Setting Up Python and PyCharm

A Beginner's Guide to Running Your First "Hello World" Program

Updated August, 2026

Contents

1 Python Setup

We start by installing the Python interpreter. As of this writing, the latest stable release is **Python 3.14.7**, and the officially supported installers cover Windows 10 and Windows 11 (Windows 7 support ended with the Python 3.8 series, so older systems can no longer use current releases).

Note

Always download Python from the official source to avoid modified or malware-laced installers: `https://www.python.org/downloads/`. The site automatically detects your operating system and offers the recommended installer.

Link: `https://www.python.org/downloads/`
Select the **Windows installer (64-bit)** for Python 3.14.7. A lightweight alternative is the new

Python Install Manager (MSIX-based), which JetBrains and python.org both recommend for Windows because it can keep your Python installation updated automatically; it is also available from the same downloads page.

1. Run the downloaded installer.
2. **Tick the "Add python.exe to PATH" box** (this is the modern label for the box previously called "Add Python to PATH").
3. Click **Install Now** to start the installation with default settings.

Figure 1: The Python 3.14.7 setup window – tick *Add python.exe to PATH* before clicking *Install Now*.

1.1 Installing PyCharm

After installing the Python interpreter, we need an Integrated Development Environment (IDE) to write and run our code. We will use **JetBrains PyCharm**, currently at version **2026.2.1**.

Link: `https://www.jetbrains.com/pycharm/download/`

Note

JetBrains moved to a **unified licensing model**: the core PyCharm IDE (what used to be the "Community Edition") is now free forever for everyone, while a paid subscription unlocks Professional-only tooling such as full web-framework support, database tools, and remote interpreters. For this tutorial the free core edition is all you need.

Download the Windows installer and run it. During setup:

1. Tick **64-bit launcher** to create a desktop shortcut.
2. Tick **.py** to associate Python files with PyCharm.
3. Leave the bundled Java runtime option checked – PyCharm now ships its own OpenJDK, so you no longer need to install Java separately.
4. On the "Import Settings" screen, choose **Do not import settings**.
5. Skip the plugin selection screen (**Skip Remaining and Set Defaults**) – no extra plugins are required for this tutorial.

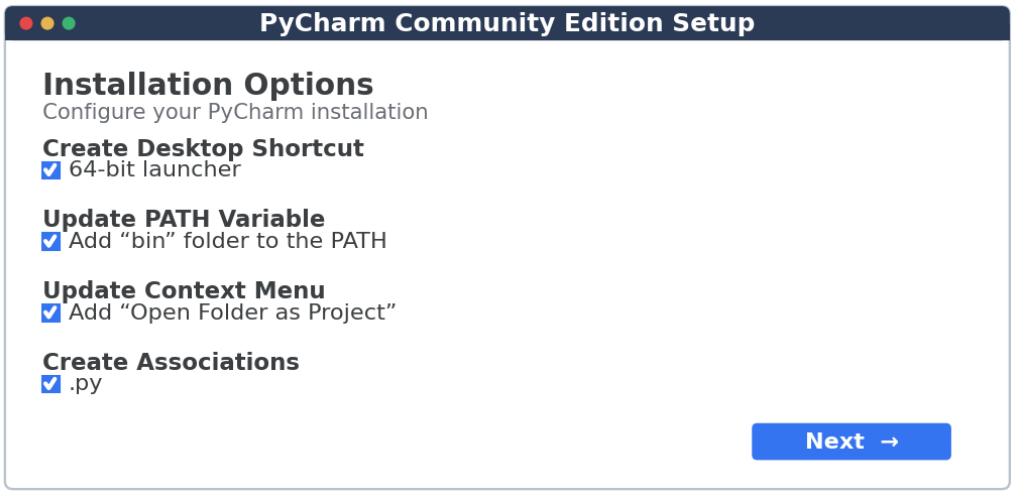

Figure 2: PyCharm installation options – tick the launcher, PATH, context-menu, and `.py` association boxes.

2 Required Packages

Next, we install the packages needed for a simple Machine Learning program. Modern PyCharm manages interpreters and packages through a per-project **virtual environment**, which keeps each project's dependencies isolated.

2.1 Creating and Configuring the Interpreter

1. Open PyCharm and, from the Welcome screen, click **Settings** (the gear icon) or press `Ctrl+Alt+S` once a project is open.
2. Navigate to **Project: <name> → Python Interpreter**.
3. Click **Add Interpreter → Add Local Interpreter**.
4. Choose **Virtualenv Environment**, select **New environment**, and make sure the **Base interpreter** points to the Python 3.14 installation you just downloaded.
5. Click **OK**. PyCharm creates an isolated virtual environment for the project, with `pip` already included.

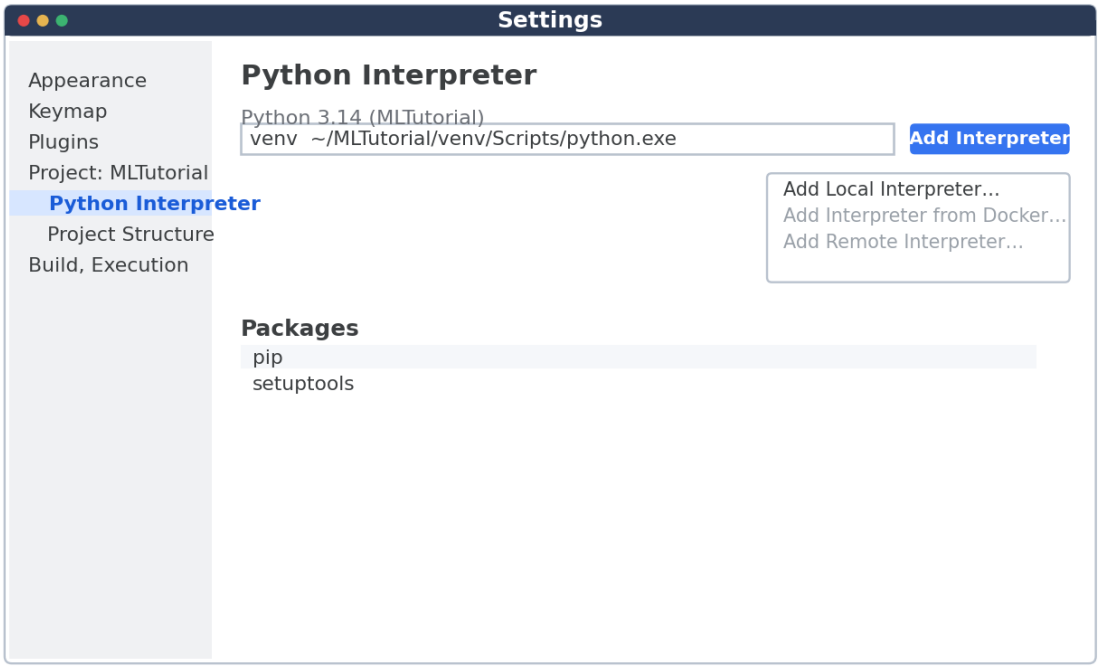

Figure 3: The Python Interpreter settings page, showing the *Add Interpreter* menu and the packages already installed in the virtual environment.

Note

PyCharm 2026 also has first-class support for **uv**, a much faster drop-in replacement for `pip` and `venv`. If `uv` is installed on your system, PyCharm will offer it as an environment manager option alongside Virtualenv – either choice works fine for this tutorial.

2.2 Installing NumPy, Pandas, SciPy, and Scikit-learn

With the interpreter configured, open the **Python Packages** tool window (bottom of the PyCharm window) or return to **Settings** → **Project** → **Python Interpreter**, where installed packages now appear in a collapsible, searchable tree.

For each of the four packages below, type its name in the search bar, select it from the results, and click **Install Package**. Wait for the green confirmation before moving to the next one.

- **numpy**
- **pandas**
- **scipy**
- **scikit-learn**

Figure 4: The Python Packages tool window – search for a package, select it, and click *Install Package*. A green checkmark confirms a successful install.

Alternatively, all four can be installed at once from PyCharm's integrated terminal (**View** → **Tool Windows** → **Terminal**):

```
pip install numpy pandas scipy scikit-learn
```

Once installation finishes, click **OK** to close Settings and return to the main window.

3 Running a Python Project

3.1 Creating the Project

1. From the Welcome screen, choose **New Project**.
2. In the **Name** field, enter a project name such as `MLTutorial`.
3. Confirm the **Python interpreter** shown is the Python 3.14 virtual environment created earlier.
4. Leave **Create a main.py welcome script** ticked if you would like a starter file, then click **Create**.

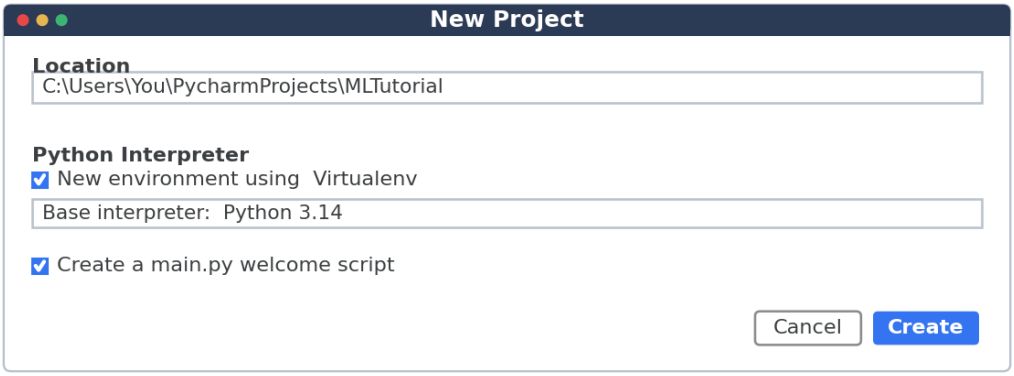

Figure 5: The New Project dialog – set the project location and confirm the virtual-environment interpreter before clicking *Create*.

3.2 Adding a Python File

1. Right-click the project folder in the **Project** panel on the left.
2. Select **New** → **Python File**.

3. Give the file a name, for example `hello_world`, and press **Enter**.

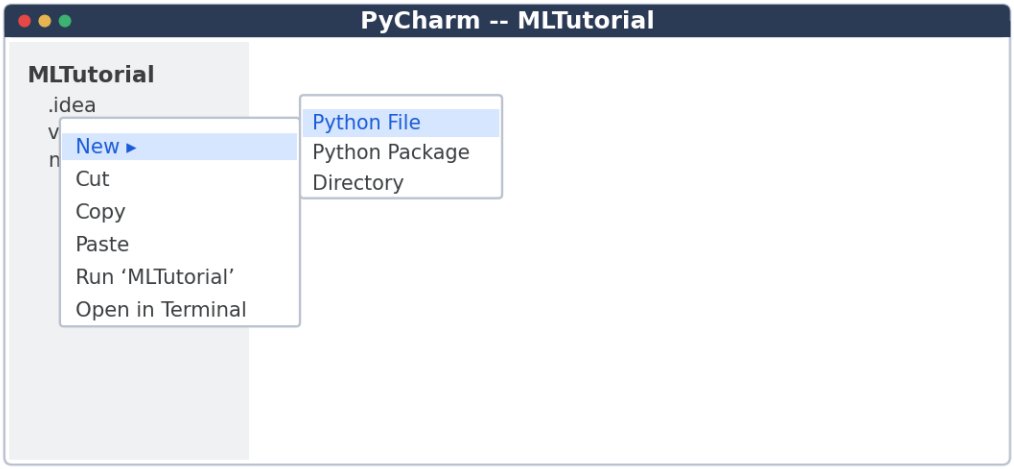

Figure 6: Right-click the project folder, then choose *New* → *Python File* from the context menu.

3.3 Writing the Code

Enter the following one-line program:

Listing 1: Hello World in Python

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

Note

Use straight double quotes (") rather than the curly "smart quotes" that word processors sometimes insert – curly quotes are not valid Python syntax and will raise a `SyntaxError`.

3.4 Running the Program

1. Right-click anywhere inside the `hello_world.py` file (or on the file in the Project panel).
2. Select **Run 'hello_world'**.
3. Alternatively, click the green **Run** arrow in the top-right corner of the editor, or press `Shift+F10`.

The **Run** console window opens at the bottom of the IDE and displays the program's output:

Figure 7: Running `hello_world.py` – the Run console at the bottom shows the program's output and exit code.

Note

If you see `Process finished with exit code 0` beneath the output, the program ran successfully – exit code `0` always indicates no errors occurred.

4 Debugging

Debugging lets you pause a running program, inspect its variables, and execute it one line at a time – far more informative than scattering `print()` statements through your code. We will debug a small function that sums the integers from 1 to n:

Listing 2: A program to debug

```
def compute_sum(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

result = compute_sum(5)
print(result)
```

4.1 Setting a Breakpoint

A **breakpoint** tells PyCharm where to pause execution so you can inspect the program's state.

1. Click in the left gutter (next to the line numbers), on the line `total += i`. A red dot appears, and the line is highlighted.
2. Click the same spot again to remove a breakpoint if you need to.

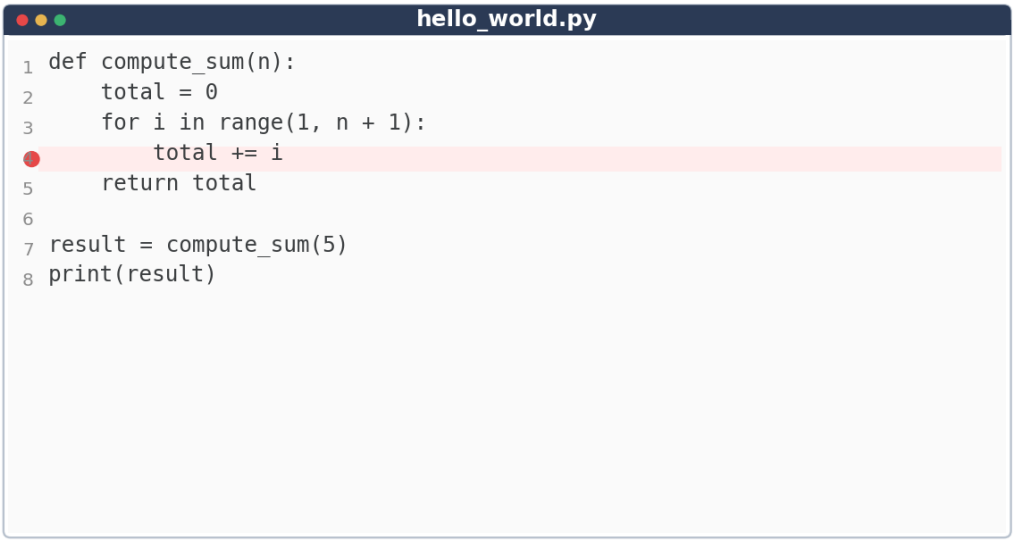

Figure 8: A breakpoint (red dot) set on the line `total += i`, inside the loop.

4.2 Starting the Debugger

1. Right-click anywhere in the file and select **Debug 'hello_world'** (instead of *Run*), or click the green **bug icon** in the top-right corner of the editor.
2. You can also press `Shift+F9` as a shortcut.
3. PyCharm runs the program normally until it reaches your breakpoint, then pauses and opens the **Debug** tool window at the bottom of the screen.

4.3 Stepping Through the Program

Once paused, use the step controls in the Debug toolbar to move through the code one instruction at a time:

- **Step Over** (`F8`) – executes the current line and moves to the next one, without entering any function it calls.
- **Step Into** (`F7`) – if the current line calls a function, jumps inside that function so you can trace it too.
- **Step Out** (`Shift+F8`) – finishes executing the current function and returns to whoever called it.
- **Resume Program** (`F9`) – continues running until the next breakpoint (or the end of the program).

Press **Step Over** repeatedly. Each time execution passes through `total += i`, the loop advances by one iteration.

4.4 Tracing Variables

While the program is paused, the **Variables** pane (bottom-left of the Debug tool window) lists every local variable in the current scope, updated live as you step through the code.

Figure 9: The Debug tool window: step controls along the top, call stack in *Frames*, and live values in *Variables* and *Watches*.

- **Frames** (left) shows the call stack – which function you are currently inside, and what called it.
- **Variables** (right) shows each variable's current value. In our example, watch `i` increase by one and `total` accumulate the running sum on every Step Over.
- **Watches** lets you add a custom expression (for example `total * 2`) and see it evaluated live, without changing your code. Right-click a variable and choose **Add to Watches**, or click the + button in the Watches pane.
- You can also hover over any variable name directly in the editor to see an inline popup with its current value.

Note

To evaluate an arbitrary expression on the fly – without adding a permanent watch – press `Alt+F8` to open **Evaluate Expression**, type any valid Python expression using the variables currently in scope, and press **Evaluate**.

4.5 Finishing Up

Once you are satisfied that the logic is correct, click **Resume Program** (`F9`) to let the program run to completion, or **Stop** (`Ctrl+F2`) to end the debugging session early. The final result, `15`, then prints to the console exactly as it would with a normal run.

5 Summary

- Installed **Python 3.14.7** with `PATH` configured.
- Installed **PyCharm 2026.2.1** (free core edition).
- Created a project-specific virtual environment.
- Installed **NumPy**, **Pandas**, **SciPy**, and **Scikit-learn** for future Machine Learning work.
- Created and ran a first `Hello World` program.
- Set a breakpoint, stepped through code line by line, and traced variables using the debugger.