Installing Python

A beginner-friendly 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`

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 (and pin) 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.
```

> ℹ Note
>
> Why pin a version? If the latest Python version moves forward mid-semester, subtle bugs can appear that might break working code (unlikely, but possible). Pinning keeps everyone in the course aligned.

Create your first project

Pick a folder where you keep course work. If you do not have one, make sure to create one! Open the course folder in your IDE and then run the following from the terminal:

```
uv init 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. Alternatively, you can create the folder manually before, open it in your IDE and run `uv init .` inside it.

`uv init` creates: - `main.py` (starter script) - `pyproject.toml` (project + dependencies config) - `.python-version` (records the Python version we chose) - `.gitignore` (useful if you ever use Git) - `README.md` (you can jot notes here) - (A `.venv` folder will appear later once packages are added or synced.)

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 World!
```

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

What does that code mean?

```
def main():
    print("Hello, World!")

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.

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
```

Update everything to newer allowed versions:

```
uv update
```

(Optional) Activating the virtual environment

You can either: - Always run scripts with `uv run ...` (safest for beginners), OR - "Activate" the environment so `python` refers to the project's Python directly.

Activate (macOS / Linux):

```
source .venv/bin/activate
```

Activate (Windows PowerShell):

```
.venv\Scripts\Activate.ps1
```

You'll see your prompt change (e.g. `(my-first-project)` at the start of the terminal line).

Now you can just run:

```
python main.py
```

Deactivate when done:

```
deactivate
```

> 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 project folder per session or assignment keeps everything tidy.
- Never install packages "globally" outside a project.
- Prefer `uv run python <file>` instead of environment activation.
- 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.

10. Where to go next

Explore the official docs later if you're curious: https://docs.astral.sh/uv/

You can always see available commands:

```
uv --help
```

Recap

You can now: 1. Install `uv`. 2. Install a pinned Python version. 3. Create a project. 4. Run a script. 5. Add/remove/sync packages.

Now, you're set for the rest of the course.