PAPI - Python for Analysis and Processing of Images

Introductory Session - Program

Day 1	Thursday	**Day 2**	Friday
Time	**Topic**	**Time**	**Topic**
8:45–9:00	Registration + morning coffee & tea	9:00–10:30	Image processing – transformations, filters, segmentation
9:00–10:00	Welcome + Intro to virtual environments, Python & Jupyter Notebook	10:30–10:45	Coffee break
10:00–11:30	Python basics – syntax 1	10:45–12:00	Image analysis – measurements, plotting
11:30–12:20	Lunch break	12:00–12:45	Lunch break
12:20–14:00	Python basics – syntax 2	12:45–14:45	Automating image analysis – batch processing
14:00–14:20	Coffee break	14:45–15:00	Coffee break
14:20–16:00	Packages: numpy, matplotlib, pandas, image I/O	15:00–16:00	Automating image analysis – batch processing
16:00–16:20	Coffee break	16:00–16:15	Coffee break
16:20–17:30	Recap exercises	16:15–17:30	Napari session – GUI

Welcome & Course Introduction

- **Introductory Session (Today & Tomorrow):** No prior coding experience needed. We start from zero, get Python running, and learn the fundamentals.
- **Advanced Session (Next Week):** Builds on the basics. We'll tackle more complex workflows and introduce AI tools.

This course is all about **learning by doing**.

- Focus on writing and running code, with image analysis concepts introduced only as they come up in practice. The goal isn’t to teach image analysis theory, but to help you get comfortable coding and explore some useful tools.
- If interested in the theoretical background, you’ll find links to additional resources you can explore on your own.
- *disclaimer:* Time is limited. We will show some ways you can use Python, but it is not the only way.

Why Python for (Image) Data Analysis

- **Why Python?**
  Widely used. | Vast community support. | Low entry barrier (readable syntax).

- **Why Python for Data Analysis?**
  Widely used in the scientific community.

  - Huge ecosystem of free, open-source libraries (pre-written code) specifically for scientific tasks.
  - Easy data manipulation and visualization.
  - Effort to make it compatible with other existing tools.

Why learn to code when you can chat with AI assistants?

You can ask this question the AI and it might humbly tell you that:

- **AI-generated code is not perfect**. It will have bugs, or it won't be perfectly suited to your unique data. When the code breaks, you, the coder, need the fundamental knowledge to understand the error messages and fix the code. You cannot debug what you do not understand.
- In science, you must be able to stand behind your results. **You need to know what** every line of **your analysis code is doing**.
- For some tasks, repeatedly prompting and debugging with AI can take longer than writing solid code yourself once you know the basics.

Python Installation and Environment Management

Virtual Environments (Conda/venv)

- Instead of using your operating system 'GLOBAL' Python, we use a virtual environment.
- Virtual environment is a self-contained, isolated 'LOCAL' setup that holds a specific version of Python and a specific set of libraries.
- This allows you to install/remove packages without affecting the global Python.
- You can create as many environments as you want. They don't interfere with each other.
- Best practice: create a dedicated environment for each project

Our Toolkit: Conda

- **For this course:** We'll use `conda` (or mamba) to create a dedicated environment with all the necessary libraries. When you start your terminal, you will "activate" this environment so Python uses the local interpreter inside it.
- **Conda** is both an **environment manager** and a **package manager**.
    - **Environment Manager:** It creates, activates, and deletes these isolated "setups".
    - **Package Manager:** It installs, updates, and removes software packages (like `numpy`, `pandas`, `napari`) inside the active environment.
    - *Note:* Conda can manage any software, not just Python packages. This is crucial for scientific libraries that rely on compiled C, C++, or Fortran code.

Conda as package manager

- Packages to be installed are available in conda channels.
- The two most important channels are:
  - `defaults`: The main channel maintained by *Anaconda, Inc.*
  - `conda-forge`: A community-led channel.

When you type `conda install some-package -c conda-forge`, you're telling Conda: "*Install `some-package`, and look for it in the `conda-forge` channel.*"

Mamba: Mamba is an alternative to Conda. It uses the same commands and channels but typically resolves dependencies faster. Simply switch word conda for mamba.
`mamba install some-package -c conda-forge`

PIP: `pip` is the standard **package manager** for Python. It is not an environment manager. And it *only* manages Python packages.

Environment specification files

You can create an envirovnment and install packages one by one, or you can have a ready-made text file that specifies the environment's name, the channels to use, and all the packages (dependencies) to install.

- `environment.yml` - The Conda Way
- `requirements.txt` - The Pip Way
    - A simpler text file that just lists packages for *pip* to install. Doesn't specify channels or the environment name.

Open the `environment.yml` (in the setup folder) in text editor to inspect what it looks like.

Exercise 0

Please proceed with Exercise 0 to set up the environment:

- file: E00_setup.md
- `link to instructions`

Steps:

- Install **Miniforge** and **VS Code** IDE (as described in the instructions) if you haven't done so beforehand.
- Open terminal and navigate to the folder with **environment.yml**
- Run the command to create the environment and install packages.

Best Practices for Environment Management

Following these rules will save you and your collaborators countless hours of frustration.

1. **Leave the base environment alone!**
   `base` contains the tools (conda itself) needed to manage all the other environments. If installation conflicts with conda's dependencies, you can corrupt your Conda installation.
2. One environment per project.
3. Name your environments sensibly. Give your environments descriptive names related to the project, not generic names like *test*, *python*, *my-env*, ...
4. Clean up old environments: Conda environments can take up significant disk space.

Conda Commands

List all environments: `conda env list`
Create a new environment: `conda create --name myenv python=3.10`
Create an environment from a YAML file: `conda env create -f environment.yml`
Activate an environment: `conda activate myenv`
Deactivate the current environment: `conda deactivate`
Remove an environment by name: `conda env remove --name myenv`

Package management:

- `conda list` - lists all installed packages in the current environment
- `conda install numpy`
- `conda update numpy`
- `conda remove numpy`

Writing and Running Code

1. Plain Text Files (`.py` scripts):

- You write your code in a file (e.g., `my_analysis.py`).
- You run the *entire file* from top to bottom using the terminal (`python my_analysis.py`).

2. Jupyter Notebooks (`.ipynb` files):

- A special file format where you can mix code, text and plots in one document.
- You can write and run code in small chunks called "cells".
- *Note*: **Google Colab:** A free, cloud-based version of Jupyter Notebooks from Google. Can be an alternative if you can't install software locally or you want some GPU.

IDE (Integrated Development Environment) is a text editor designed for coding (gives suggestions and autocompletes as you type the code). *VS Code, PyCharm, Spyder, ...*

Writing and Running Code

The Python Interpreter:
The interpreter is the “engine” that reads and executes your Python code.

- When you type `python` in your terminal, you are starting the interpreter.
- When you run `python my_analysis.py`, the interpreter executes the entire file from top to bottom.
- In Jupyter Notebooks, each cell you run sends code to the **kernel**, which acts as a live Python interpreter running in the background.

Exercise 1

Time for a first exercise.

Please open and follow instructions for Exercise 1:

- file: **E01_first_script.md**
- `link to instructions`

Exercise 2

Let's investigate how to work with Jupyter Notebooks.

Please open notebook **E02_first_notebook.ipynb** with located at:

```
PAPI/materials/session_1
```