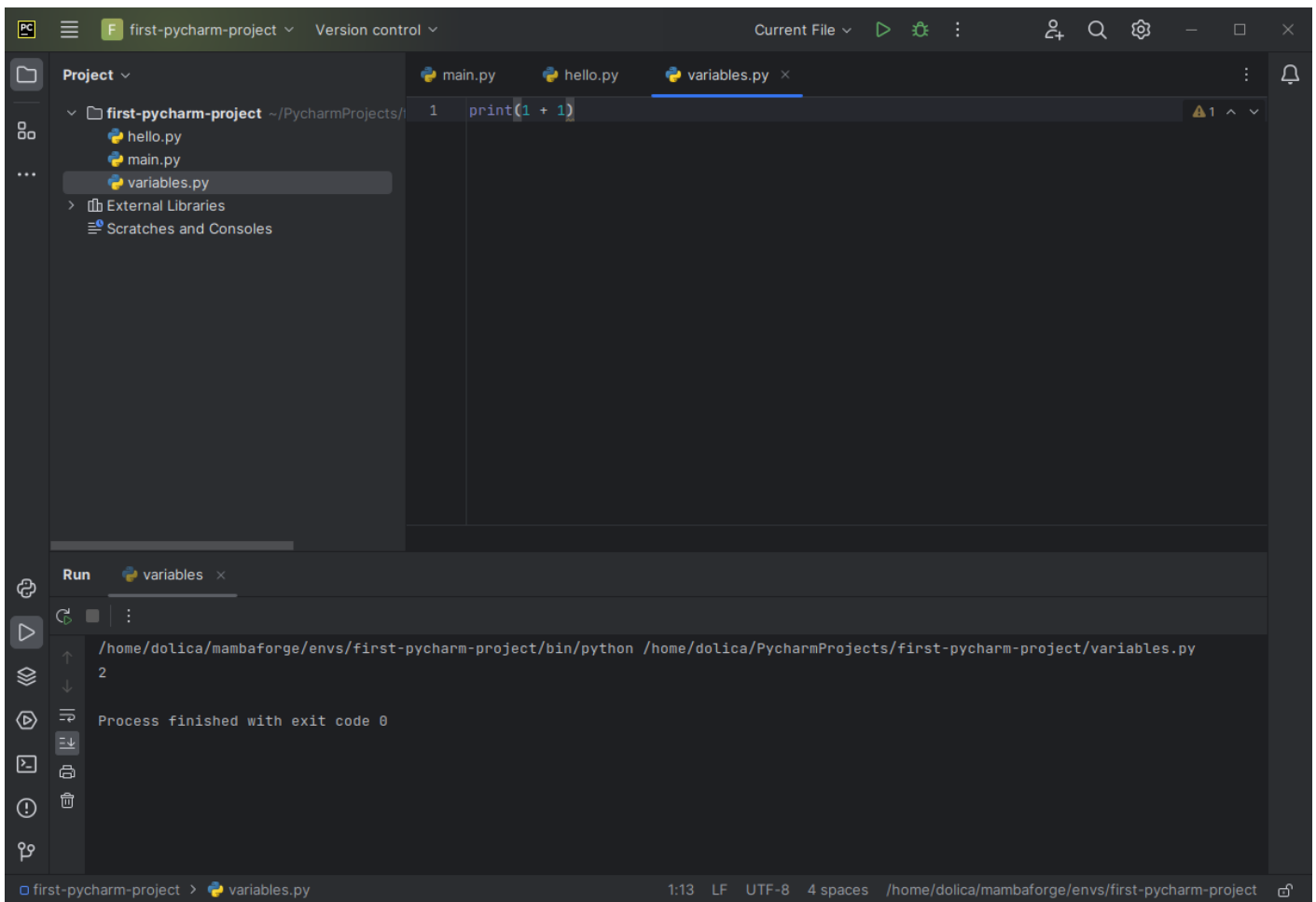
However, you may want to use a certain value more than once. It is often useful to store data in our code for repeated use. This is where the concept of **variables** comes into play.

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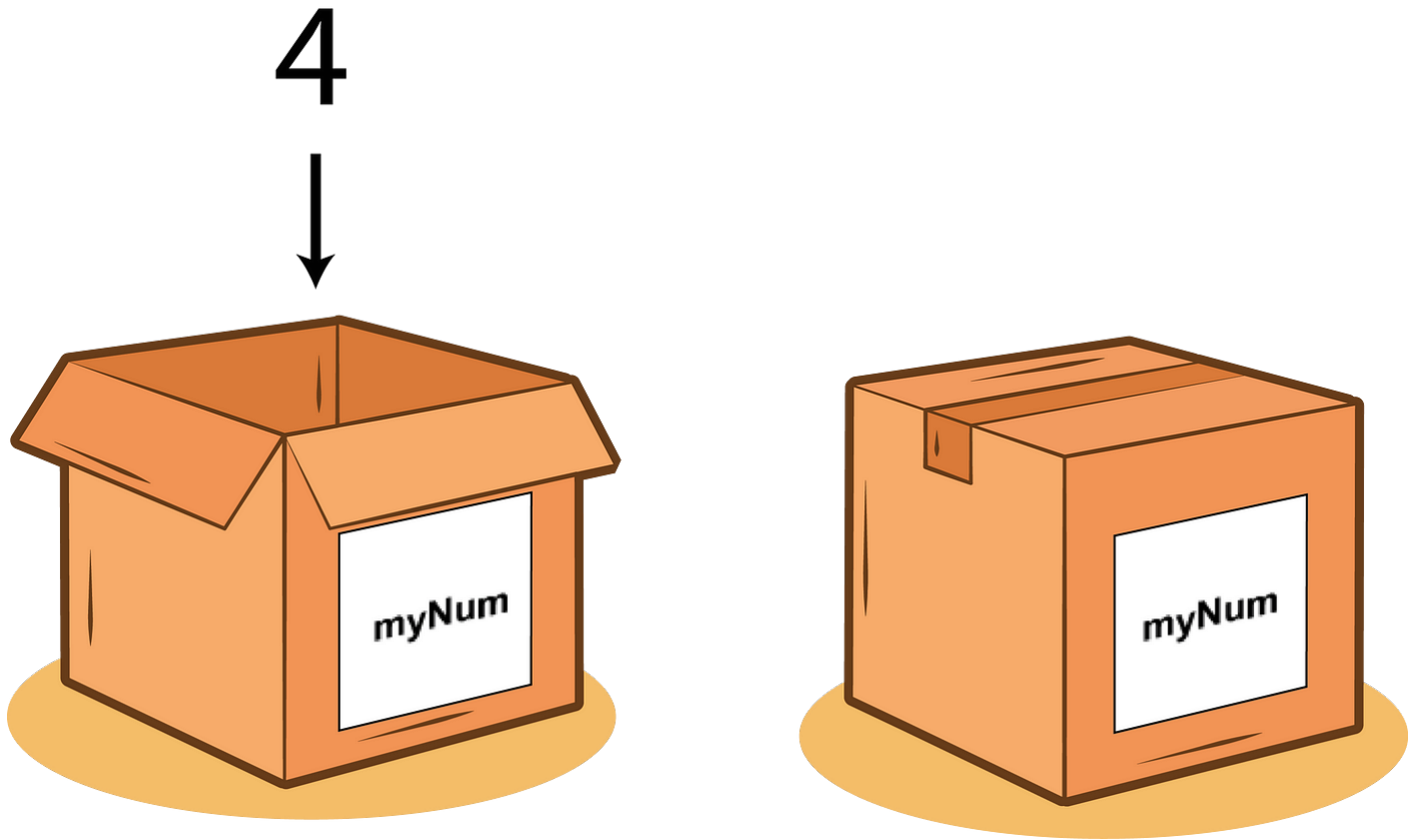
Variables allow you to give a name to a piece of data, making your code more readable and organised.

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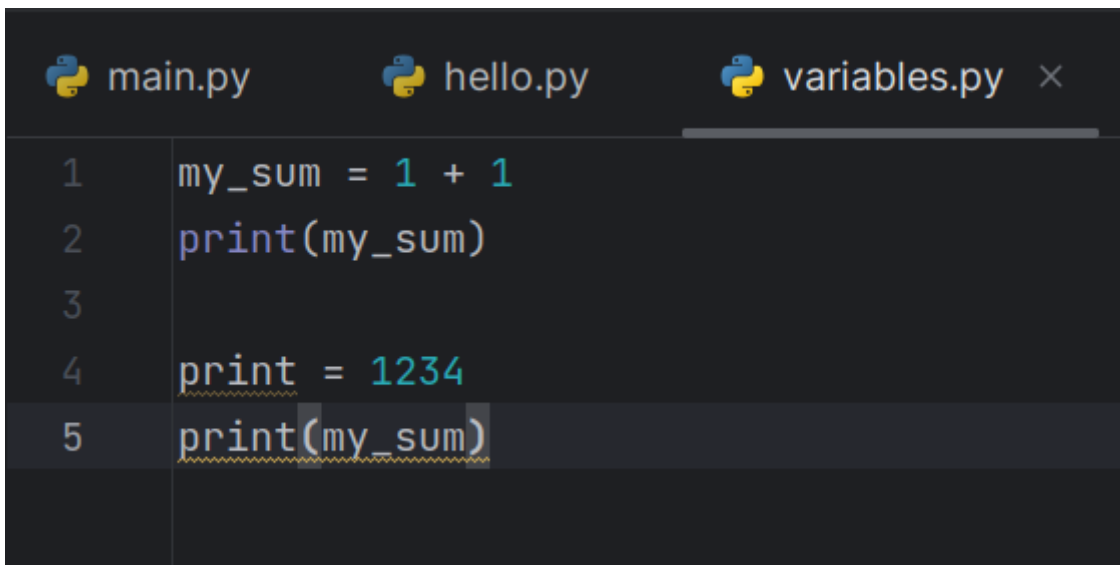
Both code snippets accomplish the same task, but the second one requires less mental effort to understand its purpose. This is an important consideration when writing code and selecting variable names.

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:
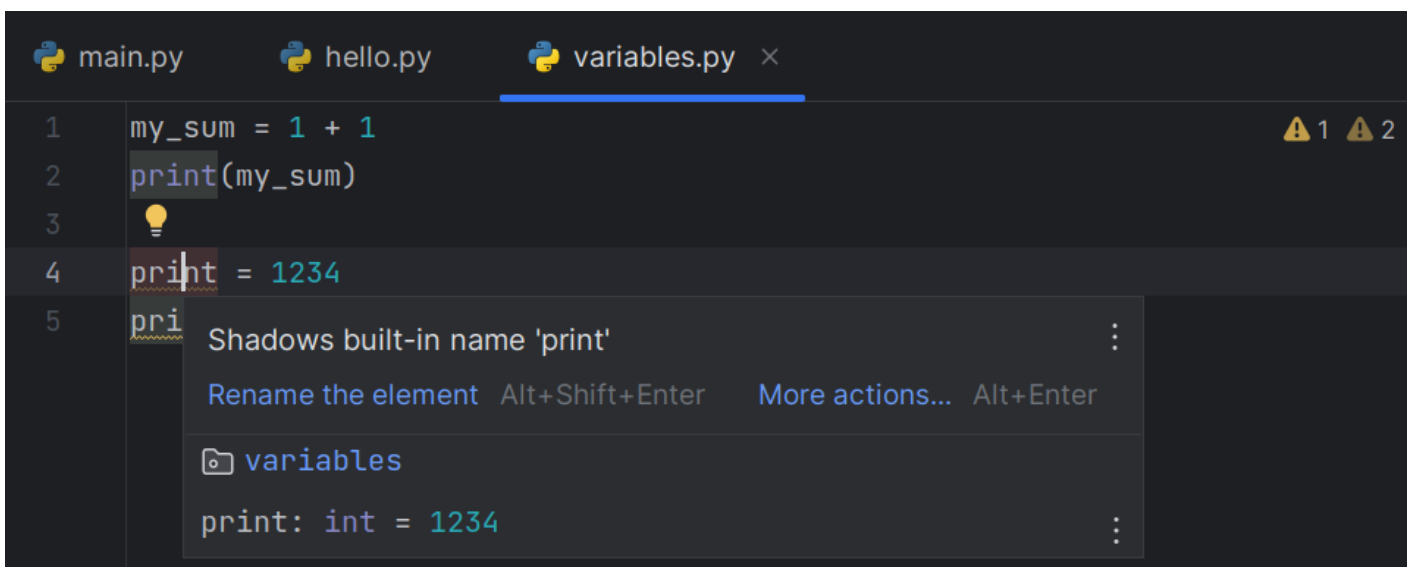
```
  main.py        hello.py        variables.py   ✕

1      my_sum = 1 + 1
2      print(my_sum)
3
4      print = 1234
5      print(my_sum)
```

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:
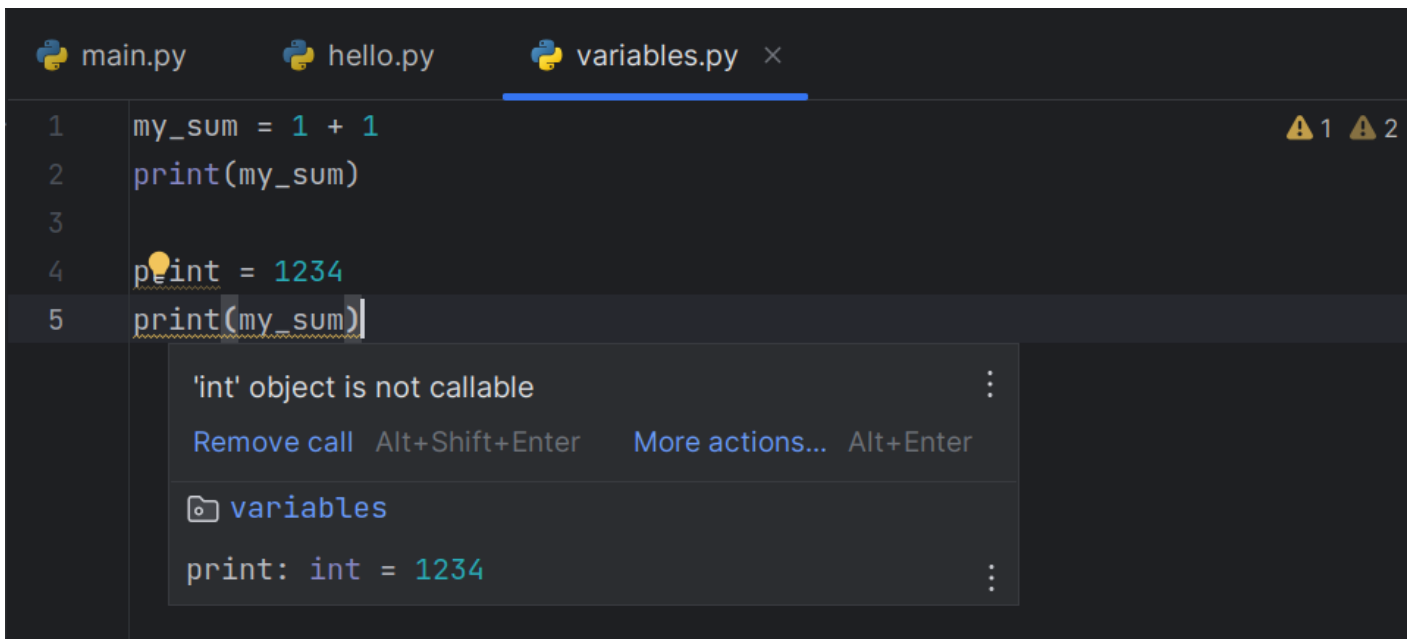
```
  main.py        hello.py        variables.py   ✕

1      my_sum = 1 + 1                                                    ⚠1  ⚠2
2      print(my_sum)
3      💡
4      print = 1234
5      pri
         Shadows built-in name 'print'                              ⋮
            Rename the element  Alt+Shift+Enter     More actions... Alt+Enter

            ⊙ variables
            print: int = 1234                                       ⋮
```

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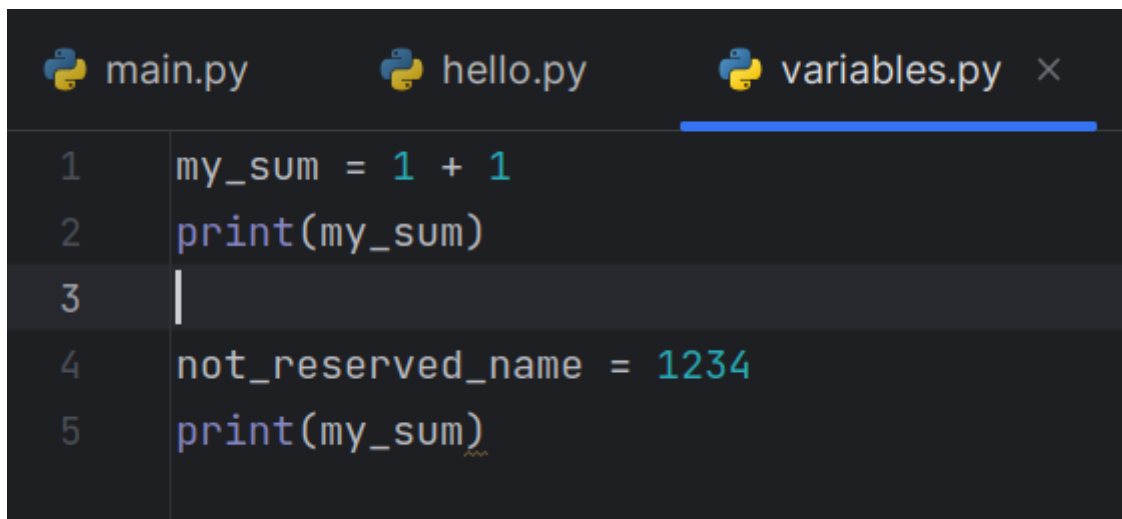
On the first line of our output, we have two paths. The first is the path to a particular version of Python that is being used specifically for our `first-pycharm-project.` The second path is the file that was run with this version of Python: the `variables.py` file that we have just created.

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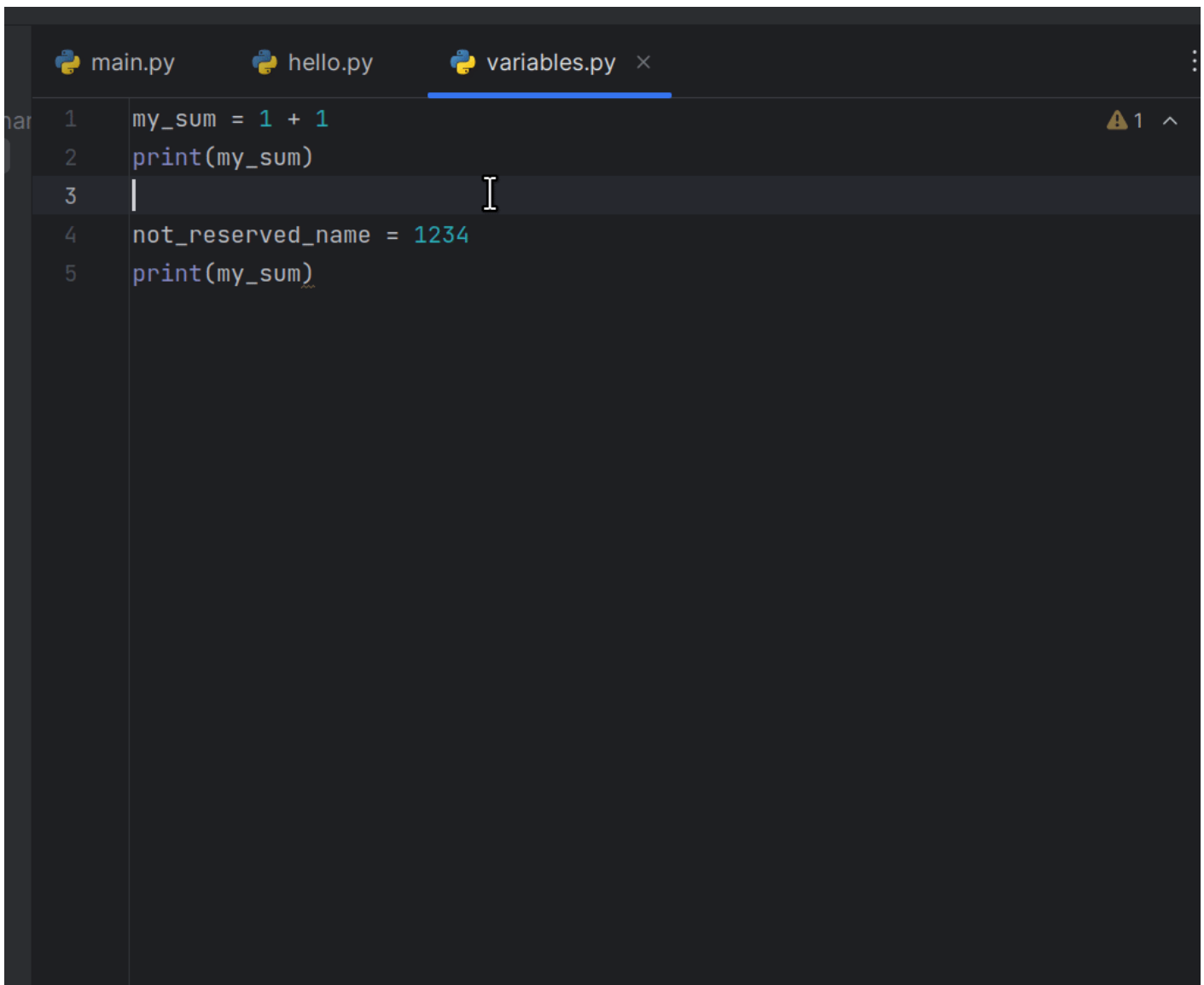
Remember: When you deal with errors the first thing you should do is identify the file and line that it is coming from.

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.

```python
main.py          hello.py          variables.py  ✕                                      ⋮

1    my_sum = 1 + 1                                                          ⚠ 1  ∧
2    print(my_sum)
3    |
4    not_reserved_name = 1234
5    print(my_sum)
```

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:

```python
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.

Basic Input and Output

Control Flow and Conditional Statements

Collections

Working with Files

Bonus: Introduction to Object Oriented Programming

Bonus: Using the Debugger

Bonus: Using Git from PyCharm

Best Practices and Further Learning