School of Games and Creative Technology
BSc (Hons) Computer Science

University for the Creative Arts

FGCT6021 MOBILE APPLICATION DEVELOPMENT
LAB 5 – TOPIC 5 PYTHON & JSON BACKEND DATA

Submission

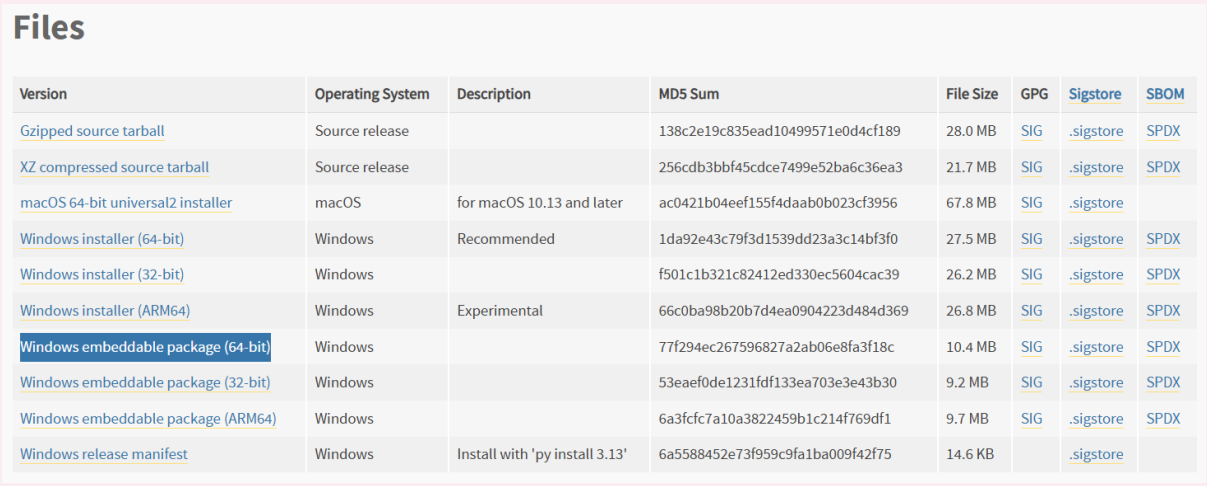
1. Submit your Learning Journal link (GitHub) and weekly journal entry on myUCA.
2. During the workshop, you will show your progress and share what you have learned.

Task 1 Set up Python in Visual Studio Code

If you cannot install Python because you lack admin rights, you can still run Python in Visual Studio Code using a portable version.

Step 1: Download Portable Python

1. Go to Python Releases for Windows
2. Find the version you want (e.g., Python 3.13.7).
3. Scroll down to `Files` > select **Windows embeddable package (64-bit)**.



Files

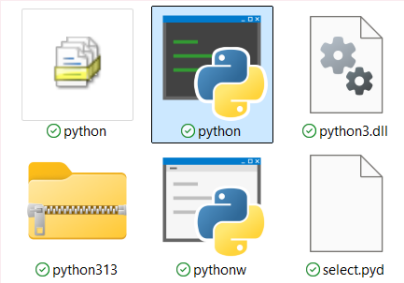
Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore	SBOM
Gzipped source tarball	Source release		138c2e19c835ead10499571e0d4cf189	28.0 MB	SIG	.sigstore	SPDX
XZ compressed source tarball	Source release		256cdb3bbf45cdce7499e52ba6c36ea3	21.7 MB	SIG	.sigstore	SPDX
macOS 64-bit universal2 installer	macOS	for macOS 10.13 and later	ac0421b04eef155f4daab0b023cf3956	67.8 MB	SIG	.sigstore	
Windows installer (64-bit)	Windows	Recommended	1da92e43c79f3d1539dd23a3c14bf3f0	27.5 MB	SIG	.sigstore	SPDX
Windows installer (32-bit)	Windows		f501c1b321c82412ed330ec5604cac39	26.2 MB	SIG	.sigstore	SPDX
Windows installer (ARM64)	Windows	Experimental	66c0ba98b20b7d4ea0904223d484d369	26.8 MB	SIG	.sigstore	SPDX
Windows embeddable package (64-bit)	Windows		77f294ec267596827a2ab06e8fa3f18c	10.4 MB	SIG	.sigstore	SPDX
Windows embeddable package (32-bit)	Windows		53eaef0de1231fdf133ea703e3e43b30	9.2 MB	SIG	.sigstore	SPDX
Windows embeddable package (ARM64)	Windows		6a3fcfc7a10a3822459b1c214f769df1	9.7 MB	SIG	.sigstore	SPDX
Windows release manifest	Windows	Install with 'py install 3.13'	6a5588452e73f959c9fa1ba009f42f75	14.6 KB		.sigstore	

4. Download the zip file python-3.13.7-embed-amd64.zip.

Step 2: Extract Python

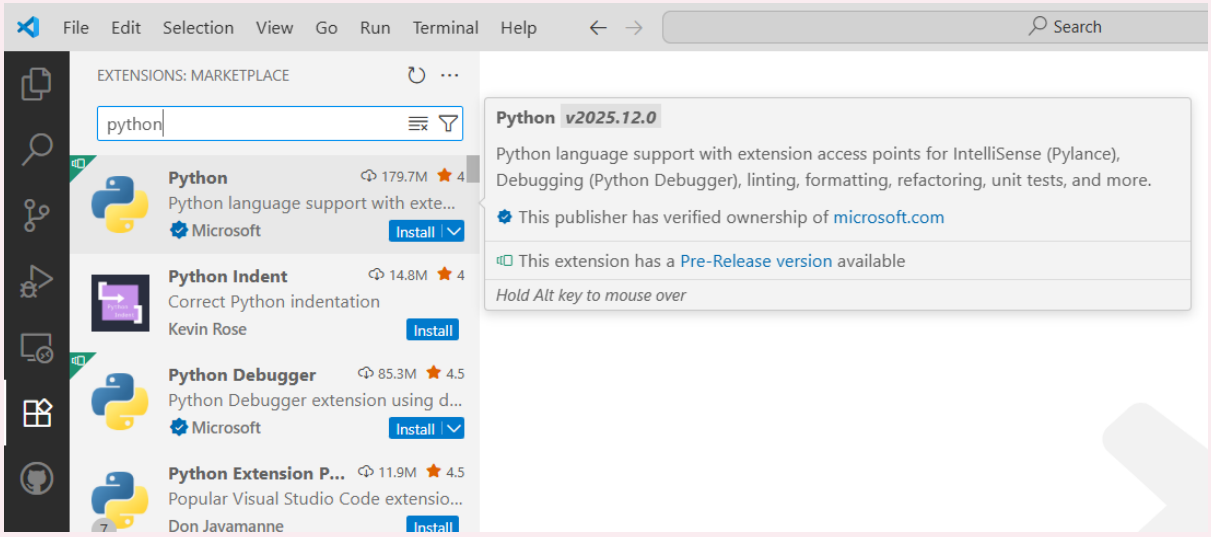
1. Create a folder you can access, e.g., `C:\Users\YourName\Python`.
2. Extract the contents of the zip file into that folder.

3. Inside, you will see `python.exe`. This is your portable Python executable.



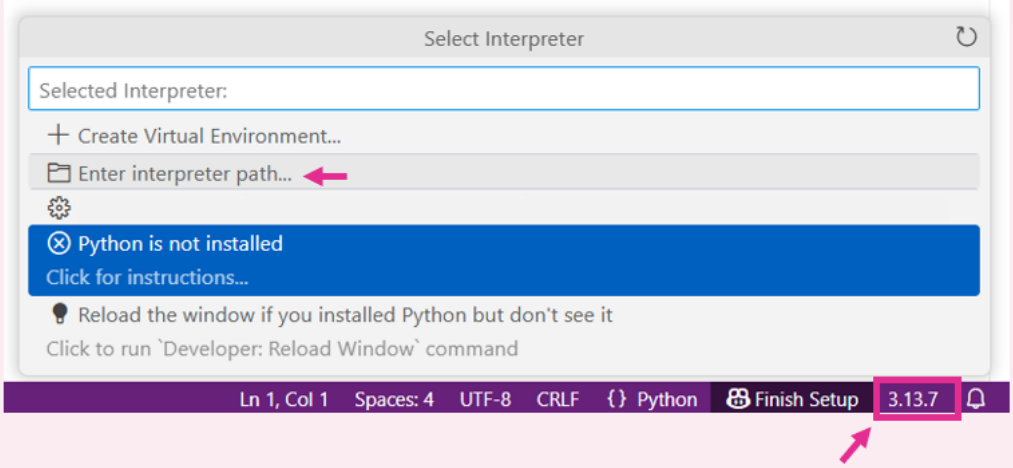
Step 3: Install the Python Extension in Visual Studio Code

1. Open Visual Studio Code.
2. Go to `Extensions` (`Ctrl+Shift+X`).
3. Install `Python (by Microsoft)`.



Step 4: Configure Visual Studio Code to Use Portable Python

1. Open Visual Studio Code.
2. Press `Ctrl+Shift+P` > type `Python: Select Interpreter`, and press Enter.
3. Click `Enter interpreter path > Find...` or select:



4. Navigate to the folder where you extracted Python and select `python.exe`.
5. Visual Studio Code will now use this Python version for running scripts.

Step 5: Test Python

1. Create a new file `example.py` in Visual Studio Code:

```
print("Hello from Python!")
```

2. Click the `Run` button in Visual Studio Code.

3. You should see the output in the terminal:

Task 2 Workshop Overview

You will extend your Learning Journal PWA by introducing **Python** and **JSON** as tools for storing your reflections in a file. While your HTML, CSS, and JavaScript continue to control the interface, Python will allow you to create and update a JSON file. Your PWA will then display these entries. This introduces the idea of file-based storage beyond the browser.

Your goal this week is to:

- Create and update a **JSON file with Python**.
- Fetch and **display JSON entries** in your PWA.
- Extend your PWA with **an additional feature** that **makes use of the JSON file**.

Goal

Add JSON file storage to your Learning Journal PWA. Entries should be written using Python, then displayed in the PWA through JavaScript.

All files should be committed to your **GitHub** repository so progress is visible and version-controlled.

What to Do

The following tasks are suggestions. You may adapt them as appropriate to your own design.

Folder and File Structure

1. Continue using the folder structure from Week 4:
   - `/css` Stylesheets
   - `/js` JavaScript files
   - `/images` Media assets
   - `/backend` JSON and Python files

2. Add the following files, for example:
   - `/backend/reflections.json` your data file
   - `/backend/save_entry.py` Python script to add entries

Python and JSON

1. Save Create an empty JSON file `reflections.json` containing `[]`.

2. Write `save_entry.py` to:
   - Ask the user to type a reflection (input).
   - Append the reflection with a date and other information into `reflections.json`.
   - Save the updated JSON file.

3. Check that `reflections.json` updates correctly.

Front-End (HTML, CSS, JavaScript)

1. Fetch JSON Data
   - Use `fetch("backend/reflections.json")` in JavaScript.
   - Parse and display the entries inside your journal page.

2. DOM Manipulation
   - Use your existing HTML/CSS for styling.
   - Insert JSON entries dynamically into the DOM (date + reflection text).

3. Extend with an Extra Feature
   - Add **at least one new feature** to your PWA that uses the JSON file.
   - Use DOM events (click, input, submit) to interact with your feature.
   - Examples (suggestions only you may come up with your own idea):
     - Add an **Export button** to download `reflections.json`.
     - Add an **Import button** to load another JSON file.
     - Display reflections with **filters** (by date, keyword, length).
     - Add a **counter** that shows the number of reflections.

Journal Questions

For your weekly entry, **submit it on myUCA** and **post it on the Learning Journal Web**:

1. How is storing data in a JSON file different from using browser storage?
2. How did you use Python to create or update your JSON file?
3. What does your PWA show locally, and what will users see on GitHub? Are they the same? Why or why not?
4. What extra feature did you add to your PWA using the JSON file, and why?