# Installing Python

## A short guide using uv

## Why we use `uv` for this course

`uv` is a new (and very fast) Python tool written in Rust. It: - Installs Python for you (no manual downloads). - Creates isolated *virtual environments* (safe sandboxes per project). - Installs and updates packages quickly.

> **i** Note
>
> WHAT is a virtual environment? Think of each project as its own coffee shop with its own supplies. One shop changing its menu does **not** affect the others. WHY it matters: You avoid random breakage when different projects need different versions of the same package.

## Install `uv`

Everything below happens in a **terminal**, a window where you type commands. On macOS open the Terminal app (Spotlight: `Cmd+Space`, type *Terminal*). On Windows open PowerShell (Start menu, type *PowerShell*). Choose the instructions for your operating system.

### macOS or Linux (Terminal)

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

If `curl` is missing:

```
wget -qO- https://astral.sh/uv/install.sh | sh
```

After installation: **close and reopen** your terminal (so your PATH updates).

### Windows (PowerShell)

Open PowerShell and run:

```
powershell -ExecutionPolicy Bypass -c "irm https://astral.sh/uv/install.ps1
| iex"
```

If you see a script execution warning, you can alternatively first run:

```
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process
```

Then re-run the install line.

### Verify installation

Run (macOS / Linux / Windows):

```
uv --version
```

If you see a version number: great!

> ⚠ Warning
>
> If you get “command not found” or “‘uv’ is not recognized”:
>
> 1. Close and reopen the terminal (important).
> 2. On Windows: make sure you used PowerShell (not Command Prompt).
> 3. Still broken? Ask for help, no need of guessing the error.

## Install Python

We want everyone on the **same** Python version for consistency. Thus, we’ll use Python 3.12 for the course this year.

Install (you only need to do this once):

```
uv python install 3.12
```

Check the installation:

```
uv run python --version
```

Expected output starts with:

```
Python 3.12.
```

## Create your first project

Pick a folder where you keep course work. **If you do not have one, make sure to create one!** In the terminal, move into that folder (`cd` followed by its path, or drag the folder onto the terminal window) and run:

```
uv init --no-package my-first-project
cd my-first-project
```

The first line creates a new folder named `my-first-project` (you can name it anything). The second line moves you into that folder. In Session X you will instead run `uv init --no-package` *inside* a folder cloned from GitHub; same command, no name.

`uv init --no-package` creates: - `main.py` (starter script) - `pyproject.toml` (project + dependencies config) - `.python-version` (records the Python version we chose) - `.gitignore` and `README.md` (only when the folder is new; a folder cloned from GitHub

already has both) - (A `.venv` folder will appear later once packages are added or synced.)

> **i** Note
>
> **Why `--no-package`?** Without it, recent versions of uv build an installable *package* layout (`src/my_first_project/`) with no `main.py`. For scripts, which is all this course needs, the flat layout is simpler.

You do **not** need to edit any of these (except maybe `README.md` for your notes and `main.py` if you want to run something different).

---

# Run the starter script

Inside the project folder:

```
uv run python main.py
```

You should see something like:

```
Hello from my-first-project!
```

(If you want, you can open `main.py` and change the message, then re-run.)

## What does that code mean?

```
def main():
    print("Hello from my-first-project!")


if __name__ == "__main__":
    main()
```

- `def main():` defines a function (a reusable block of code).
- `print(...)` shows text in the terminal.
- The line `if __name__ == "__main__":` ensures this only auto-runs when the file is executed directly.

Don't worry about this yet, we'll gradually build up to it.

# Run scripts directly in Zed

This section needs **Zed**, which you install with the AI-tools guide; come back to it afterwards. Instead of switching to the terminal every time, you can set up a **task** in Zed to run the currently open Python file with a single shortcut.

### Create the task file

Inside your project folder, create a `.zed/` directory and add a file called `tasks.json` inside it:

```
[
  {
    "label": "Run current Python file",
    "command": "uv run python $ZED_FILE",
    "use_new_terminal": false
  }
]
```

`$ZED_FILE` is a variable Zed fills in automatically with the path of the file you have open.

### Run the task

1. Open any `.py` file in your project.
2. Open the task picker with `Cmd+Shift+P` (macOS) / `Ctrl+Shift+P` (Windows/Linux).
3. Type **task** and choose **task: spawn**.
4. Select **Run current Python file**. Zed will execute `uv run python <your-file>` in the built-in terminal.

> Tip
>
> You can also bind the task to a keyboard shortcut. Open **Zed → Settings → Key Bindings** and add an entry for `"task: spawn"` to make it even faster.

## Adding packages (later in the course)

If/when you need a package (example: `pandas`):

```
uv add pandas
```

If you added the wrong one:

```
uv remove pandas
```

If your `pyproject.toml` changed (e.g. you pulled code from someone else):

```
uv sync
```

> Tip
>
> If something seems “off”, just close the terminal and reopen in the project folder. Fresh starts fix many early mistakes.

## Updating `uv`

Occasionally:

```
uv self update
```

(If it ever errors, you can just reinstall using the same one-liner from earlier.)

---

## Best practices for this course

- One folder for the course keeps everything tidy.
- Never install packages "globally" outside a project.
- Keep a short personal log in each project's `README.md` (What did I do? What still confuses me?).
- Ask early for help, guessing usually takes much more time than asking.

You can always see available commands:

```
uv --help
```

## Recap

You can now: 1. Install `uv`. 2. Create a project. 3. Run a script. 4. Add/remove/sync packages.

Now, you're set to continue the course.