

Getting started with Python scripting

PN204 | Posted on September 14, 2026 | Updated on September 14, 2026

This is an imperix knowledge base article, by Benoît STEINMANN, Jules PERRIN

Table of Contents

1. [Choosing a Python scripting environment](#)
 - 1.1. [Cockpit's built-in script launcher](#)
 - 1.2. [Standalone Python environment](#)
2. [Launching Python scripts from Cockpit](#)
3. [Setting up a standalone Python environment](#)
 - 3.1. [Installing Python](#)
 - 3.2. [Creating a Python virtual environment](#)
 - 3.3. [Installing the imperix Python library](#)
 - 3.4. [Running the Hello World script](#)
4. [Working with Python in Visual Studio Code](#)
5. [Going further](#)

Imperix [Cockpit](#) provides a graphical interface for managing real-time control software running on imperix controllers. It supports a wide range of actions, from testing transient response and tuning control parameters to long-term monitoring and logging. While direct interaction with the software suits most experiments, highly iterative manipulations can still be tiresome.

Starting with version 2026.3, the [ACG SDK](#) and [CPP SDK](#) include a **Python library** that provides programmatic access to Cockpit. This enables fully automated tasks in Cockpit using Python scripts. With them, users can easily run commands to repeat operations, make decisions based on collected data, and interface with third-party software. Scripting also enables automated data processing and export without manual intervention.

The [API documentation](#) provides ready-to-use code snippets and a complete reference of all available classes and methods. Respectively, this note introduces the basic structure of a script and explains how to run it either directly from Cockpit or from a standalone Python environment. A more complete, application-focused example is available in [PN205](#) and demonstrates using Python to sweep controller parameters to tune a PI controller.

Python script runs on the PC and controls the experimental workflow through Cockpit. The real-time control code still runs on the imperix controller and cannot be written in Python. Instead, it must be developed with the ACG SDK, using [Simulink](#) or [PLECS](#), or in C/C++.

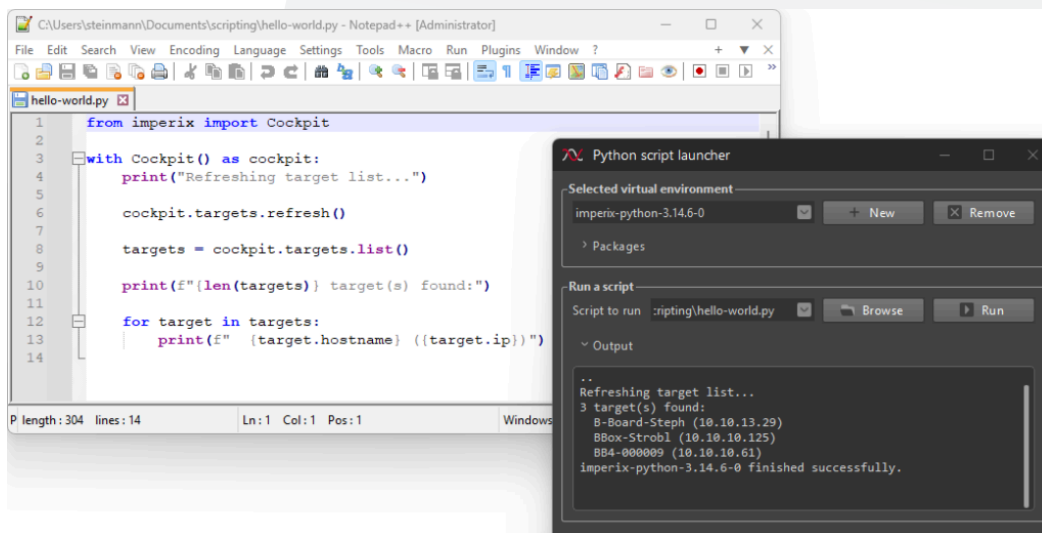
1. Choosing a Python scripting environment

Python scripts can be executed using Cockpit's built-in launcher or from a standalone Python environment. The built-in launcher is recommended for evaluating the global approach, developing very simple scripts, or executing existing scripts without the overhead of installing additional software. Beyond that, switching to a dedicated Integrated Development Environment (IDE) with a standalone Python environment is highly recommended.

1.1 Cockpit's built-in script launcher

Cockpit's script launcher requires zero configuration. The execution environment and the Python library are bundled in the SDK, meaning there is no need to install Python or configure system paths separately.

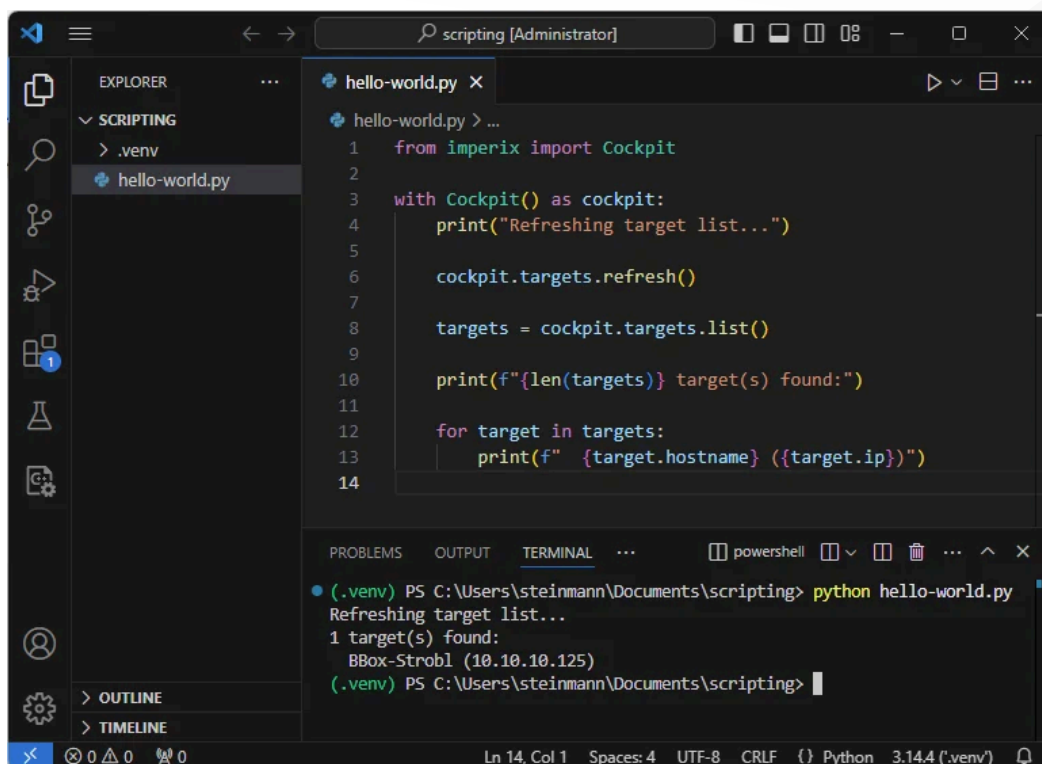
However, Cockpit does not include a code editor, so scripts must be written in a separate text editor, such as Notepad++. Additionally, execution is limited to a single .py file at a time.



Cockpit built-in Python script launcher

1.2 Standalone Python environment

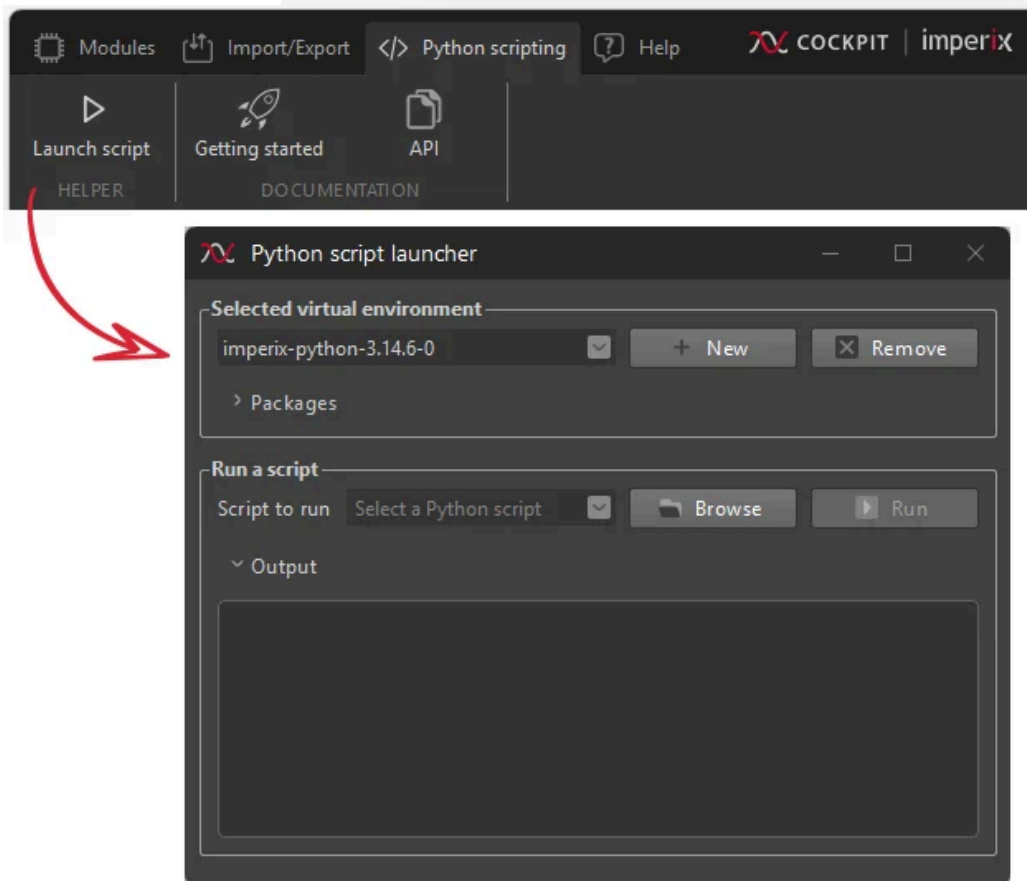
Setting up a standalone Python environment enables using dedicated IDEs such as VS Code. The initial setup is slightly more involved, as it requires installing Python independently, choosing an IDE, configuring a working environment, and manually installing the imperix Python library. On the other hand, these editors provide syntax highlighting, intelligent auto-completion, debugging tools, and support for structuring complex, multi-file projects, resulting in a much better user experience overall.



imperix Python library used in VS Code

2. Launching Python scripts from Cockpit

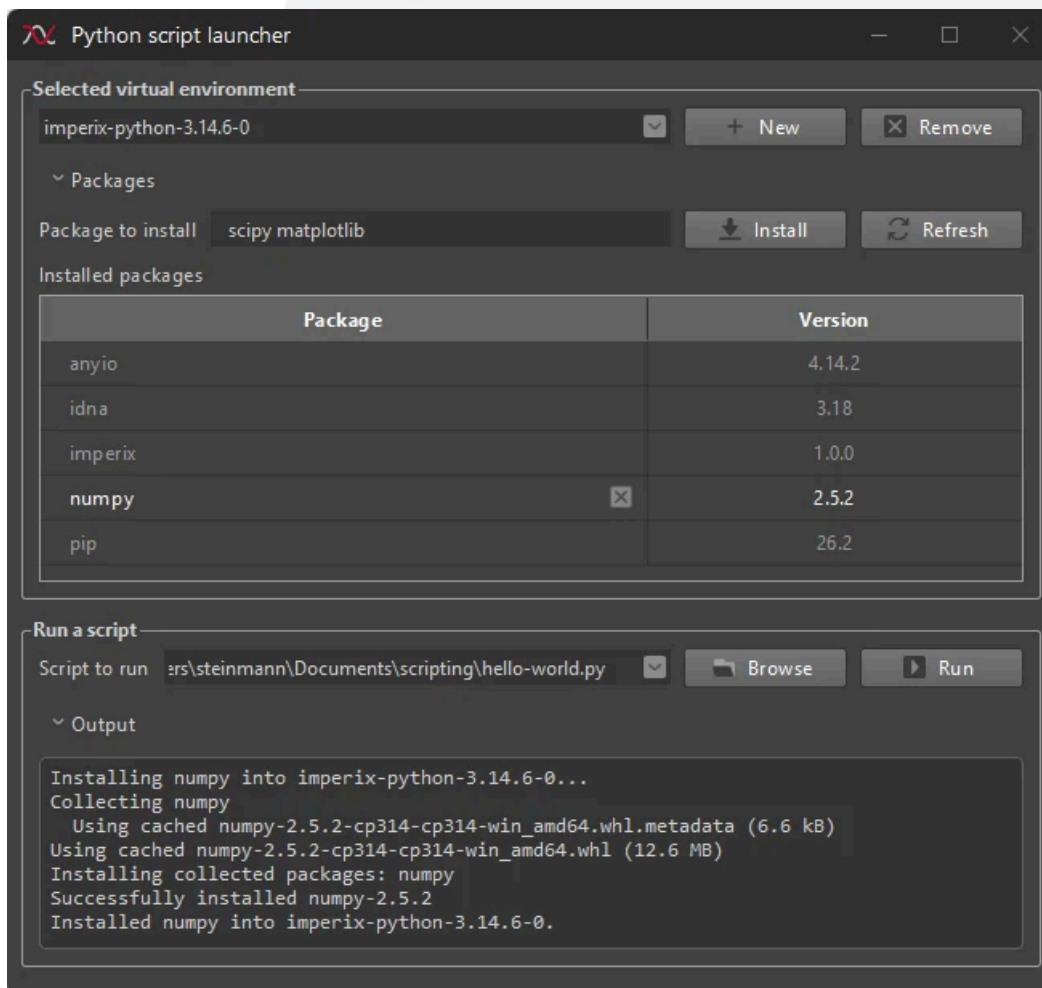
In Cockpit, the Python script launcher can be accessed by navigating to the **Python scripting** tab and selecting **Launch script**.



Opening Cockpit's Python script launcher

The script launcher window allows creating **virtual environments**. A Python virtual environment is an isolated workspace where scripts run. Each environment uses its own Python version (e.g. 3.14.6 in the screenshots) and its own set of **packages**.

Virtual environments created from Cockpit already include the imperix library. If additional packages are required, they can be installed via the Packages area.



Python packages installation from the script launcher

The hello-world.py script provided below illustrates the basic structure of an imperix Python script and provides a simple way to verify that the scripting environment works correctly. This script instructs Cockpit to refresh its target list, retrieves the list of detected targets, and displays their hostnames and IP addresses, similar to the list shown in the *Targets* perspective of Cockpit.

```
from imperix import Cockpit

with Cockpit() as cockpit:
    print("Refreshing target list...")
    cockpit.targets.refresh()

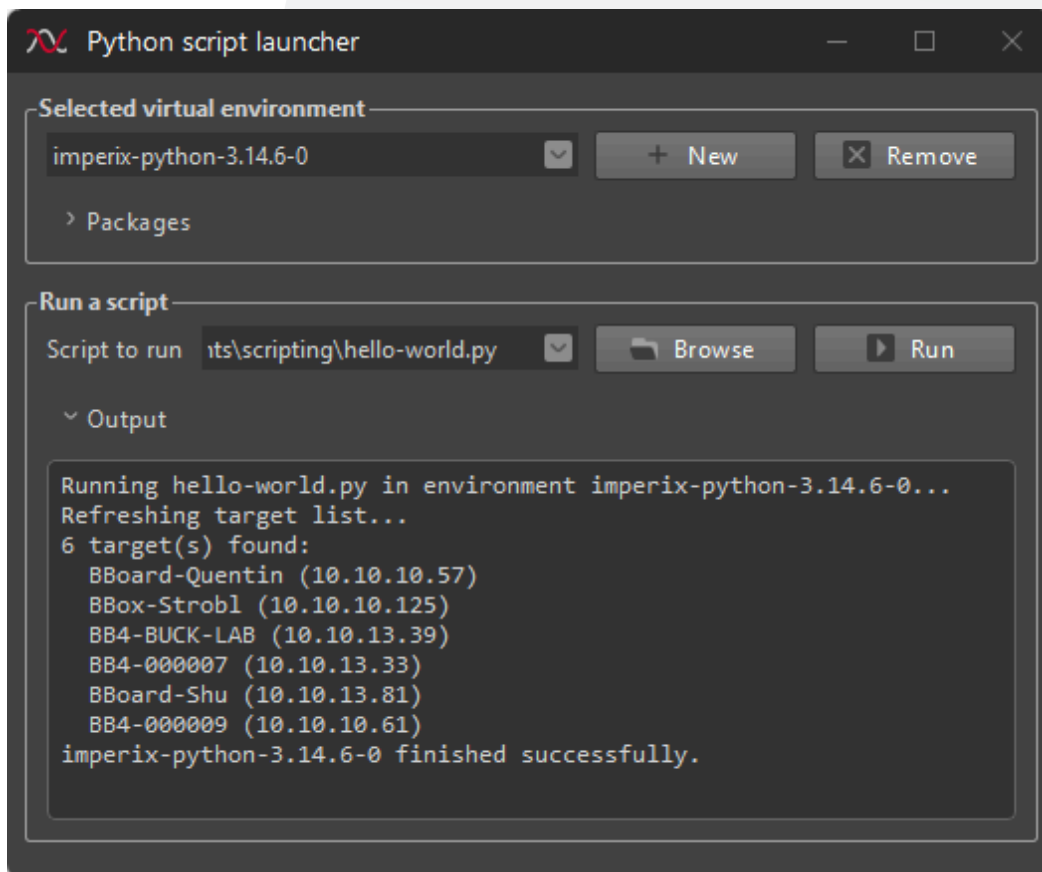
    targets = cockpit.targets.list()
    print(f"{len(targets)} target(s) found:")

    for target in targets:
        print(f"  {target.hostname} ({target.ip})")
```

The Hello World script is also available in the [API documentation](#).

To run the Hello World example, or any other script, follow these steps:

1. Select a virtual environment. If none exists, create a new one.
2. For this example, no extra packages are required. If another script is being run and requires additional packages, make sure to install them.
3. Create a new file named hello-world.py using a text editor (e.g., Notepad++) and paste the code from the code snippet above.
4. Browse to select the newly created .py script, then click Run. At this point, the list of available targets should be displayed in the output window.



hello-world.py executed from the script launcher

3. Setting up a standalone Python environment

Configuring an external Python environment is a prerequisite before integrating an IDE such as VS Code.

This step-by-step guide is designed for Python beginners. Experienced users who only require a quick reference can refer directly to the [installation](#) page of the API documentation.

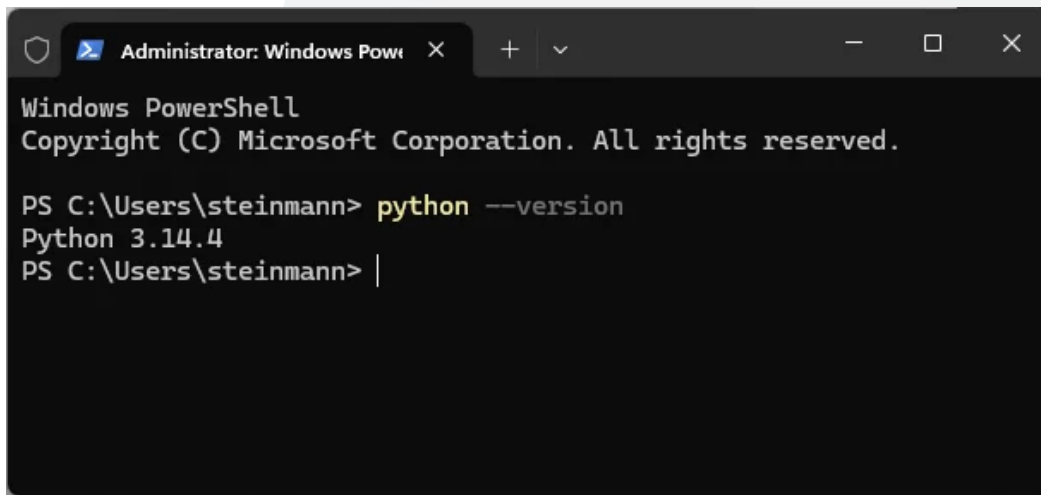
3.1 Installing Python

The imperix Python library requires Python 3.11 or later. It can be installed by following the official [Python guide for Windows](#).

To verify that Python was properly installed, open a terminal (either Command Prompt or PowerShell) and run the following command:

```
python --version
```

It should display the active Python version. If Python is missing or not added to the system PATH, an error will indicate that the term “python” is not recognized. In this case, the [troubleshooting section](#) of the official guide may help to resolve it.



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\steinmann> python --version
Python 3.14.4
PS C:\Users\steinmann> |
```

Verification that Python is installed

3.2 Creating a Python virtual environment

A Python virtual environment is an isolated workspace where scripts run. Each environment uses its own Python version and package set. Creating and activating Python virtual environments is optional but highly recommended, since it provides isolation between projects and consistency over time.

To learn more about virtual environments, refer to the [official Python documentation](#).

To create and activate a virtual environment:

1. Open a terminal in the project directory
 1. Right-click inside the folder and select **Open in Terminal**, or
 2. Navigate to the directory manually using:

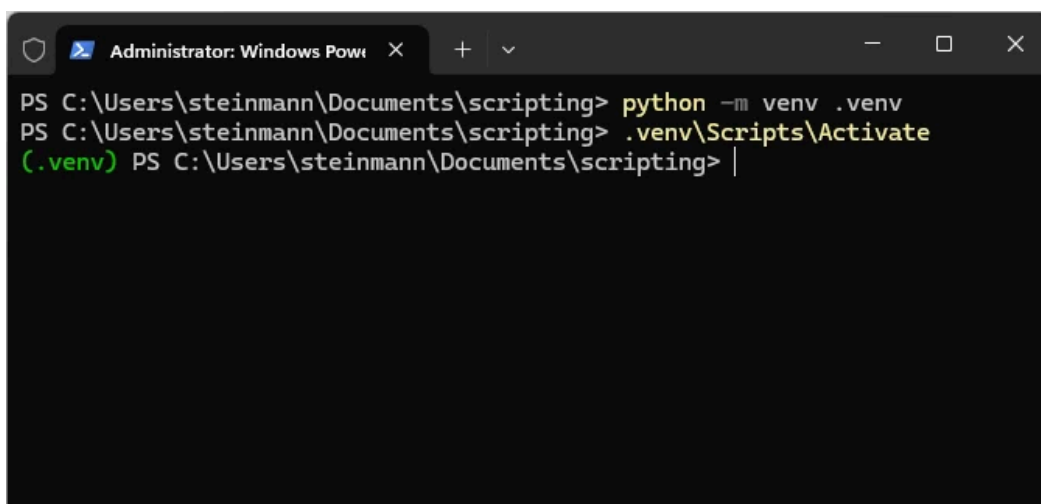
```
cd C:\path\to\director
```

2. Create a virtual environment by running the following command. A folder named `.venv` will be created in the project directory.

```
python -m venv .venv
```

3. Activate it by running

```
.venv\Scripts\Activate
```



```
Administrator: Windows PowerShell
PS C:\Users\steinmann\Documents\scripting> python -m venv .venv
PS C:\Users\steinmann\Documents\scripting> .venv\Scripts\Activate
(.venv) PS C:\Users\steinmann\Documents\scripting> |
```

Creation and activation of a Python virtual environment

Once activated, the terminal prompt displays (.venv) at the beginning of the line. Packages installed via pip (Python's package manager) are confined exclusively to the active virtual environment and will not affect the global system installation.

To exit the virtual environment, run: deactivate

PowerShell Security Error

If running Activate.ps1 produces a PSSecurityException error, PowerShell might be blocking script execution. Allowing the execution of scripts can be done by running the following command:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Alternatively, Window's Command Prompt (cmd) can be used instead.

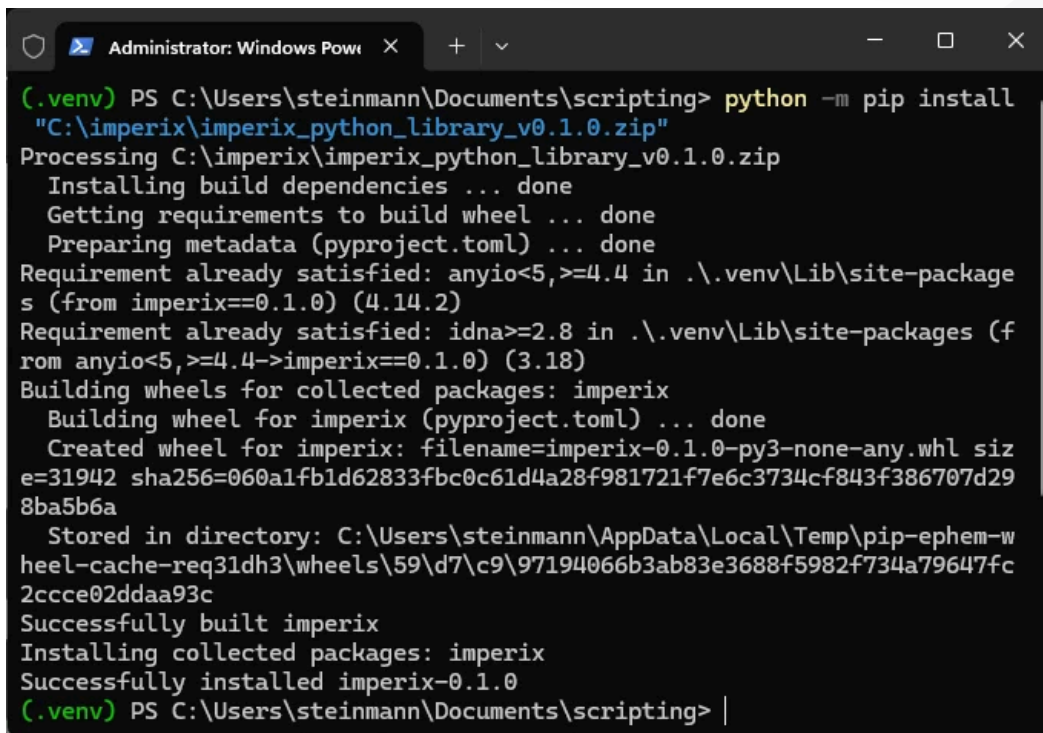
See [about execution policies](#) to learn more.

3.3 Installing the imperix Python library

To install the imperix Python library, download imperix_python_library_vX.X.X.zip from the [installation](#) page and install using pip by running the following command. If using a virtual environment, make sure it is activated (the terminal prompt displays (.venv))

```
python -m pip install "C:\path\to\imperix_python_library_vX.X.X.zip"
```

The full path can be retrieved by right-clicking the zip file and selecting **Copy as path**.



```
(.venv) PS C:\Users\steinmann\Documents\scripting> python -m pip install
"C:\imperix\imperix_python_library_v0.1.0.zip"
Processing C:\imperix\imperix_python_library_v0.1.0.zip
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: anyio<5,>=4.4 in .\.venv\Lib\site-package
s (from imperix==0.1.0) (4.14.2)
Requirement already satisfied: idna>=2.8 in .\.venv\Lib\site-packages (f
rom anyio<5,>=4.4->imperix==0.1.0) (3.18)
Building wheels for collected packages: imperix
  Building wheel for imperix (pyproject.toml) ... done
  Created wheel for imperix: filename=imperix-0.1.0-py3-none-any.whl siz
e=31942 sha256=060a1fb1d62833fbc0c61d4a28f981721f7e6c3734cf843f386707d29
8ba5b6a
  Stored in directory: C:\Users\steinmann\AppData\Local\Temp\pip-ephem-w
heel-cache-req31dh3\wheels\59\d7\c9\97194066b3ab83e3688f5982f734a79647fc
2ccce02ddaa93c
Successfully built imperix
Installing collected packages: imperix
Successfully installed imperix-0.1.0
(.venv) PS C:\Users\steinmann\Documents\scripting> |
```

Installation of the imperix library in the virtual environment (.venv)

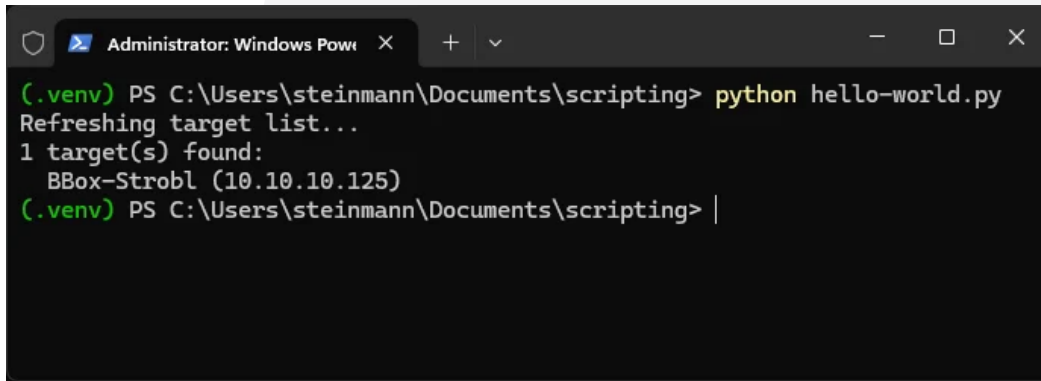
3.4 Running the Hello World script

To verify that everything was installed properly, run the Hello World script.

The Hello World script code is available in the *Launching Python scripts from Cockpit* section above, as well as in the [API documentation](#).

1. Create a new file named hello-world.py using a text editor (e.g., Notepad++) and paste the code from the code snippet provided [above](#).
2. Run the script using the command below. The output should provide the list of available targets.

```
python hello-world.py
```



```
Administrator: Windows PowerShell
(.venv) PS C:\Users\steinmann\Documents\scripting> python hello-world.py
Refreshing target list...
1 target(s) found:
  BBox-Strobl (10.10.10.125)
(.venv) PS C:\Users\steinmann\Documents\scripting> |
```

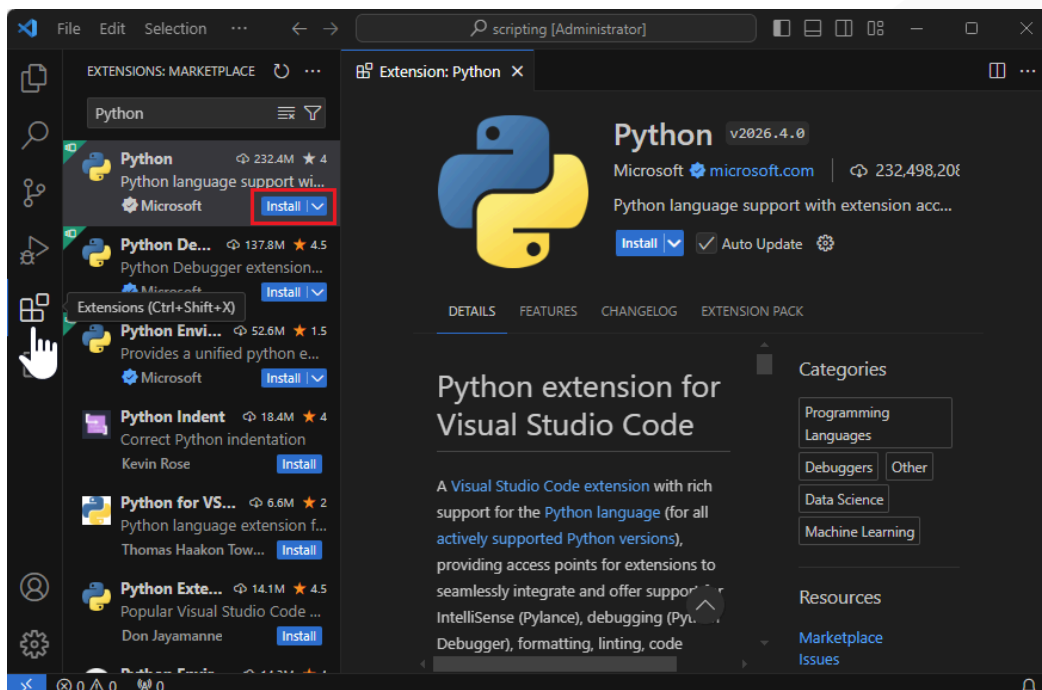
Running hello-world.py from a terminal

4. Working with Python in Visual Studio Code

Visual Studio Code (or VS Code) is an open-source code editor developed by Microsoft. This editor is used as an example, as it is currently the most widely adopted IDE for Python development. However, other popular IDEs, such as PyCharm or Spyder, can work just as well.

Before configuring VS Code, set up a standalone Python environment as explained above.

The installer is available from the official [download page](#). After installing and launching the editor, the Python extension can be installed from the *Extensions* tab on the left (Ctrl+Shift+X). This extension provides code completion, debugging tools, and other features, which are described further in the [Python in VS Code](#) documentation.



Installing the Python extension in Visual Studio Code

Next, open or create a project folder (such as the directory containing hello-world.py) and launch a terminal there. A project folder serves as the working directory containing all Python scripts (.py) and the associated virtual environment.

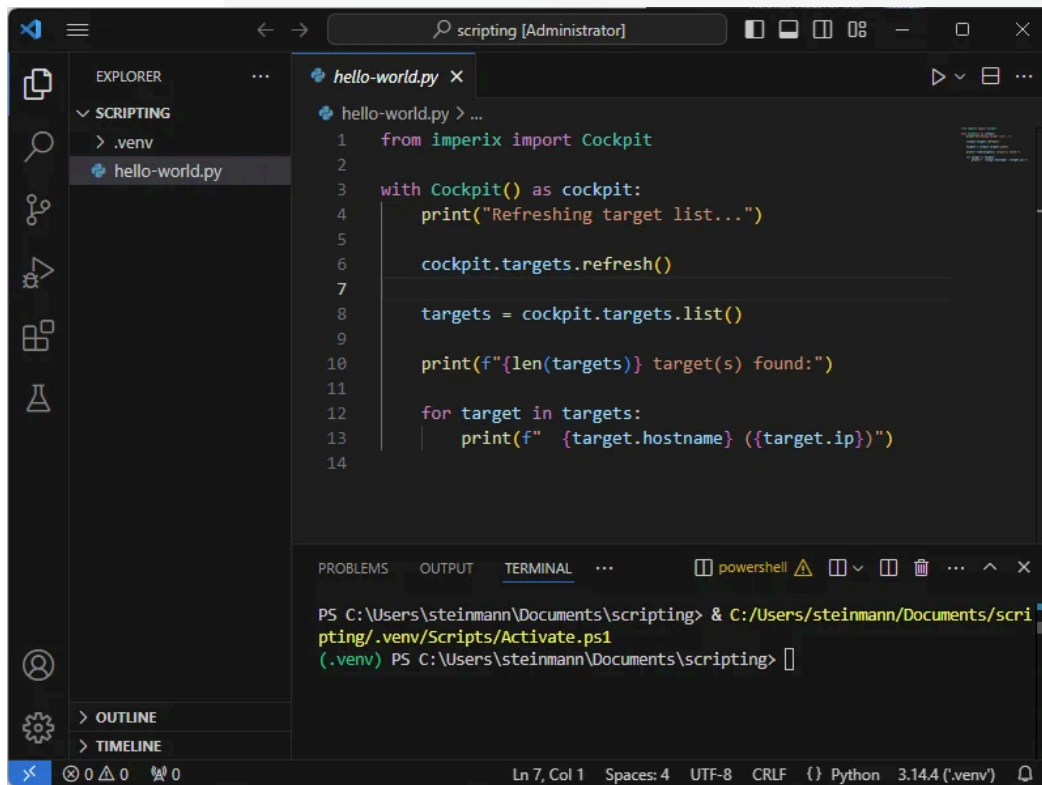
1. After installing the Python extension, close and reopen VS Code to make sure Python is properly detected
2. Open a project folder by selecting **File > Open Folder**
3. Open a Terminal by selecting **View > Terminal**.

If the project folder contains a virtual environment, VS Code should automatically detect it and activate it, as illustrated below (the line starts with (.venv)). If not, the virtual environment can be manually selected as

follows:

1. Navigate to Help > Show All Commands (or press Ctrl+Shift+P)
2. Type and select **Python: Select Interpreter**
3. Choose the Python executable inside the .venv folder (e.g., .\venv\Scripts\python.exe).

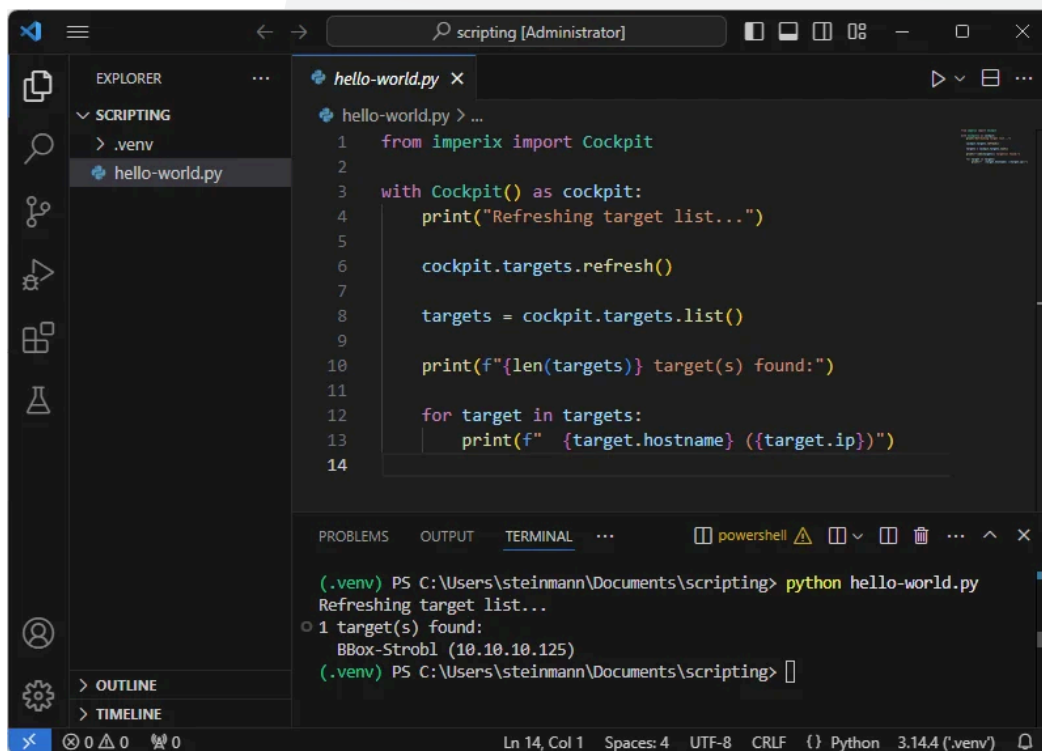
The **Python: Select Interpreter** command also enables creating new virtual environments directly, without executing terminal commands.



Opening a project folder in Visual Studio Code (here .venv is detected and automatically activated)

To verify the installation, run the hello-world.py script using the following command. Alternatively, click the play button (▶) on the top right.

```
python hello-world.py
```

A screenshot of the Visual Studio Code editor interface. The top bar shows the file explorer on the left with a tree view containing 'SCRIPTING' and a subfolder '.env'. The main editor window displays a file named 'hello-world.py' with the following Python code:

```
1 from imperix import Cockpit
2
3 with Cockpit() as cockpit:
4     print("Refreshing target list...")
5
6     cockpit.targets.refresh()
7
8     targets = cockpit.targets.list()
9
10    print(f"{len(targets)} target(s) found:")
11
12    for target in targets:
13        print(f"    {target.hostname} ({target.ip})")
14
```

The bottom panel shows the 'TERMINAL' tab with a PowerShell prompt. The command executed is `python hello-world.py`, and the output is:

```
(.venv) PS C:\Users\steinmann\Documents\scripting> python hello-world.py
Refreshing target list...
1 target(s) found:
  BBox-Strobl (10.10.10.125)
(.venv) PS C:\Users\steinmann\Documents\scripting>
```

The status bar at the bottom indicates the file is at line 14, column 1, using UTF-8 encoding, CRLF line endings, and is a Python 3.14.4 file in the '.venv' environment.

Running hello-world.py from Visual Studio Code

5. Going further

With the Python environment fully configured, the next step involves building and running automated control sequences. The [PN205](#) example presents a complete script that updates the Kp and Ki parameters and acquires the resulting response to a current reference step. This example serves as a practical template for building automation sequences and provides guidelines on the following key concepts:

- Creating a project and launching user code.
- Configuring the Scope module.
- Performing the experiment and exporting the resulting acquired data .