PAPI - Python for Analysis and Processing of Images

Introductory Session - Program

Day 1	Monday (Oct 13)	**Day 2**	Tuesday (Oct 14)
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 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, statistics
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	Napari session – GUI
16:00–16:20	Coffee break	16:00–16:15	Coffee break
16:20–18:00	Recap exercises	16:15–18:00	Recap exercises

Welcome & Course Introduction

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

Learning Style: 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 deep image analysis theory, but to help you get comfortable coding, exploring data, and building useful tools.
- If interested in the theoretical background, you’ll find links to additional resources you can explore on your own.

Why Python for (Image) Data Analysis

- **Why Python?**
  Widely used. | Vast community support. | Easy learning curve (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.

Course Goals

- Show what Python can do so you can automate all of the boring, mundane, time-sucking tasks.
- The skills you learn here aren't just for images. You can apply them to data analysis, plotting, automation, and much more.

Disclaimer 1:
Time is limited. We will show some ways you can use Python, but it is not the only way,

Disclaimer 2:
We want to teach some useful concepts and ways to automate analysis and processing tasks using Python, but we're not trying to turn you into software developers

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 complex 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

The Problem: Imagine you're working on two different projects.

- Project A (from last year) requires an old version of a library. Project B (your new project) needs a feature from newer library version.
- If you just install everything in one main "system" Python, installing the new version might break Project A.

The Solution: Virtual Environments (Conda/venv).

- Instead of using your operating 'system' Python, we use a virtual environment.
  = a self-contained, isolated "project box" that holds a specific version of Python and a specific set of libraries
- You can create as many environments as you want. They don't interfere with each other.

Our Toolkit: Conda

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

Exercise 0

Please proceed with Exercise 0 to set up the environment:

- file: E00_setup.md
- link: `https://github.com/IMCF-Biocev/PAPI/blob/papi_2025/materials/session_1/E00_setup.md`

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.

Conda Channels

- Packages to be installed are available in conda channels.
- The two most important channels are:
  - `defaults`: The main channel maintained by Anaconda, Inc. It's reliable but can sometimes be slightly out of date.
  - `conda-forge`: A massive, community-led channel. It has a wider selection of packages and is often more up-to-date.

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 a replacement for 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.
- **Best Practice / Rule of Thumb:**
    1. Always use **Conda** to create and manage your environment.
    2. Install as many of your packages as possible using **Conda**.
    3. If a package is *not available* on any Conda channel, activate your Conda environment and then use **pip** to install that specific package.
        - `conda` and `pip` can coexist happily inside a single Conda environment.

Environment specification files

You can create and 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.

Best Practices for Environment Management

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

1. One environment per project.
2. 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.
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`
Recreate 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_program.md**
- link: `https://github.com/IMCF-Biocev/PAPI/blob/papi_2025/materials/session_1/E01_first_program.md`

Exercise 2

Let's investigate how to work with Jupyter Notebooks.

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

```
PAPI/materials/session_1
```