Introduction to Python

BoBiAC
Boston Bioimage Analysis Course | 2026

Dr Federico Gasparoli

A <u>brief overview</u> of key Python concepts you need to know to get started

!! While Python concepts are universal, the tools used to manage them vary !!

This is an opinionated introduction!

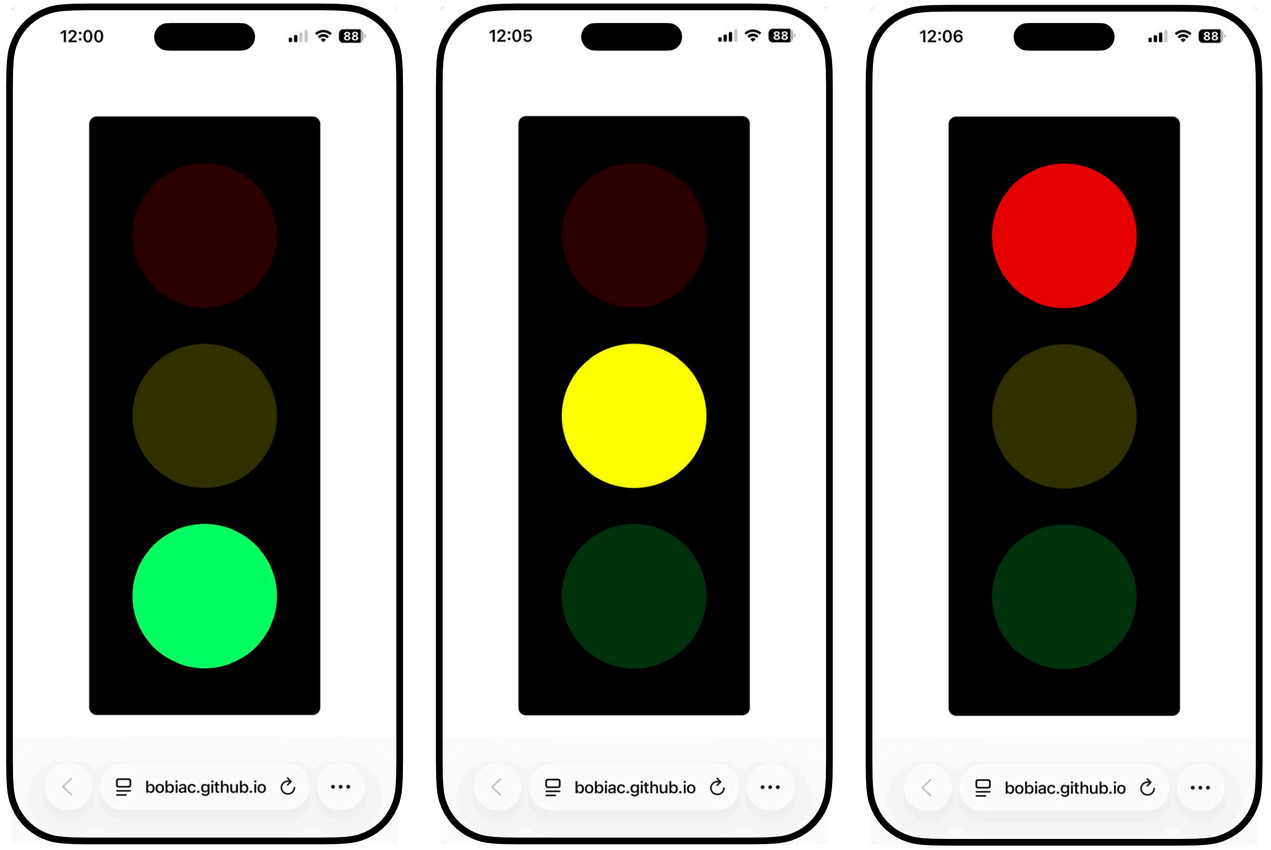
classlight

By Talley Lambert

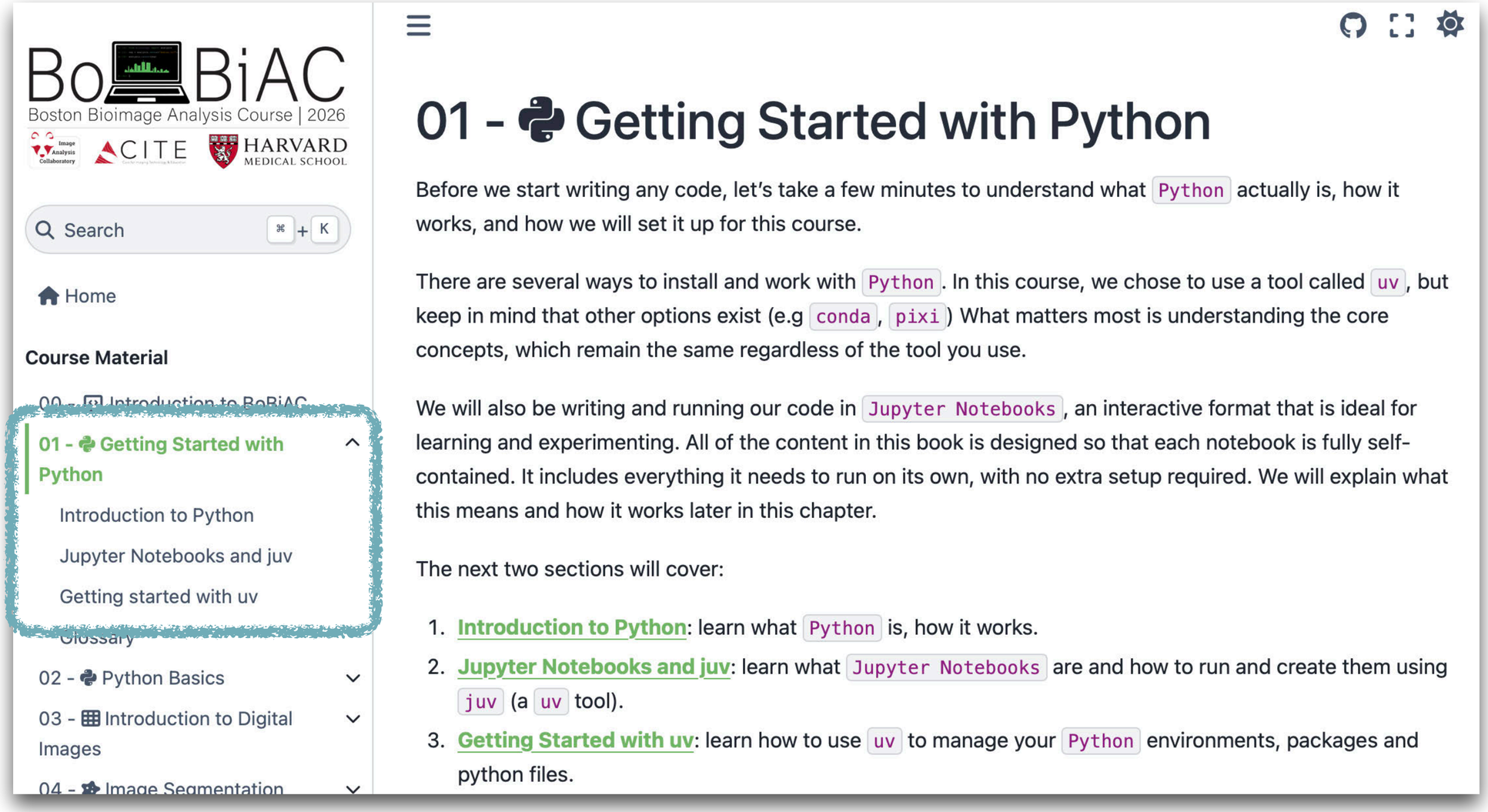
bobiac.github.io

bobiac.github.io/classlight/bobiac2026

Realtime Feedback!



Introduction to Python



Introduction to Python - Overview

Part I - General Python Concepts

- What is Python?
- What are Python packages?
- What are virtual environments? Why are they useful?

Part II - Jupyter Notebooks and *juv*

- What are Jupyter Notebooks?
- What is Jupyter Lab?
- How can use ***juv*** to deal with Python, packages and virtual environments?
- How can use ***juv*** to run Jupyter Notebooks?

Part III - Python and *uv*

- How do we use ***uv*** to manage python, virtual environments, and packages
- How can we use ***uv*** to run Python code?
- How can we create a Python file?

Introduction to Python

Part I - General Python Concepts

What is Python?

When people say `Python`, they can mean two things:

1. **The Programming Language**: a way that allows us to write instructions that a computer can understand and carry out based on specific **rules** and **syntax**.
2. **The Interpreter**: the actual program (also called `Python`) installed on your computer that is able to **read** the instructions and **execute** them.

Note

🍕 **Analogy**: Think of the **Programming Language rules** and **syntax** as the **grammar** and **vocabulary** we use to write **a pizza recipe**, and the **Interpreter** as the **cook** who **reads**, **understands**, and **executes** the instructions to make a wonderful pizza.

- Totally free & open-source
- Easy to learn, read, and write

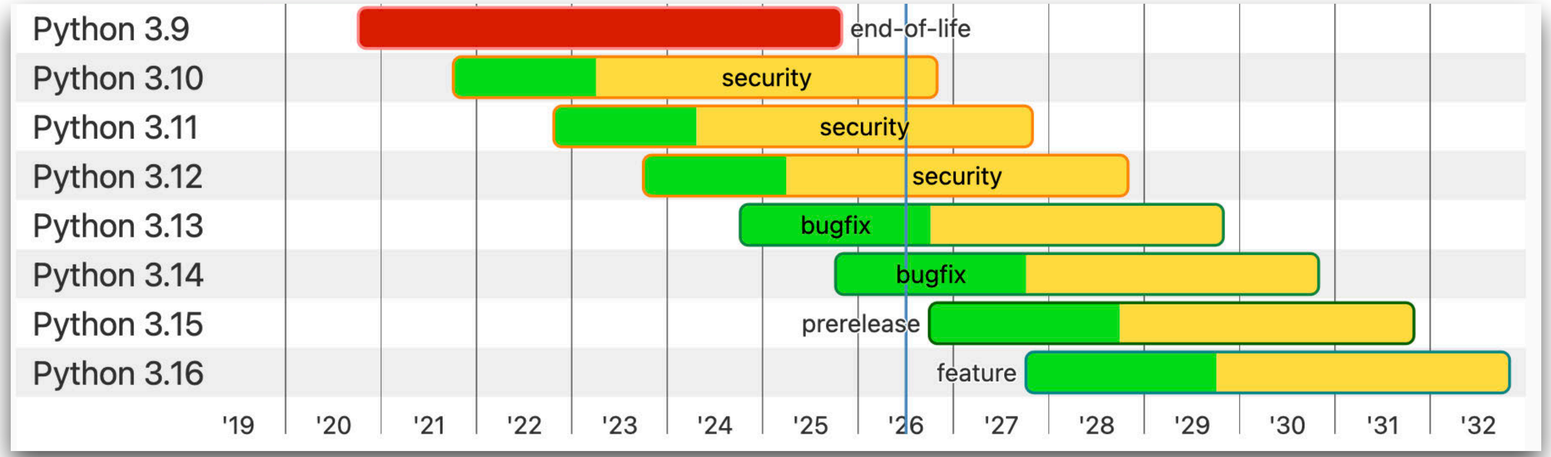
Introduction to Python

Part I - General Python Concepts

Python Versions

`Python` itself evolves over time and comes in **different versions** (for example `Python 3.10`, `Python 3.11`, `Python 3.12`, ...): newer versions bring new features and improvements, while older ones gradually stop being supported.

This means that not everything works with every version: some code may only run with a **specific** version of `Python`, while another project might need a **newer** one. It is therefore useful to be able to **choose which version of `Python`** to use for each project, and, as we will see, there are tools that make this very easy.



Introduction to Python

Part I - General Python Concepts

The Python Packages

On its own, `Python` can already do a lot: it can do **calculations**, **work with text**, **read and write different file types**, **repeat tasks**, and much more.

These capabilities are organized into **packages** (also referred to as **libraries** or **dependencies**). A **package** is simply a **collection of** `Python` **code** grouped together to provide a specific set of functionality.

Note

🍕 **Analogy**: To put the **mozzarella** (the **package**) on our pizza, we don't have to buy a cow, milk it, and make the cheese ourselves. We can simply get **mozzarella that someone else has already made beautifully** for us. Sure, we *could* try to make it from scratch (i.e. write the code ourselves), but it can be **quite hard** and time-consuming. **Packages** are exactly that: ready-made ingredients we can reuse instead of preparing everything from scratch.

Introduction to Python

Part I - General Python Concepts

The Python Packages

On its own, `Python` can already do a lot: it can do **calculations**, **work with text**, **read and write different file types**, **repeat tasks**, and much more.

These capabilities are organized into **packages** (also referred to as **libraries** or **dependencies**). A **package** is simply a **collection of** `Python` **code** grouped together to provide a specific set of functionality.

Example: the ***random*** package

example.py

```
1   import random
2
3   random.randint(1, 6)
4
```

how can I pick random numbers from a specified range?

Introduction to Python

Part I - General Python Concepts

The Python Packages

On its own, `Python` can already do a lot: it can do **calculations**, **work with text**, **read and write different file types**, **repeat tasks**, and much more.

These capabilities are organized into **packages** (also referred to as **libraries** or **dependencies**). A **package** is simply a **collection of** `Python` **code** grouped together to provide a specific set of functionality.

Standard Libraries (Packages)

- Already included and available by default (e.g. the *random* package)
- No installation required

example.py

```
1  import random
2
3  random.randint(1, 6)
4
```

Huge Community of Packages

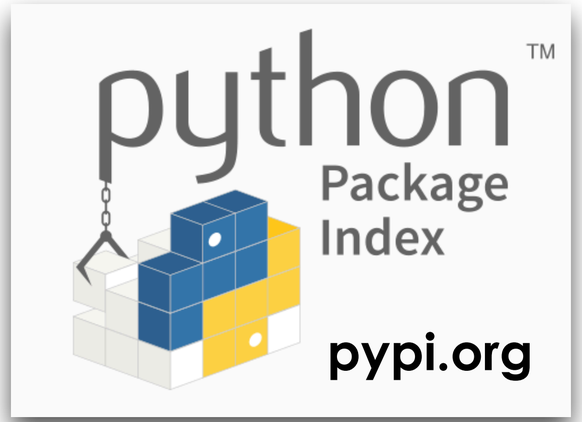
- the real power of Python
- have to be installed
- For example, in the course we will use:
  - **read images** (e.g. `tifffile`, `bffile`)
  - **manipulate** images and perform **numerical calculations** efficiently (e.g. `numpy`, `scikit-image`)
  - use **deep learning** models (e.g. `cellpose`, `spotiflow`)
  - create **plots** and **visualizations** (e.g. `matplotlib`, `ndv`)

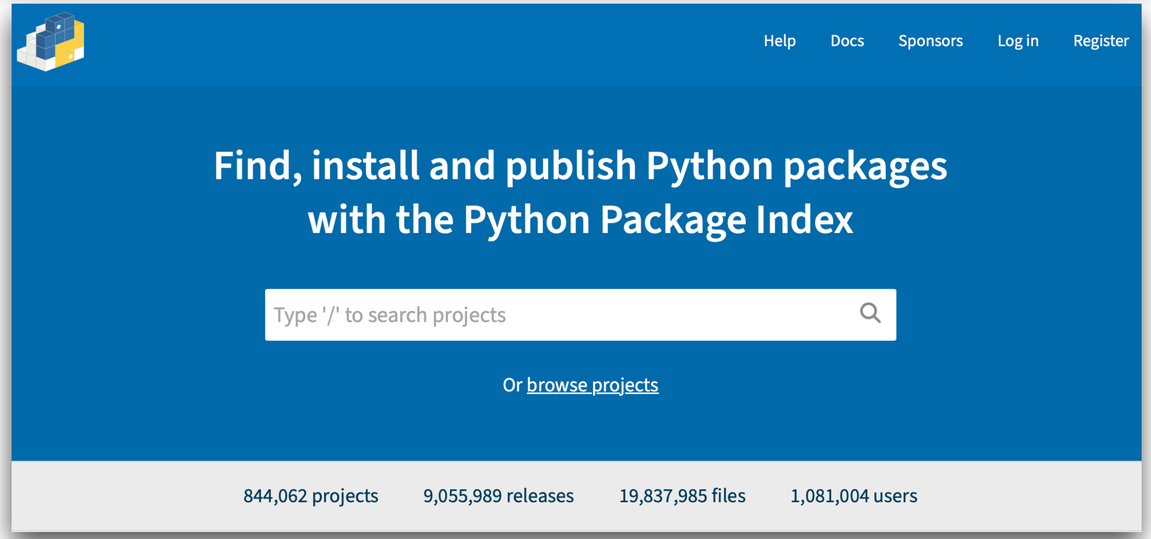
Introduction to Python

Part I - General Python Concepts

The Python Packages

> **Note**
>
> 📦 **Other repositories:** `PyPI` is probably the most common place packages come from, but it is not the only one. In **scientific programming** you might also come across conda-forge, which is another place where packages can be stored for download and installation.





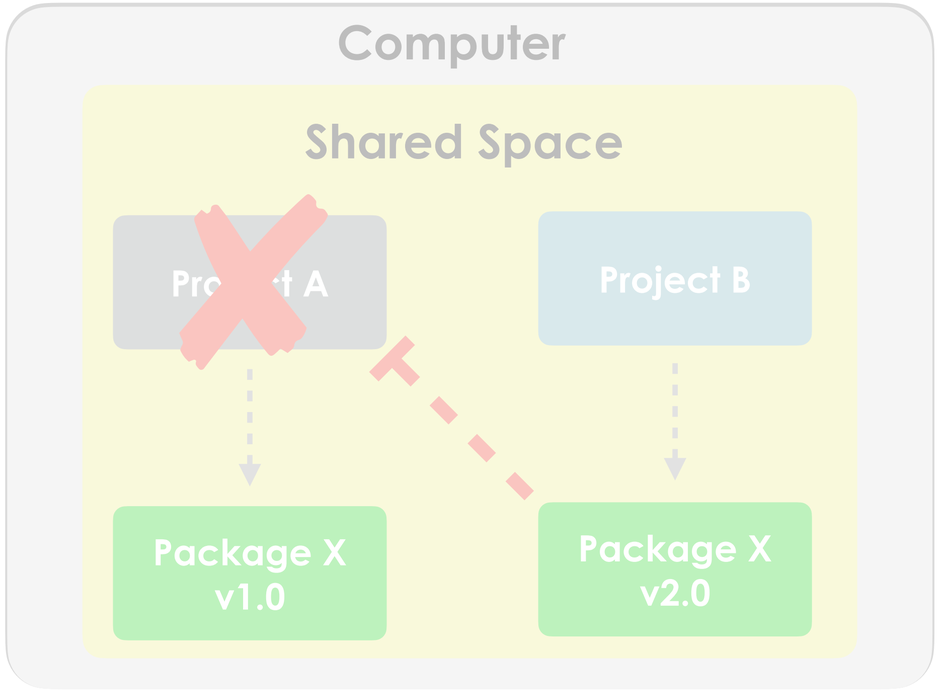
Introduction to Python

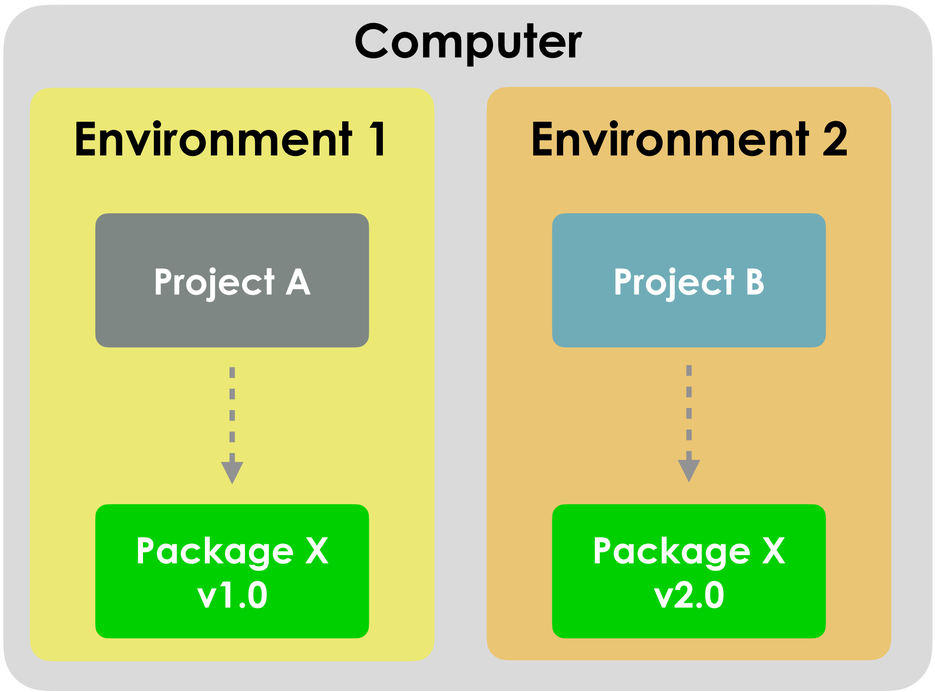
Part I - General Python Concepts

the virtual environment is a folder on your computer

The Virtual Environments

A **virtual environment** is an **isolated workspace** (imagine a folder on your computer) that contains its **own packages** and its **own** `Python`, completely **separate** from everything else on your computer. Each project gets its **own environment**, so the packages of one project can **never interfere** with the packages of another.





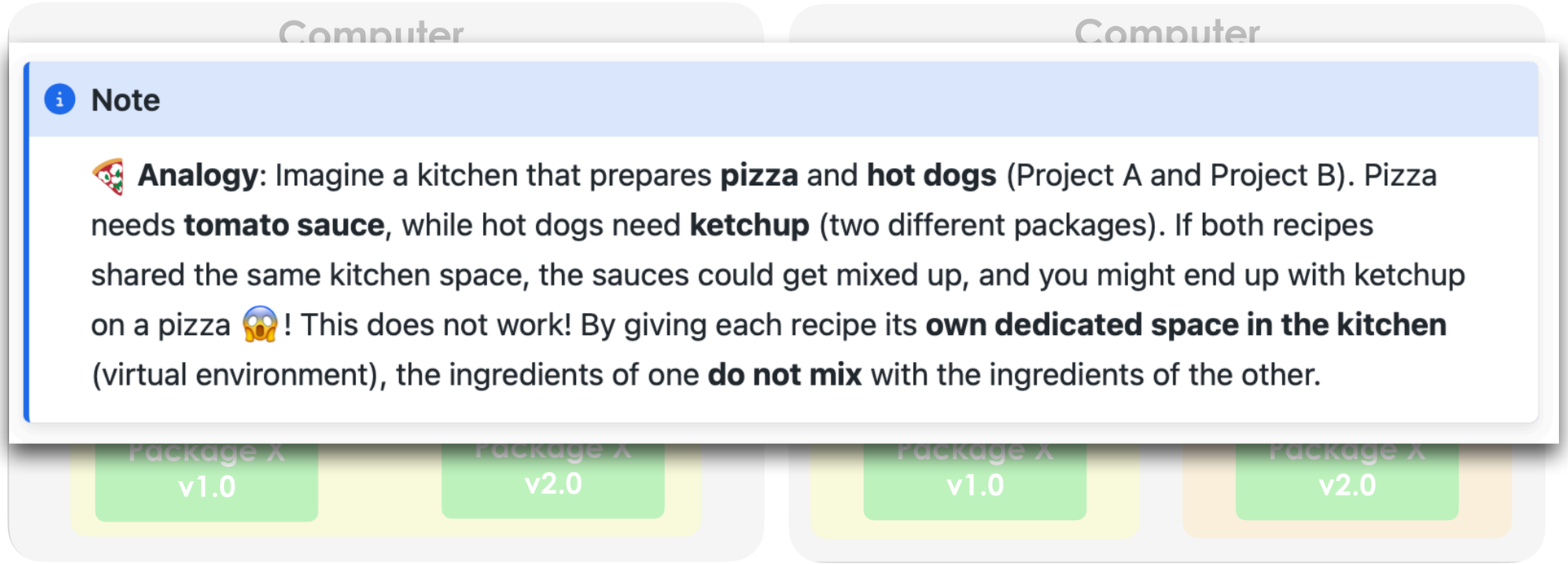
Introduction to Python

Part I - General Python Concepts

the virtual environment is a folder on your computer

The Virtual Environments

A **virtual environment** is an **isolated workspace** (imagine a folder on your computer) that contains its **own packages** and its **own** `Python`, completely **separate** from everything else on your computer. Each project gets its **own environment**, so the packages of one project can **never interfere** with the packages of another.



Note

🍕 **Analogy**: Imagine a kitchen that prepares **pizza** and **hot dogs** (Project A and Project B). Pizza needs **tomato sauce**, while hot dogs need **ketchup** (two different packages). If both recipes shared the same kitchen space, the sauces could get mixed up, and you might end up with ketchup on a pizza 😱! This does not work! By giving each recipe its **own dedicated space in the kitchen** (virtual environment), the ingredients of one **do not mix** with the ingredients of the other.

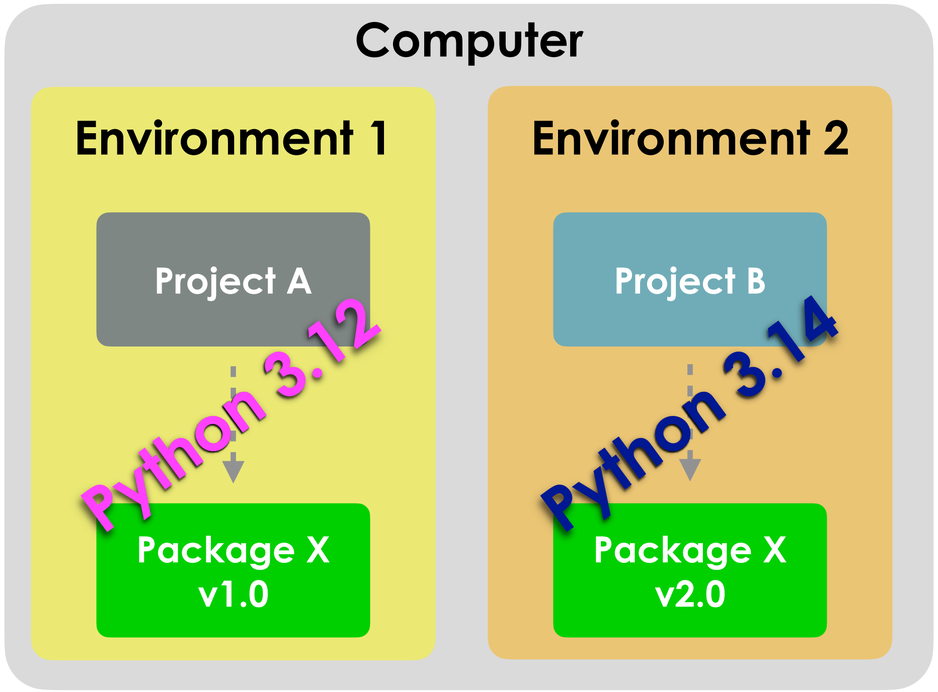
Introduction to Python

Part I - General Python Concepts

the virtual environment is a folder on your computer

The Virtual Environments

Since each environment contains its **own** `Python`, virtual environments also solve the **version** problem we saw earlier: every project can use exactly the **version of** `Python` it needs (e.g. Project A can use `Python 3.10` while Project B uses `Python 3.12`), again **without any conflict**.



Introduction to Python

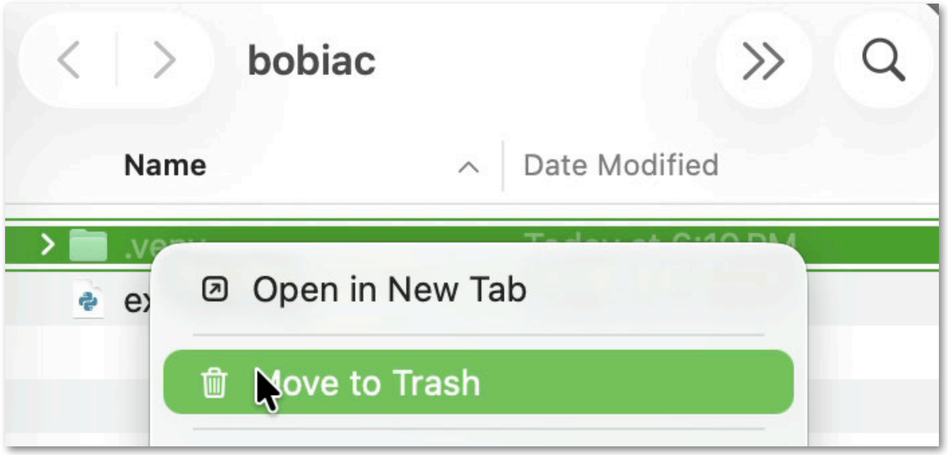
Part I - General Python Concepts

the virtual environment is a folder on your computer

The Virtual Environments

!! DISPOSABLE !!

A key idea is that virtual environments are **disposable**. An environment is **not precious**: if something goes wrong, or you no longer need a project, you can simply **delete the environment and recreate it from scratch**, without affecting `Python` itself or any of your other projects. This makes experimenting **safe**, you can always throw an environment away and start fresh.

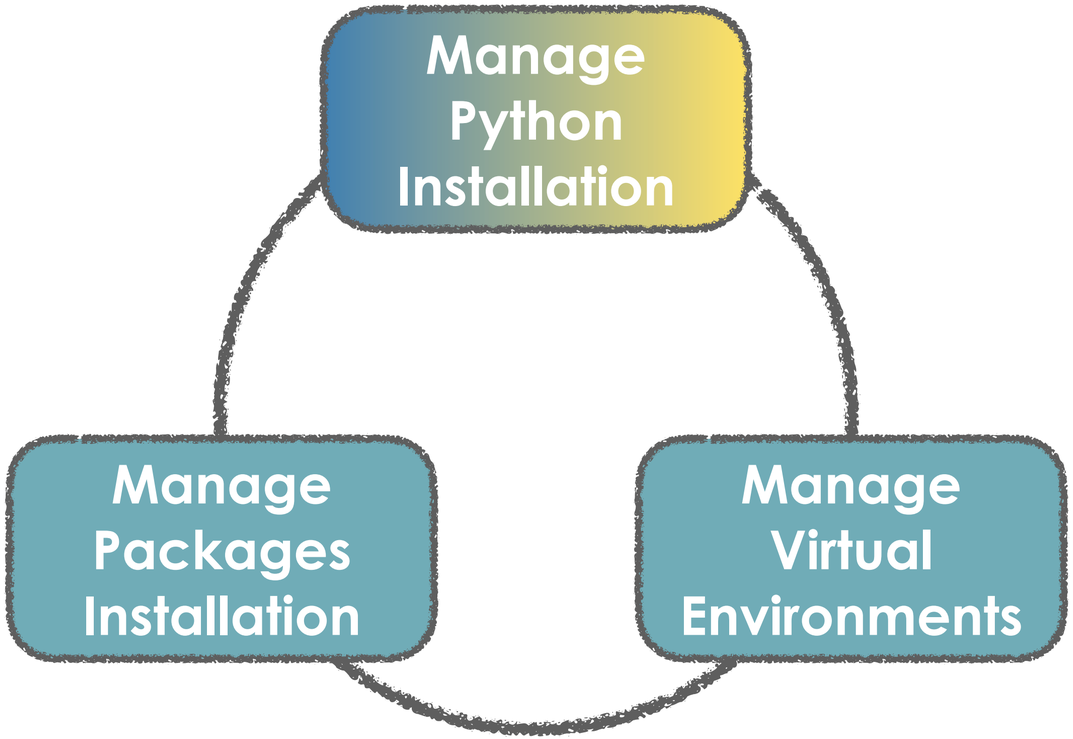


Introduction to Python

Part I - General Python Concepts

Summary

To work with Python we need:



Introduction to Python

Part I - General Python Concepts

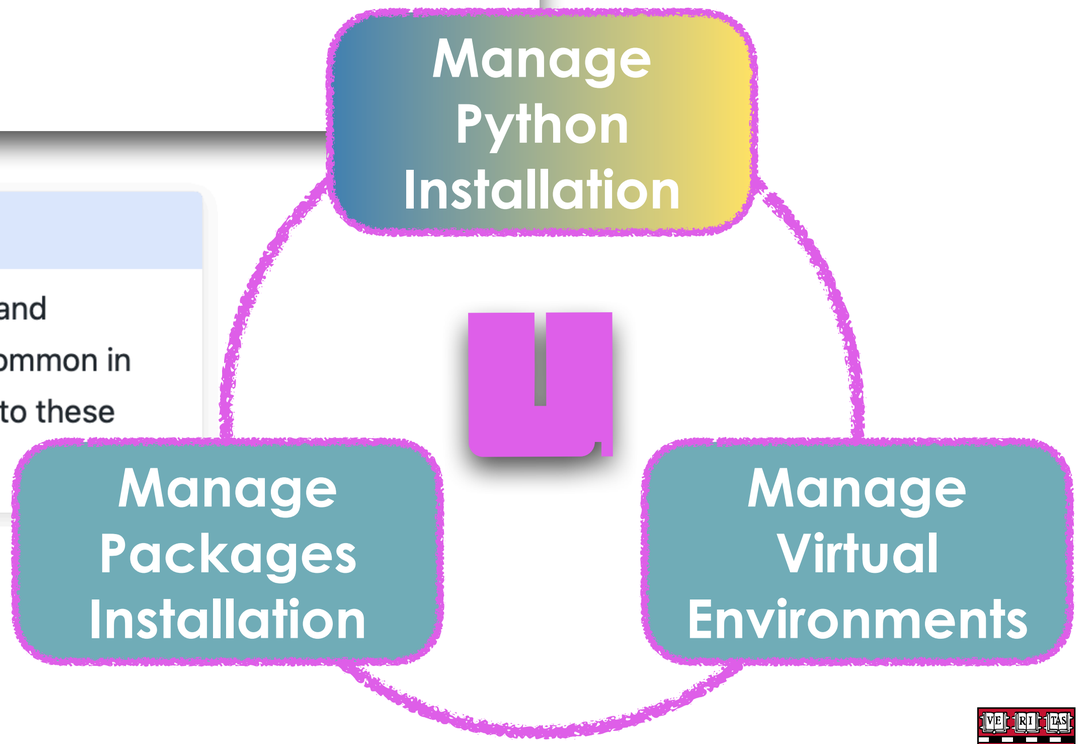
UV

`uv` is a **modern, extremely fast tool** that takes care of **everything** we just described. Instead of learning many different tools, you only have to learn **one**. With `uv` you can:

- **download and install any version of** `Python` you need,
- **install and manage packages** in your project,
- **create and manage virtual environments** automatically.

Note

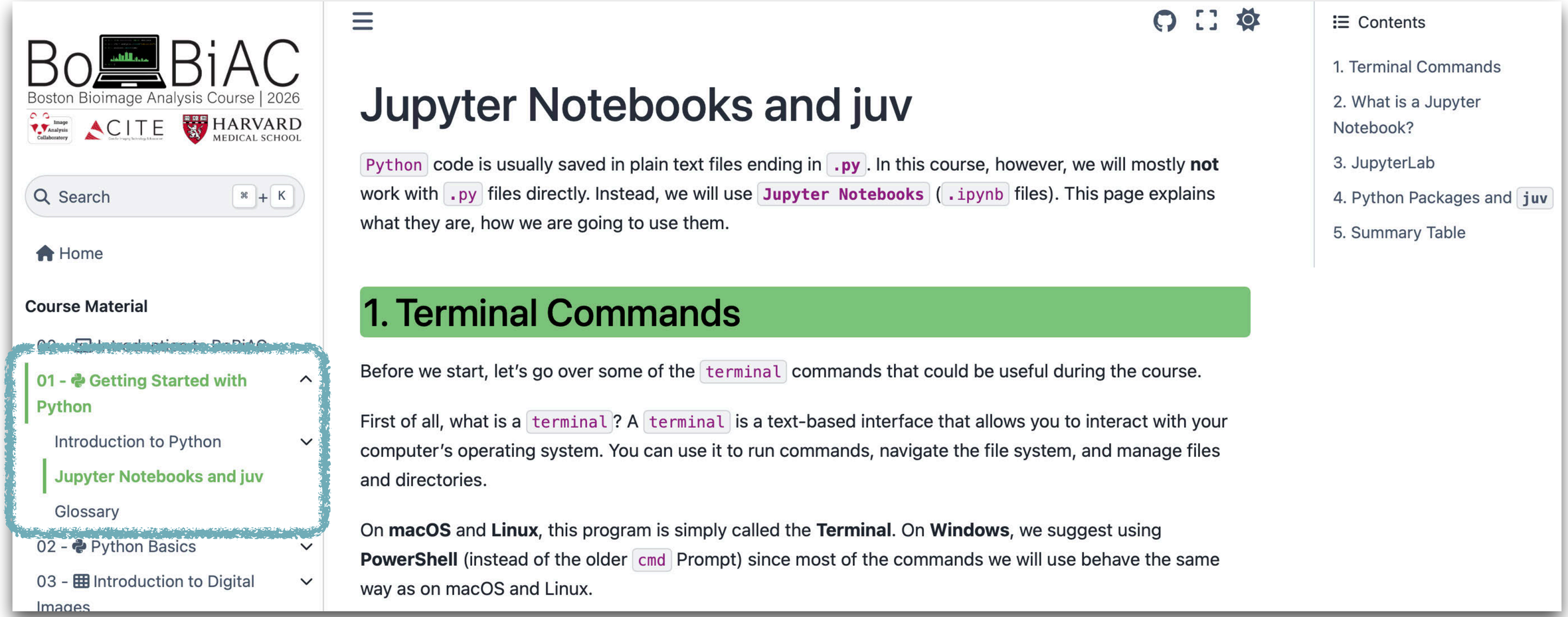
🔧 **Other tools**: `uv` is not the only tool that can manage packages, virtual environments, and `Python` versions together. Other popular options include conda and pixi, which are also common in scientific computing. In this course we will use `uv`, but the concepts you learn here apply to these other tools as well.



docs.astral.sh/uv

Introduction to Python

Part II - Jupyter Notebooks and *juv*



Introduction to Python

Part II - Jupyter Notebooks and *juv*

Jupyter Notebooks

Running full `.py` files is great once you are familiar with `Python` and you are comfortable with the workflow, but for **learning** and **experimenting** it's a bit clunky: you have to write the whole file, save it, run it, look at the output, then go back and edit.

In this course we will use `Jupyter Notebooks` and its particular file type with the extension `.ipynb` (short for **i**nteractive **py**thon **n**ote**b**ook). For example, a file called `bobiac.ipynb`.

`Jupyter Notebooks` allow us to:

- Write and run `Python` code in **small chunks** (called *cells*) and execute them one at a time.
- Display the output of each cell **immediately** below the cell, so you can see the results of your code as you go.
- Mix `Python` **code** with **text**, **images**, and **equations** in the same document, which is great for **learning** purposes.

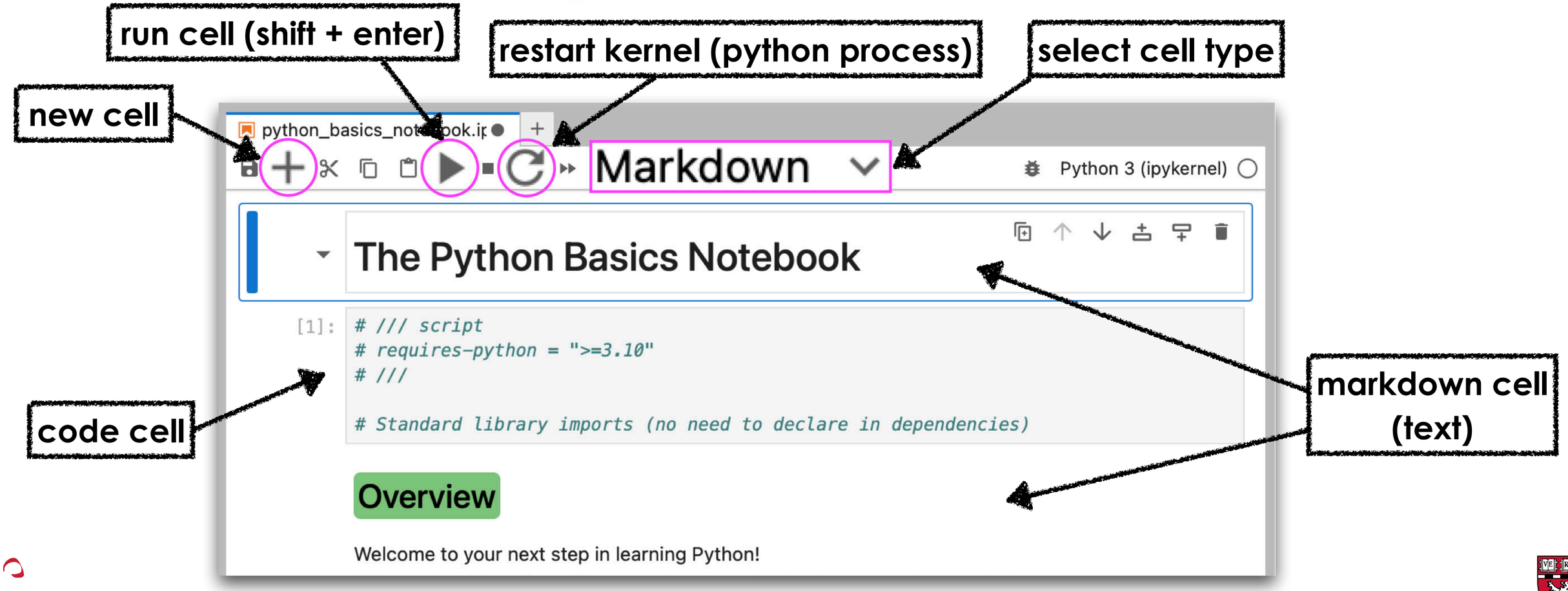
jupyter.org

Introduction to Python

Part II - Jupyter Notebooks and *juv*

Jupyter Notebooks

JupyterLab
jupyterlab.readthedocs.io



jupyter.org

Introduction to Python

Part II - Jupyter Notebooks and *juv*

Jupyter Notebooks

As we saw in the Python introduction, running `Python` code requires three things: a **virtual environment**, the **packages** (aka **dependencies** or **libraries**) your code imports, and the `Python` **version** you want to use.

juv by Trevor Manz

To handle all of this automatically, we use `juv`, a tool built on top of `uv` specifically designed for `Jupyter Notebooks` (check out the Getting Started with uv to learn more about `uv`). Instead of manually creating environments and installing packages, `juv` reads a dependencies list declared directly inside the notebook file and sets everything up for you in one command (through `PEP 723`, see below).

Introduction to Python

Part II - Jupyter Notebooks and *juv*

PEP 723

A PEP (Python Enhancement Proposal) is a document that describes a new feature or convention for the `Python` language. PEP 723 is one of these proposals, and it defines a standard way to declare a script's dependencies **directly inside the file itself**, using a special comment block at the top of the file:

```
# /// script
# requires-python = ">=3.12"
# dependencies = [
#     "numpy",
#     "matplotlib",
# ]
# ///
```

It works with both ***.py*** ( ) and ***.ipynb*** ( ) files!!!

Introduction to Python

Part II - Jupyter Notebooks and *juv*

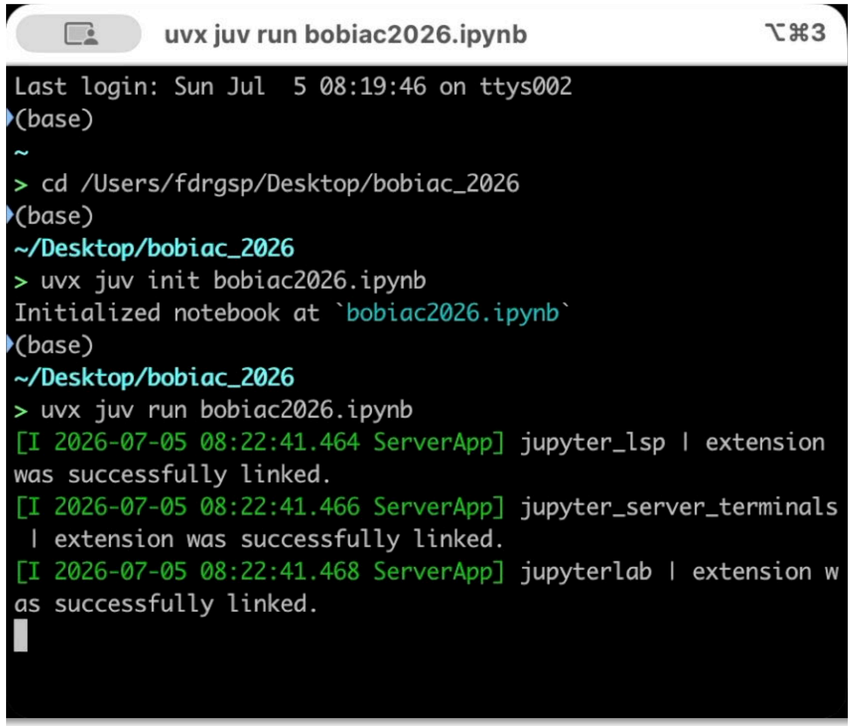
juv by Trevor Manz

```
# create a new notebook (and specify python version)
uvx juv init <name.ipynb>
uvx juv init -p <PYTHON> <name.ipynb>

# add packages
uvx juv add <name.ipynb> <PACKAGE>

# run the notebook
uvx juv run <name.ipynb>
```

the terminal



*uvx = uv tool run

github.com/manzt/juv

Introduction to Python

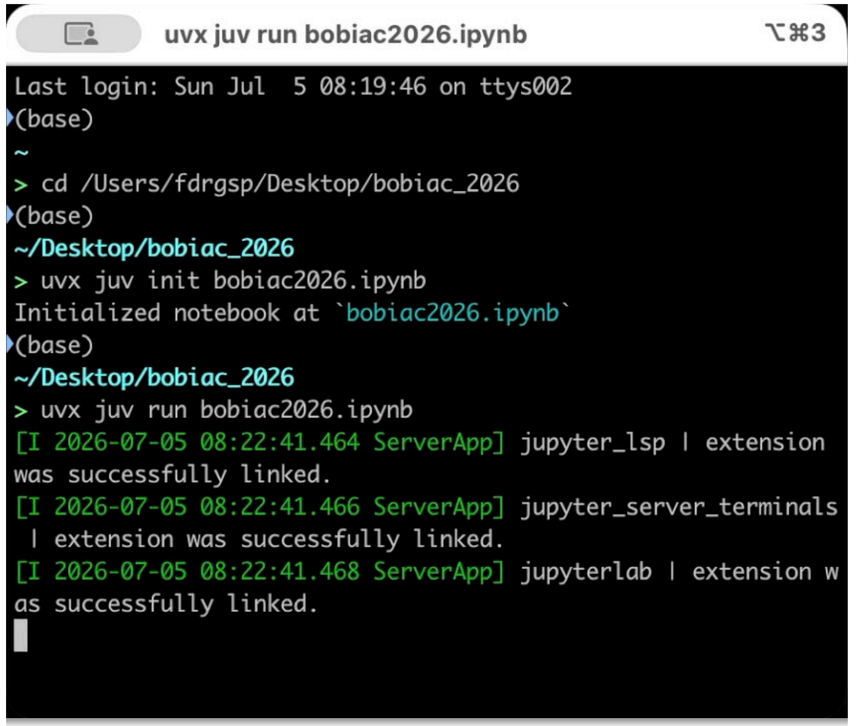
Part II - Jupyter Notebooks and *juv*

juv by Trevor Manz

Each day, you will download the Jupyter Notebook for that day's topic and launch it with:

```
uvx juv run <name.ipynb>
```

the terminal



github.com/manzt/juv

Introduction to Python

Part II - Jupyter Notebooks and *juv*

Command	Description
`pwd`	Print the path of the current working directory.
`ls`	List the contents of the current directory.
`cd [DIR]`	Change directory to `[DIR]`. Example: `cd Desktop/bobiac` *If `[DIR]` is not provided, defaults to the home directory.*
`mkdir [DIR]`	Create a new directory named `[DIR]`. Example: `mkdir Desktop/bobiac`
`uvx juv init <name.ipynb>`	Create a new `Jupyter Notebook` with an empty `///script` header. Example: `uvx juv init my_notebook.ipynb`
`uvx juv init -p <PYTHON> <name.ipynb>`	Create a new `Jupyter Notebook` with a specific `Python` version (e.g. `-p 3.12`). Example: `uvx juv init -p 3.13 my_notebook.ipynb`
`uvx juv add <name.ipynb> <PACKAGE>`	Add `PACKAGE` to the notebook's dependency list. Example: `uvx juv add my_notebook.ipynb numpy`
`uvx juv run <name.ipynb>`	Launch the notebook in JupyterLab, installing all dependencies automatically. Example: `uvx juv run my_notebook.ipynb`

juv by Trevor Manz

github.com/manzt/juv

Introduction to Python

Part II - Jupyter Notebooks and *juv*

BoBiAC
Boston Bioimage Analysis Course | 2026

Search ⌘ + K

Home

Course Material

- 00 - Introduction to BoBiAC
- 01 - Getting Started with Python
- 02 - Python Basics
- 03 - Introduction to Digital Images
- 04 - Image Segmentation
- 05 - Spot Detection
- 06 - Object Classification
- 07 - Measurements & Quantification
- 08 - Colocalization

BoBiAC Exercises

BoBiAC Exercises

BoBiAC Exercises

Let's Try!!!

Course Exercises

Here is a list of exercises that you can do to practice the concepts covered in the book.

1. uv Exercises
2. juv Exercises
3. Group Work 1: Building your Own Segmentation and Spot Detection Pipeline
4. Group Work 2: Analysing Oscillations in Time-Series Images
5. Group Work 3: Does a Protein Localize to a Cellular Compartment?

Extra Practice

1. Working with Bioimages
2. Classic Segmentation

Previous
Notebook 2: Do two different protein clusters co-localize?

juv by Trevor Manz

github.com/manzt/juv

Introduction to Python

Part III - Python and *uv*