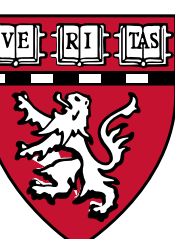
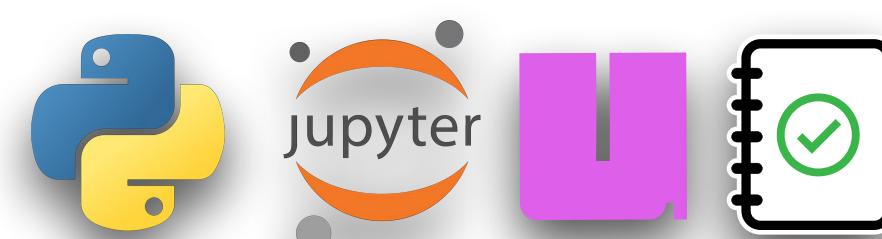


Dr Federico Gasparoli

A brief overview of key Python concepts you need to know to get started

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

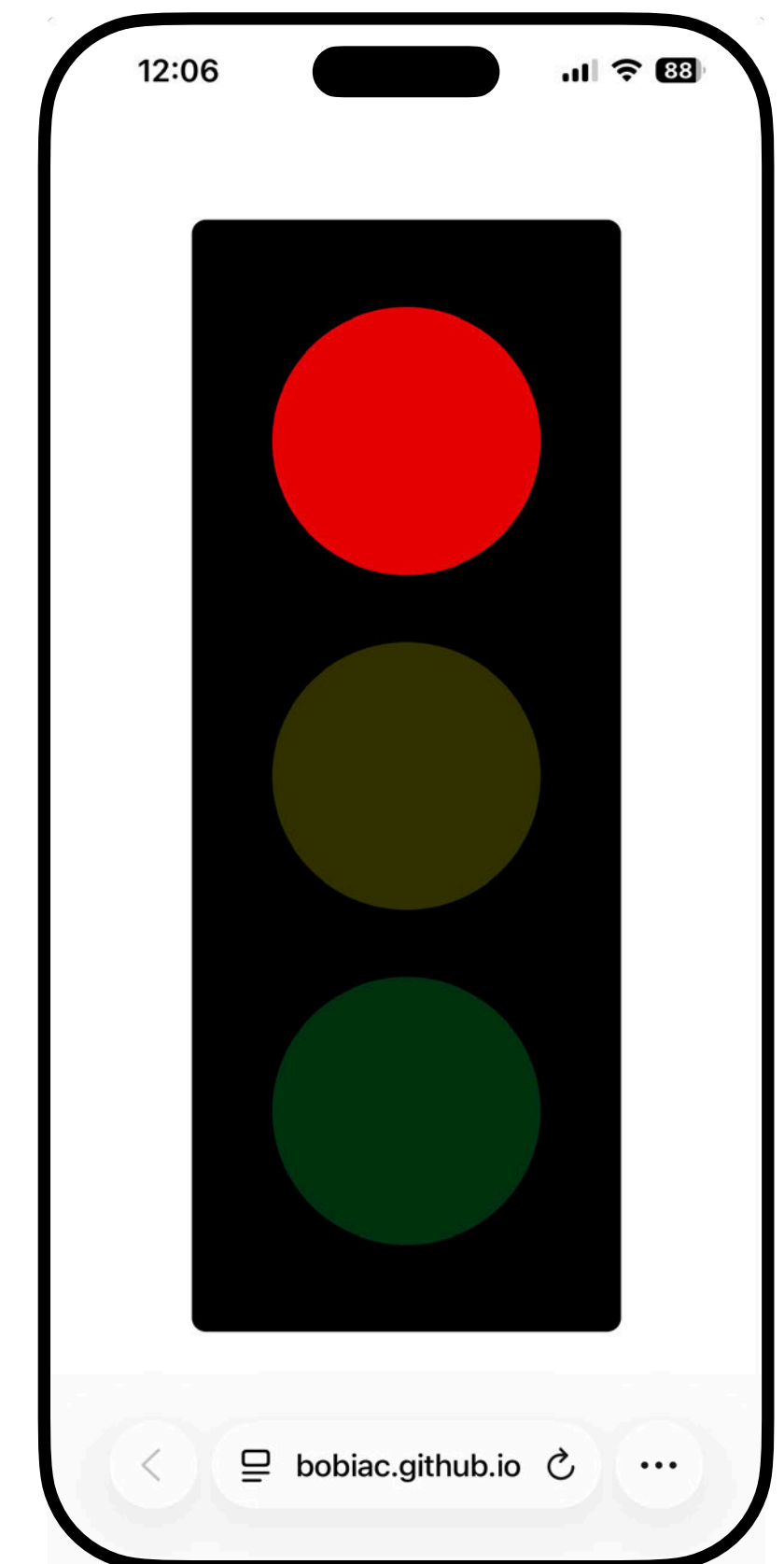
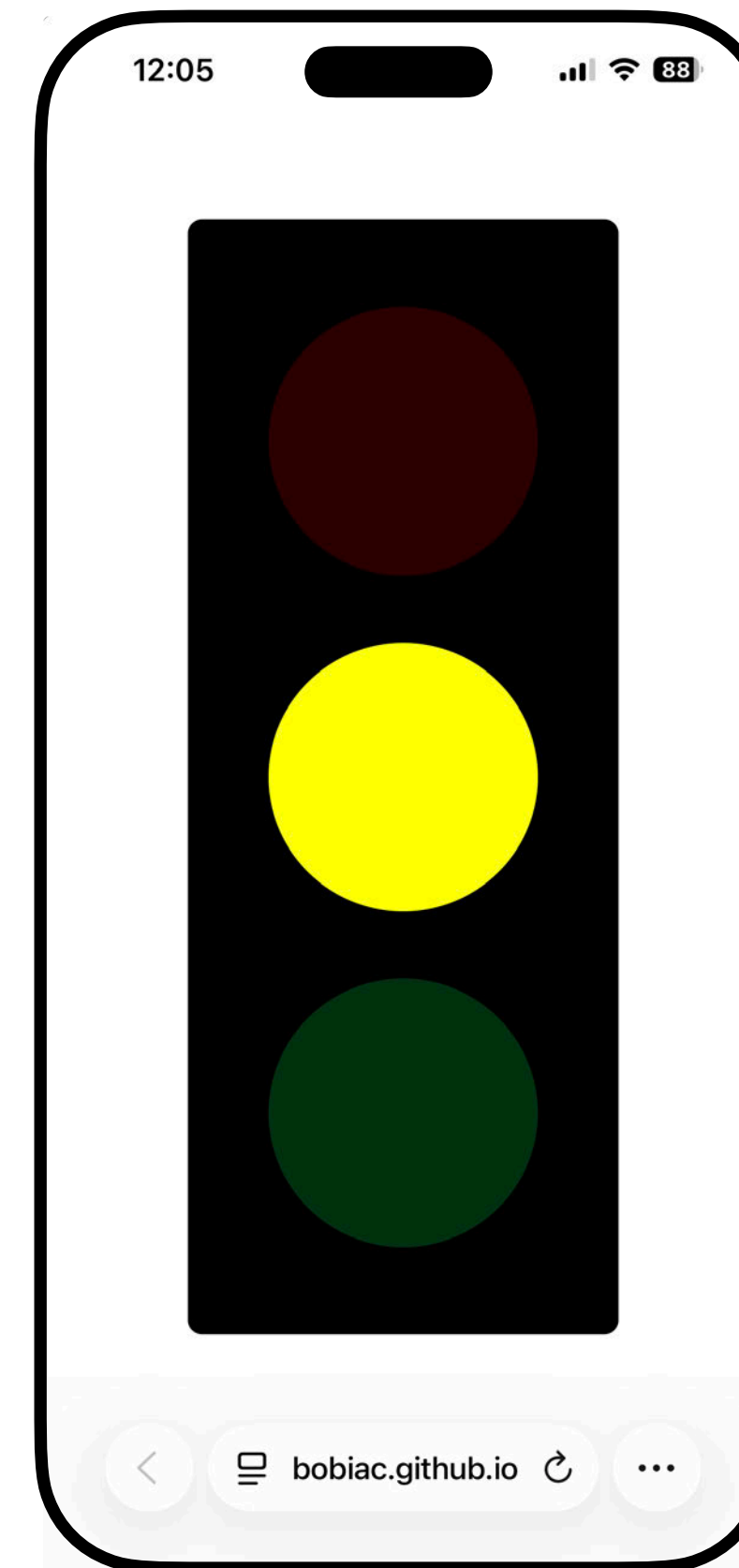
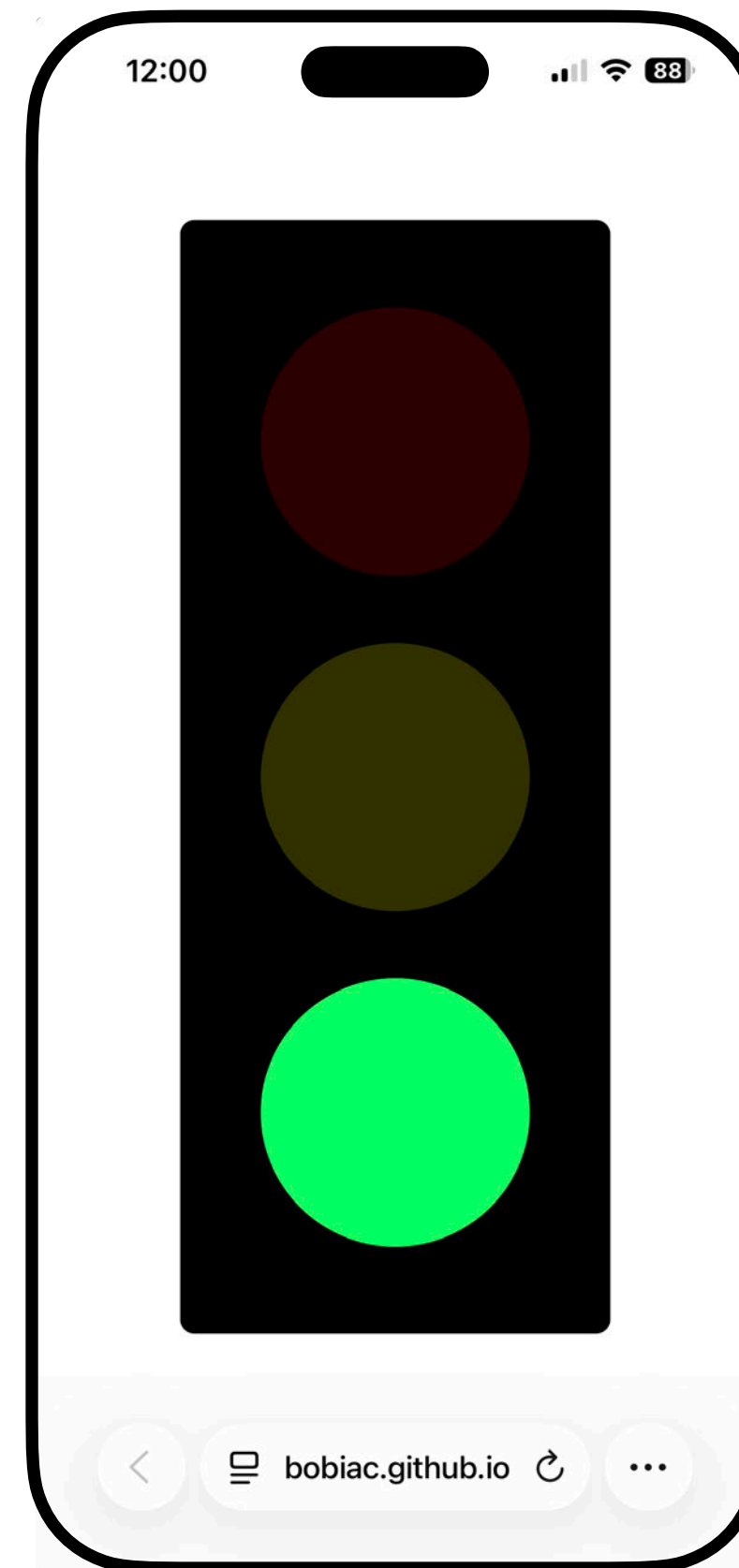
This is an opinionated introduction!

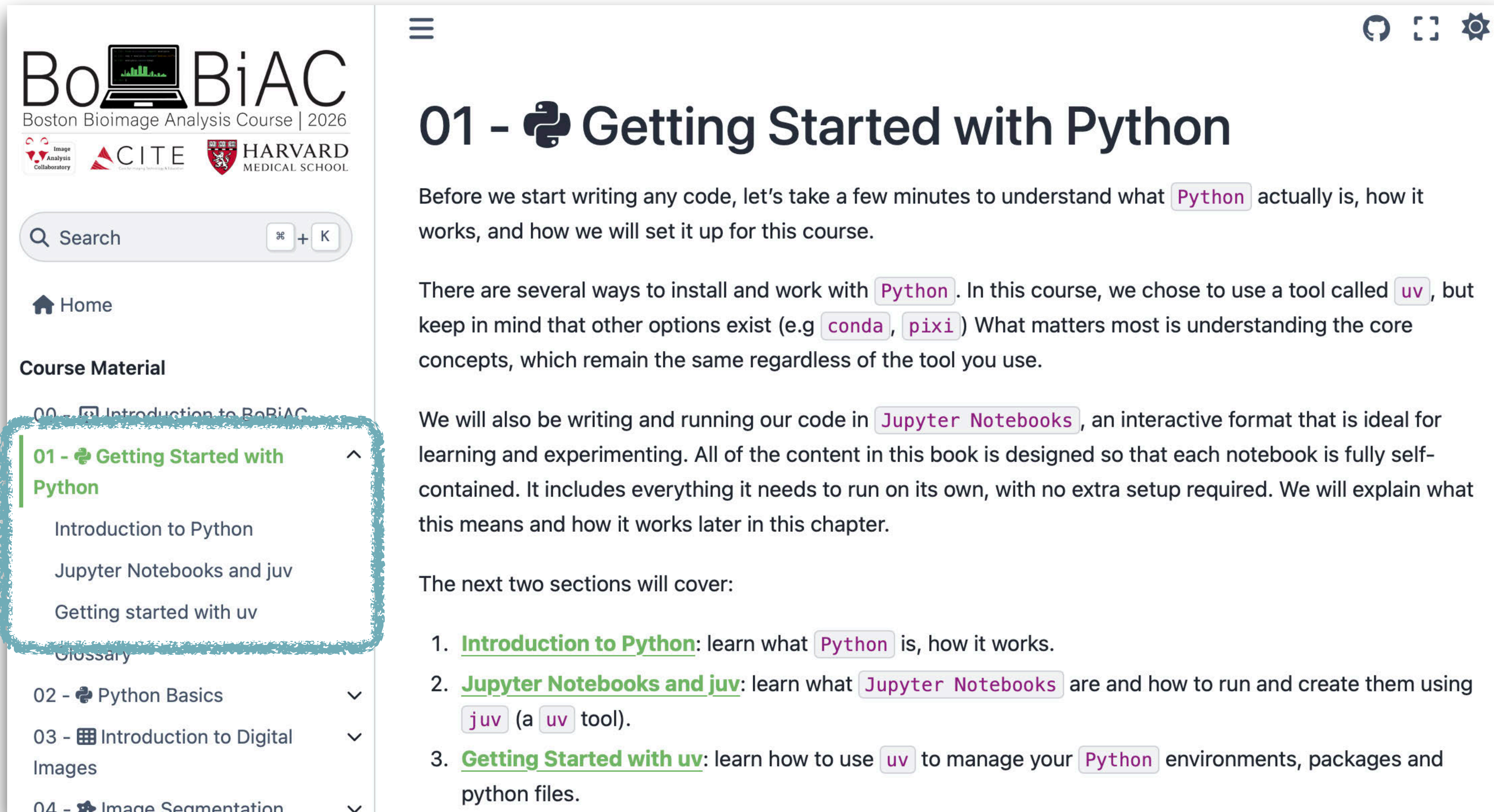




bobiac.github.io/classlight/bobiac2026

Realtime Feedback!





BoBiAC
Boston Bioimage Analysis Course | 2026

Image Analysis Collaboratory CITE HARVARD MEDICAL SCHOOL

Search [+ K]

Home

Course Material

- 00 - Introduction to BoBiAC
- 01 - Getting Started with Python**
 - Introduction to Python
 - Jupyter Notebooks and juv
 - Getting started with uv
- Glossary
- 02 - Python Basics
- 03 - Introduction to Digital Images
- 04 - Image Segmentation

01 - Getting Started with Python

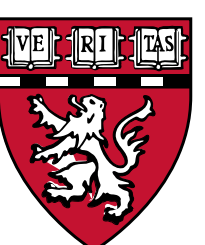
Before we start writing any code, let's take a few minutes to understand what `Python` actually is, how it works, and how we will set it up for this course.

There are several ways to install and work with `Python`. In this course, we chose to use a tool called `uv`, but keep in mind that other options exist (e.g. `conda`, `pixi`) What matters most is understanding the core concepts, which remain the same regardless of the tool you use.

We will also be writing and running our code in `Jupyter Notebooks`, an interactive format that is ideal for learning and experimenting. All of the content in this book is designed so that each notebook is fully self-contained. It includes everything it needs to run on its own, with no extra setup required. We will explain what this means and how it works later in this chapter.

The next two sections will cover:

- Introduction to Python**: learn what `Python` is, how it works.
- Jupyter Notebooks and juv**: learn what `Jupyter Notebooks` are and how to run and create them using `juv` (a `uv` tool).
- Getting Started with uv**: learn how to use `uv` to manage your `Python` environments, packages and python files.



Introduction to Python - Overview



Part I - General Python Concepts

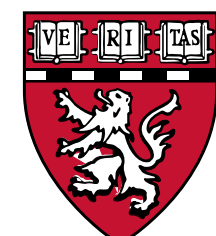
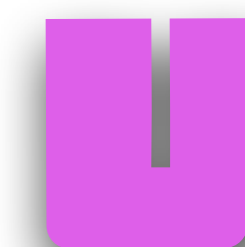
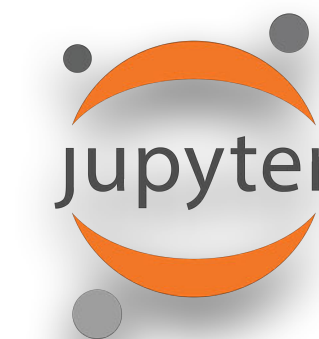
- ▶ What is Python?
- ▶ What are Python packages?
- ▶ What are virtual environments? Why are they useful?

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?




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

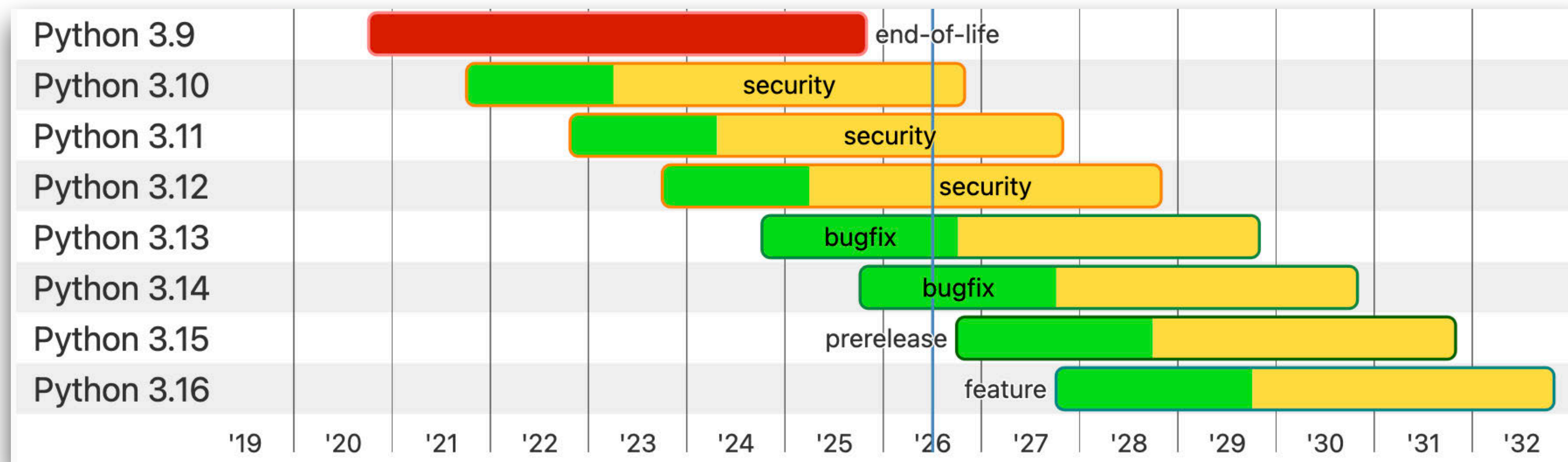


Part I - General Python Concepts

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.




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.




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?





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`)

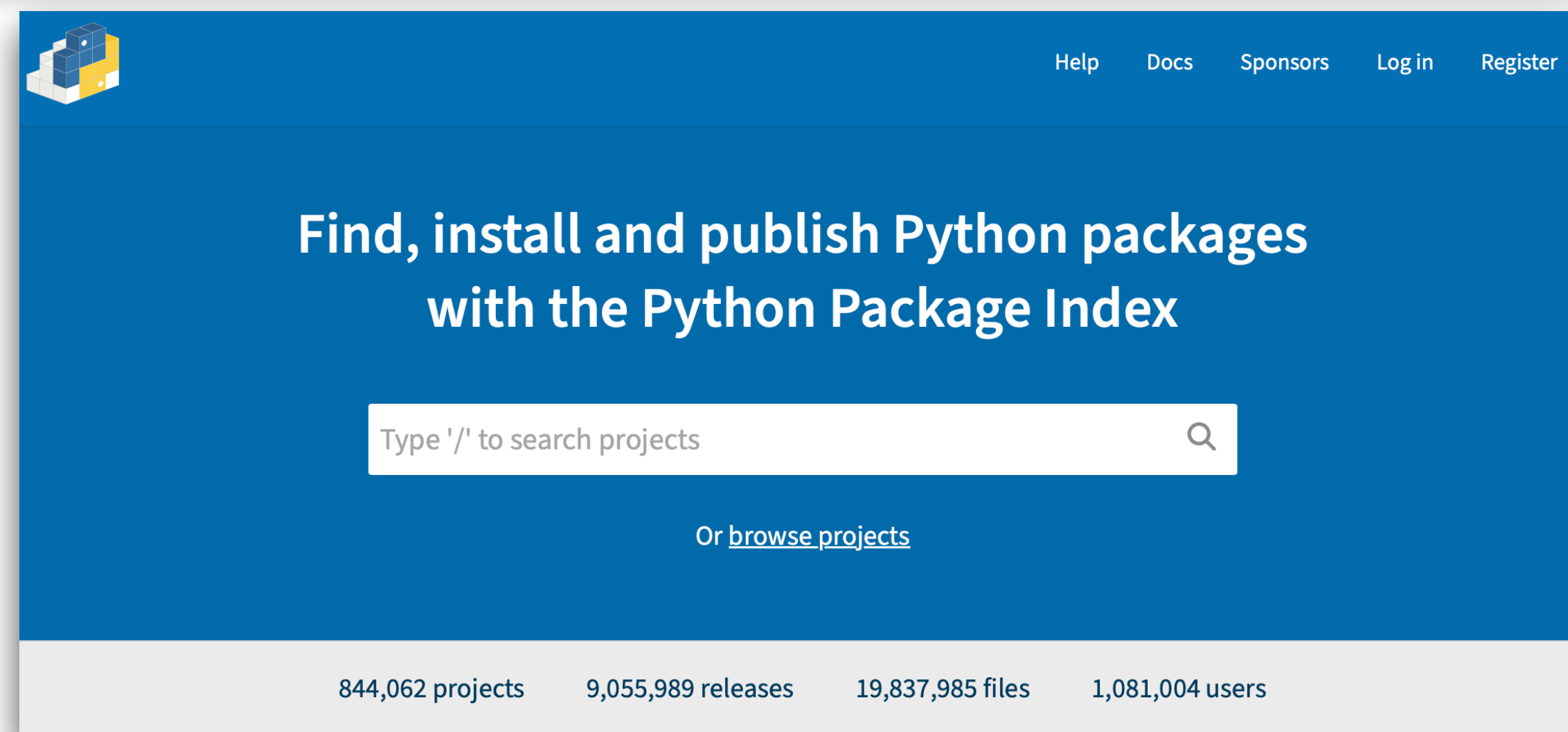
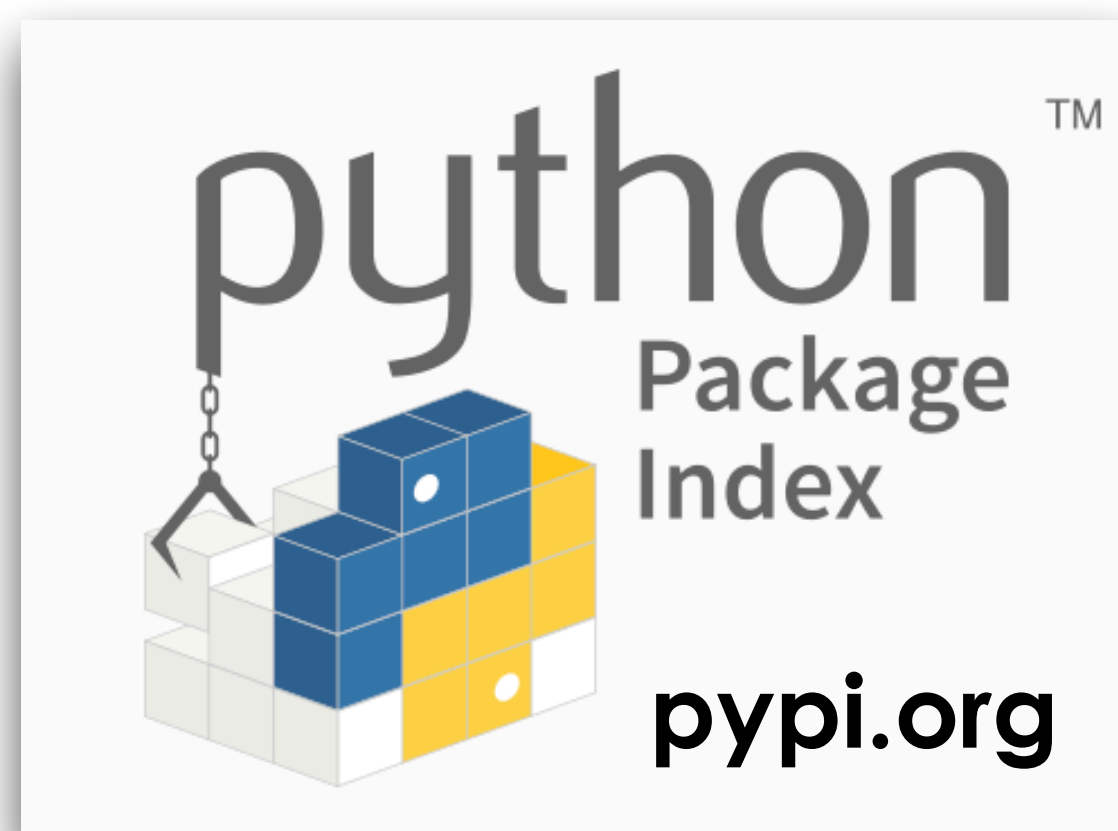




The Python Packages

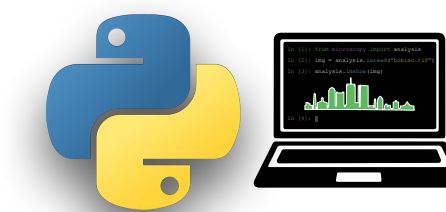
Note

 **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

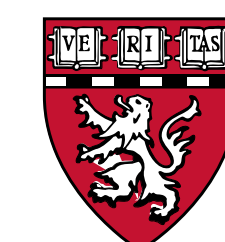
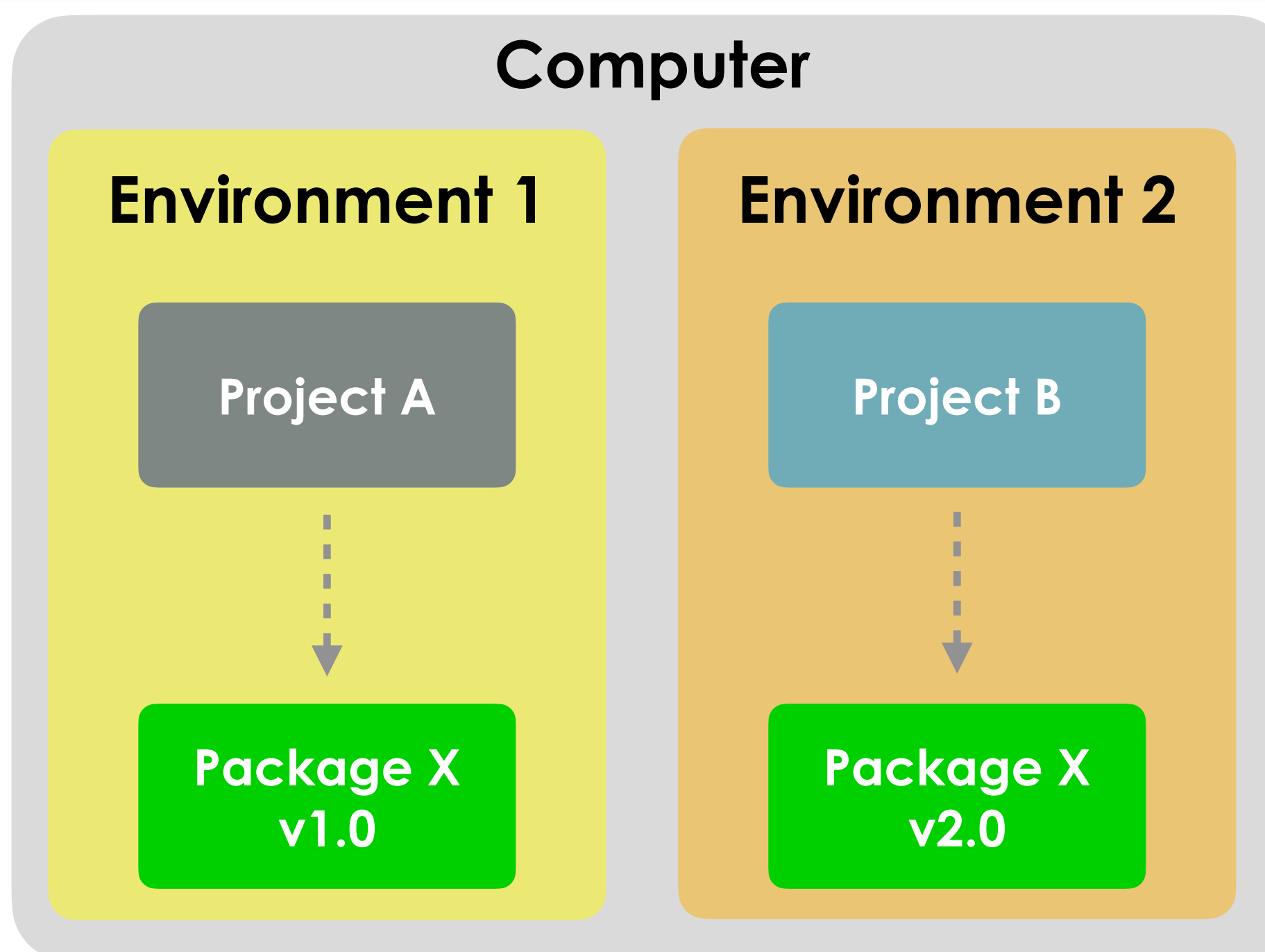
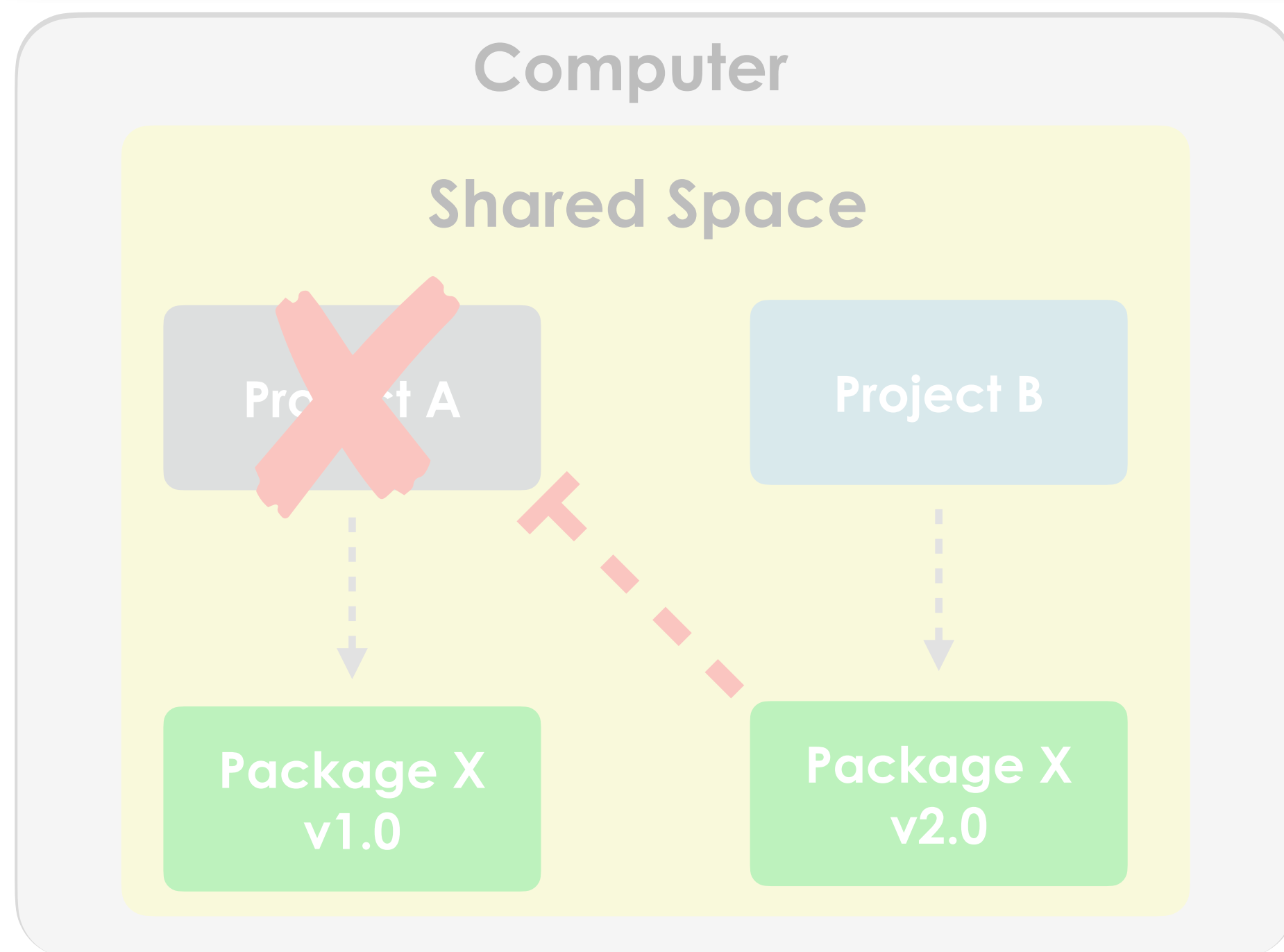
Part I - General Python Concepts



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.

Computer

Computer

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.

Package X
v1.0

Package X
v2.0

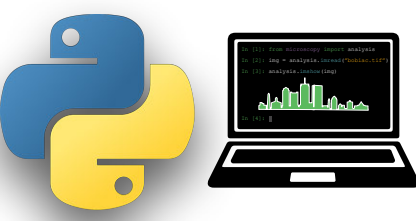
Package X
v1.0

Package X
v2.0



Introduction to Python

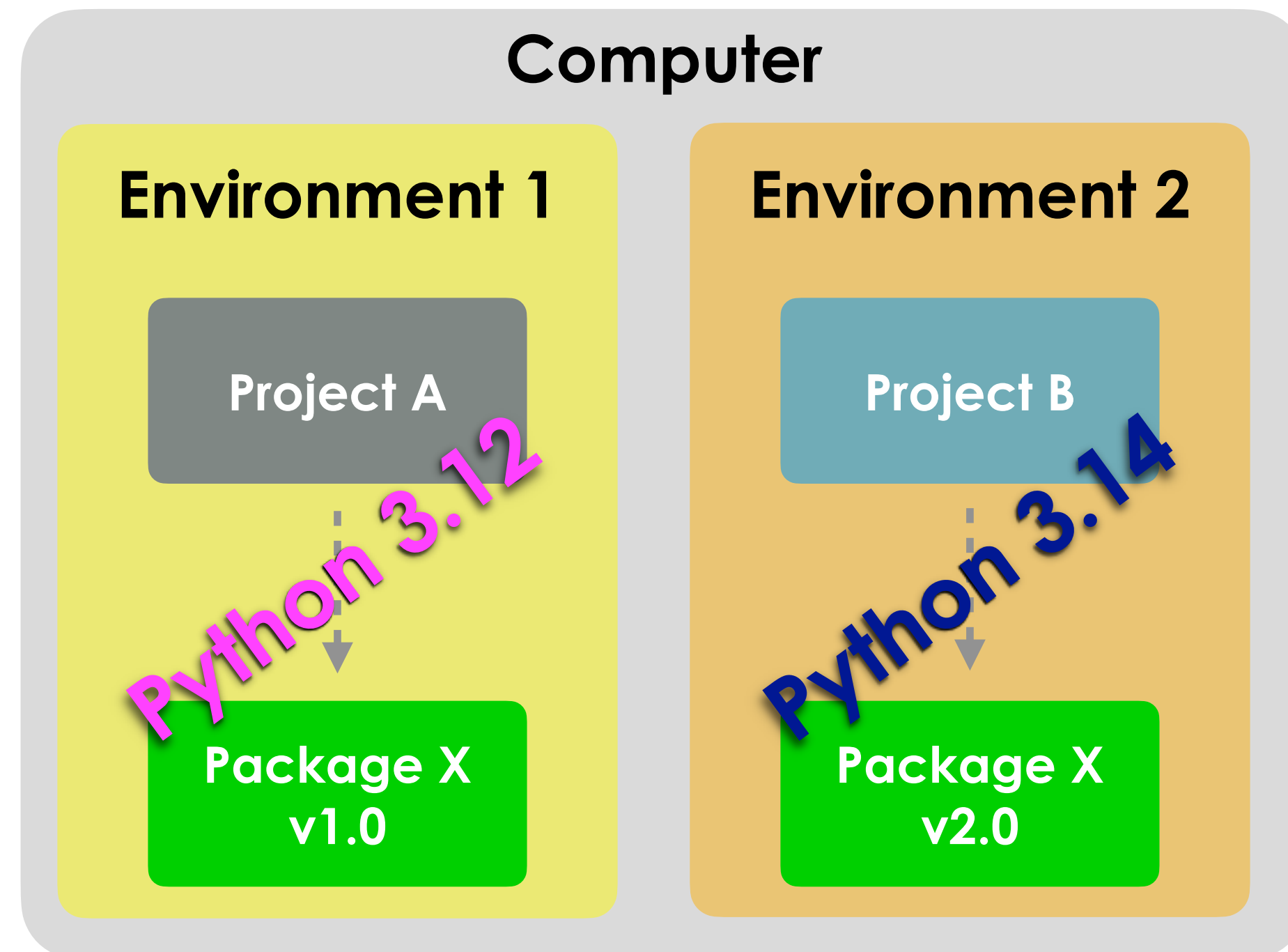
Part I - General Python Concepts



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

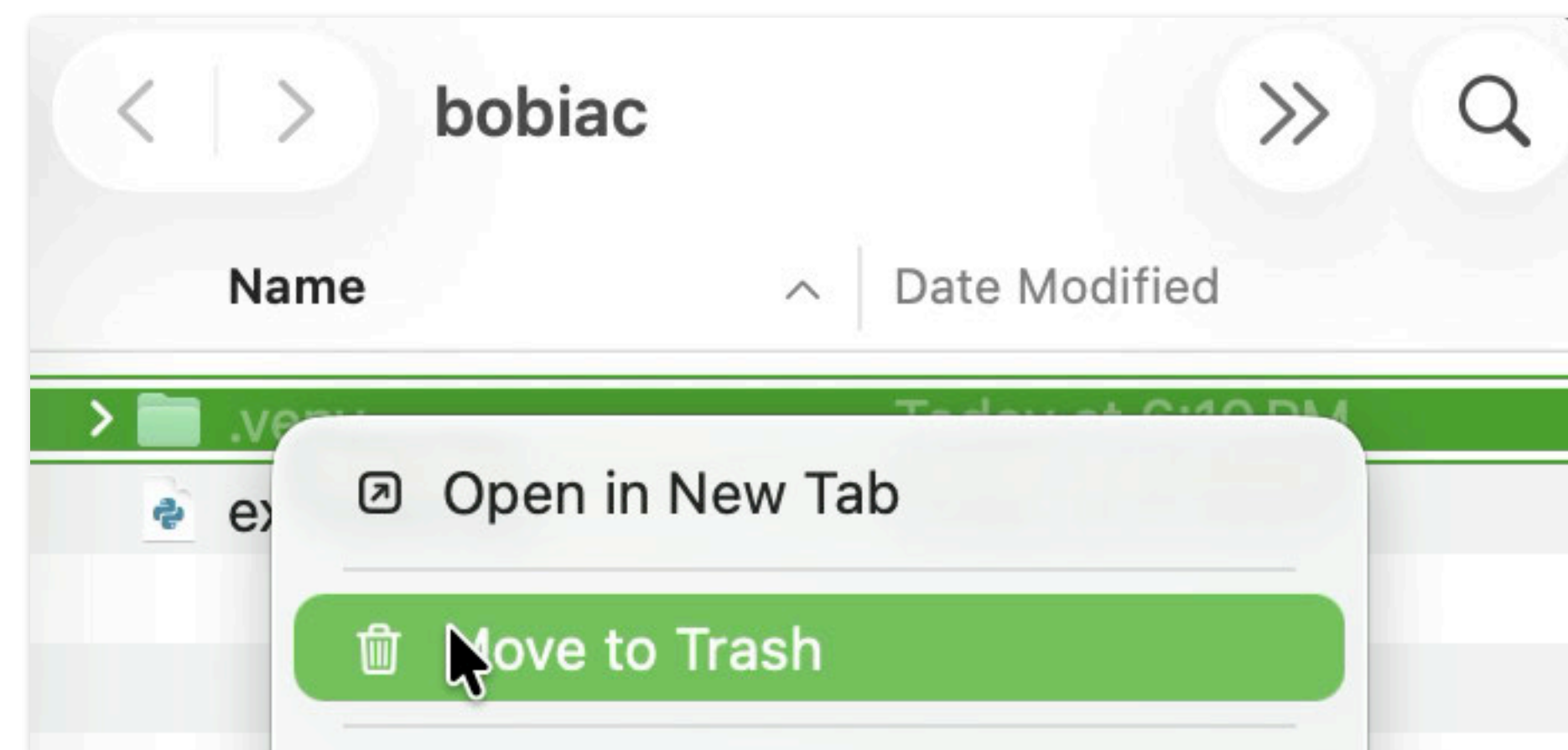


the virtual environment is a folder on your computer

The Virtual Environments

!! DISPOSABLE !!

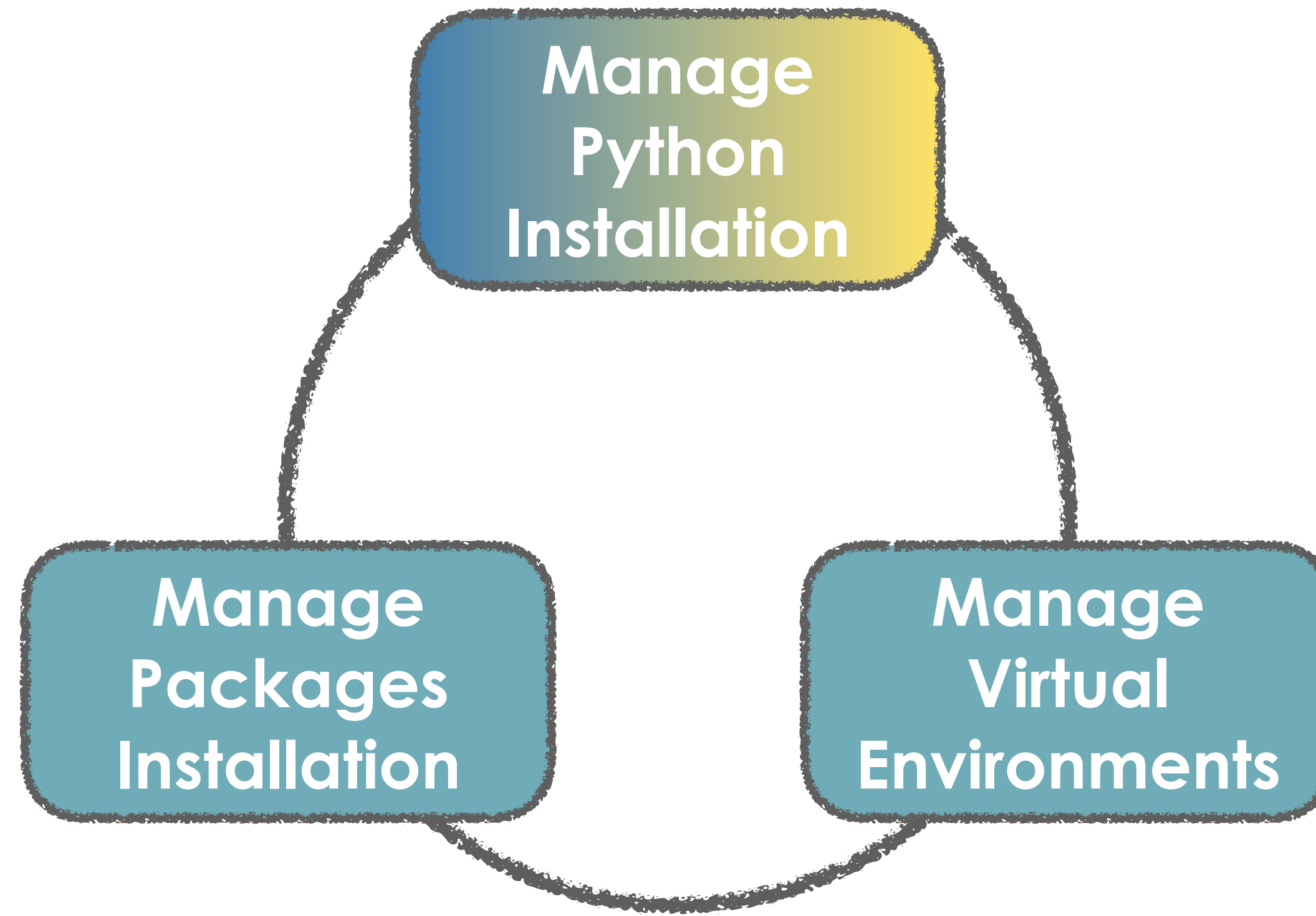
A key idea is that virtual environments are **disposable**. An environment is **not precious**: if something goes wrong, or you no longer need a project, you can simply **delete the environment and recreate it from scratch**, without affecting `Python` itself or any of your other projects. This makes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





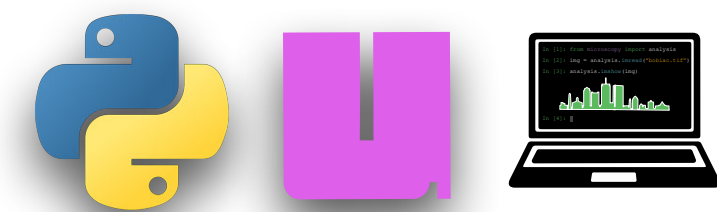
Summary

To work with Python we need:



Introduction to Python

Part I - General Python Concepts




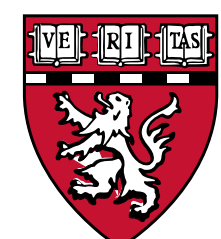
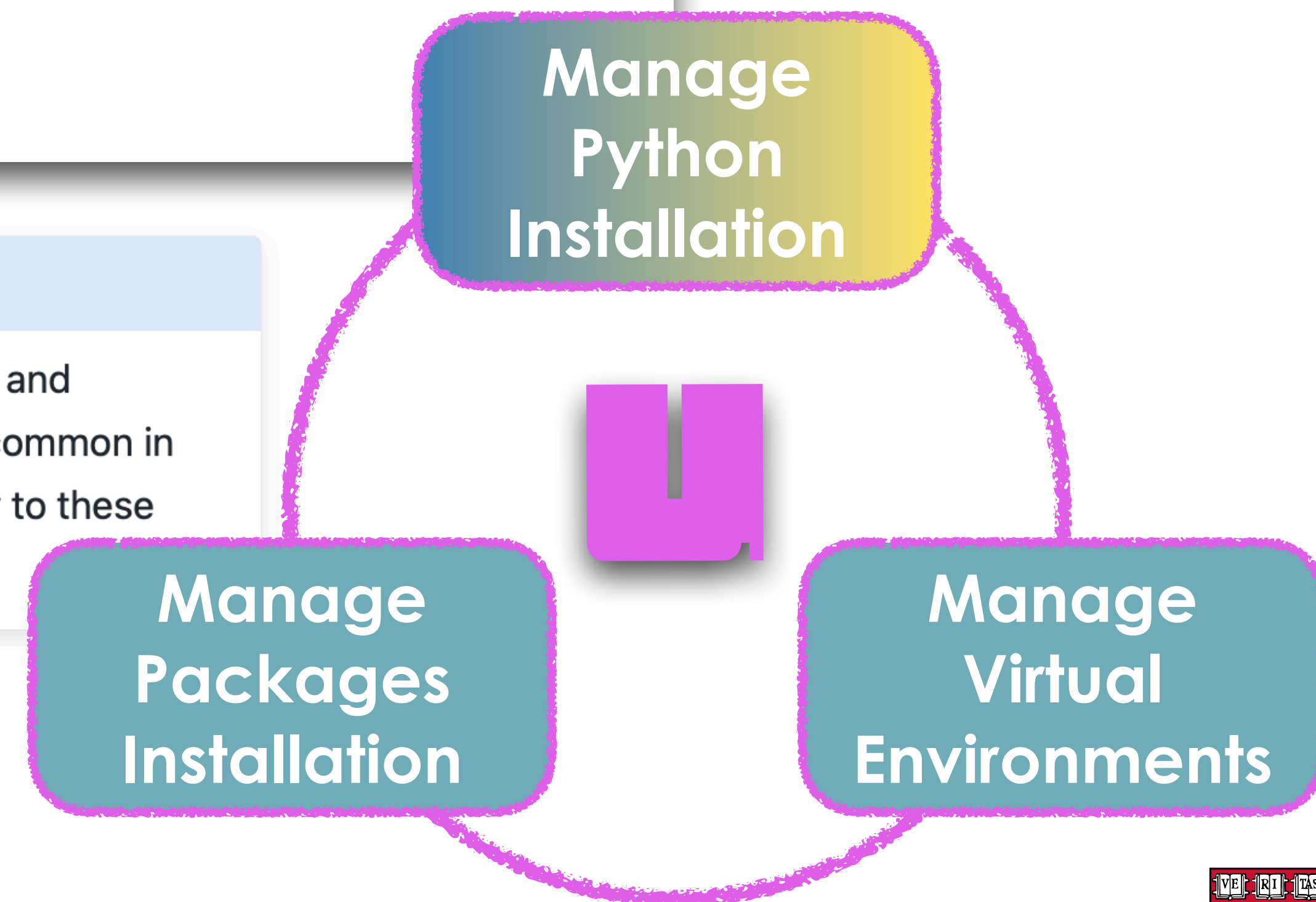
UV

`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.

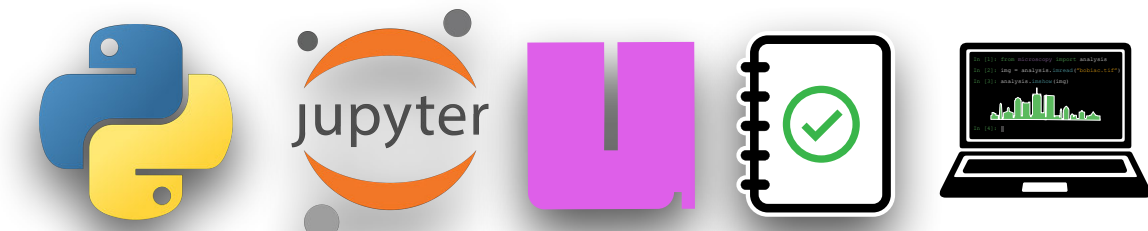
Note

 **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.



Introduction to Python

Part II - Jupyter Notebooks and *juv*



Bo BiAC
Boston Bioimage Analysis Course | 2026

Image Analysis Collaboratory | CITE | HARVARD MEDICAL SCHOOL

Search [] + K

Home

Course Material

- 01 - **Getting Started with Python**
 - Introduction to Python
 - Jupyter Notebooks and juv**
 - Glossary
- 02 - Python Basics
- 03 - Introduction to Digital Images

Python code is usually saved in plain text files ending in `.py`. In this course, however, we will mostly **not** work with `.py` files directly. Instead, we will use **Jupyter Notebooks** (`.ipynb` files). This page explains what they are, how we are going to use them.

1. Terminal Commands

Before we start, let's go over some of the `terminal` commands that could be useful during the course.

First of all, what is a `terminal`? A `terminal` is a text-based interface that allows you to interact with your computer's operating system. You can use it to run commands, navigate the file system, and manage files and directories.

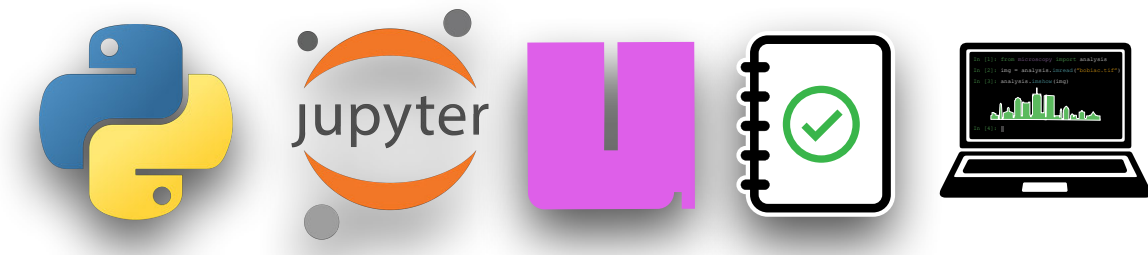
On **macOS** and **Linux**, this program is simply called the **Terminal**. On **Windows**, we suggest using **PowerShell** (instead of the older `cmd` Prompt) since most of the commands we will use behave the same way as on macOS and Linux.

Contents

1. Terminal Commands
2. What is a Jupyter Notebook?
3. JupyterLab
4. Python Packages and `juv`
5. Summary Table



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 **interactive python notebook**). 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.



Introduction to Python

Part II - Jupyter Notebooks and *juv*



Jupyter Notebooks

JupyterLab
jupyterlab.readthedocs.io

The screenshot shows a Jupyter Notebook interface with several annotations:

- new cell**: Points to the '+' icon in the toolbar.
- run cell (shift + enter)**: Points to the play button icon in the toolbar.
- restart kernel (python process)**: Points to the refresh icon in the toolbar.
- select cell type**: Points to the 'Markdown' dropdown menu in the toolbar.
- code cell**: Points to the code cell containing the following text:

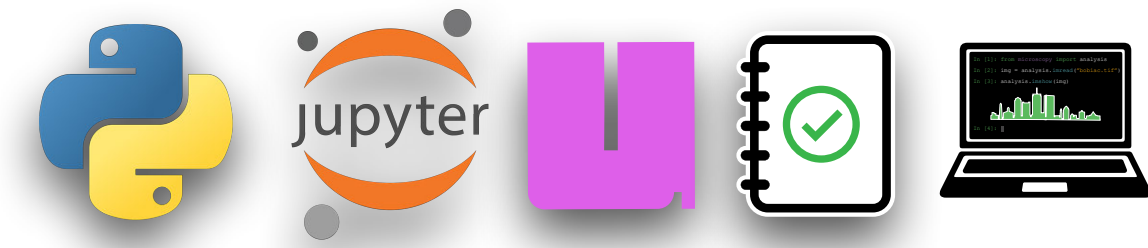
```
[1]: # /// script
# requires-python = ">=3.10"
# ///

# Standard library imports (no need to declare in dependencies)
```
- markdown cell (text)**: Points to the 'Overview' section below the code cell.



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.

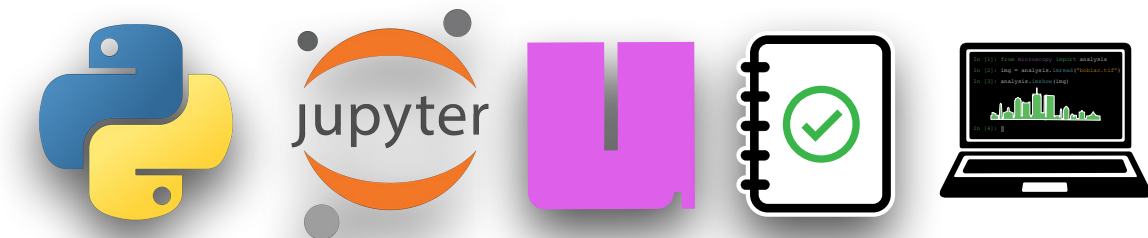


To handle all of this automatically, we use `juv`, a tool built on top of `uv` specifically designed for `Jupyter Notebooks` (check out the [Getting Started with uv](#) to learn more about `uv`). Instead of manually creating environments and installing packages, `juv` reads a dependencies list declared 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:

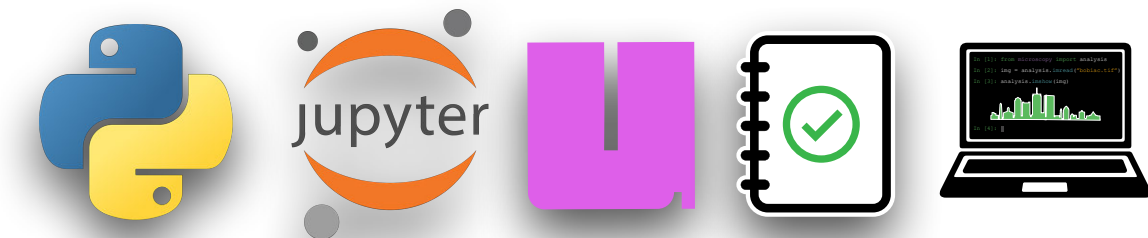
```
# /// script
# requires-python = ">=3.12"
# dependencies = [
#     "numpy",
#     "matplotlib",
# ]
# ///
```

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



Introduction to Python

Part II - Jupyter Notebooks and *juv*



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 juv run bobiac2026.ipynb  ㄿ#3
Last login: Sun Jul  5 08:19:46 on ttys002
(base)
~
> cd /Users/fdrisp/Desktop/bobiac_2026
(base)
~/Desktop/bobiac_2026
> uvx juv init bobiac2026.ipynb
Initialized notebook at `bobiac2026.ipynb`
(base)
~/Desktop/bobiac_2026
> uvx juv run bobiac2026.ipynb
[I 2026-07-05 08:22:41.464 ServerApp] jupyter_lsp | extension
was successfully linked.
[I 2026-07-05 08:22:41.466 ServerApp] jupyter_server_terminals
| extension was successfully linked.
[I 2026-07-05 08:22:41.468 ServerApp] jupyterlab | extension w
as successfully linked.
|
```



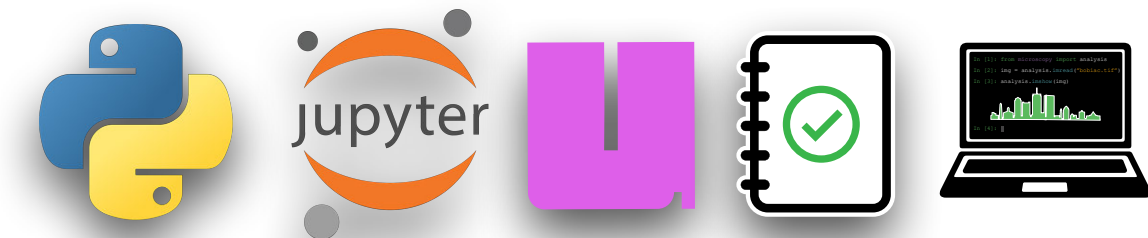
*uvx = uv tool run

github.com/manzt/juv



Introduction to Python

Part II - Jupyter Notebooks and *juv*



create a new notebook (and specify python version)

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

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

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

add packages

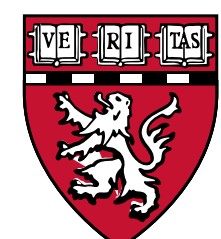
```
uvx juv run <name.ipynb>
```

run the notebook

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

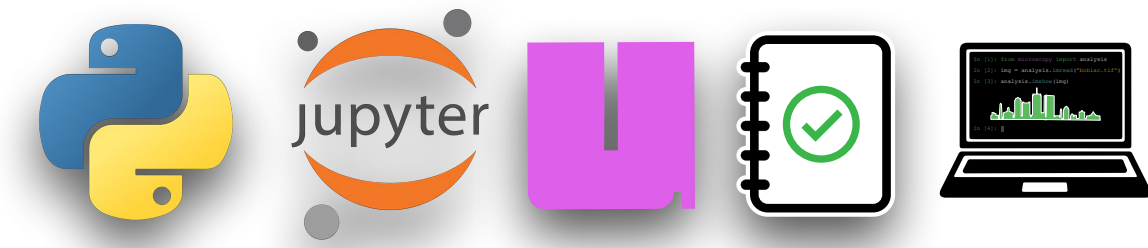
the terminal

```
uvx juv run bobiac2026.ipynb  ㄿ#3
Last login: Sun Jul  5 08:19:46 on ttys002
(base)
~
> cd /Users/fdrgrp/Desktop/bobiac_2026
(base)
~/Desktop/bobiac_2026
> uvx juv init bobiac2026.ipynb
Initialized notebook at `bobiac2026.ipynb`
(base)
~/Desktop/bobiac_2026
> uvx juv run bobiac2026.ipynb
[I 2026-07-05 08:22:41.464 ServerApp] jupyter_lsp | extension
was successfully linked.
[I 2026-07-05 08:22:41.466 ServerApp] jupyter_server_terminals
| extension was successfully linked.
[I 2026-07-05 08:22:41.468 ServerApp] jupyterlab | extension w
as successfully linked.
```



Introduction to Python

Part II - Jupyter Notebooks and *juv*



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

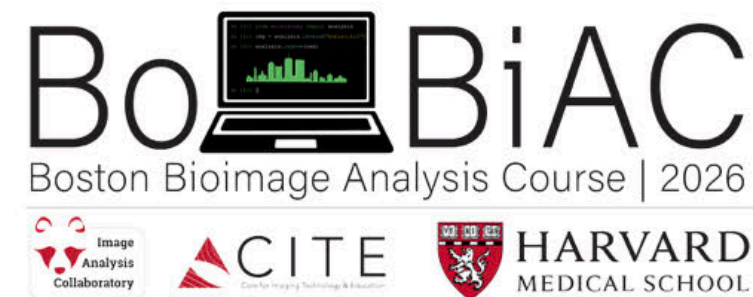
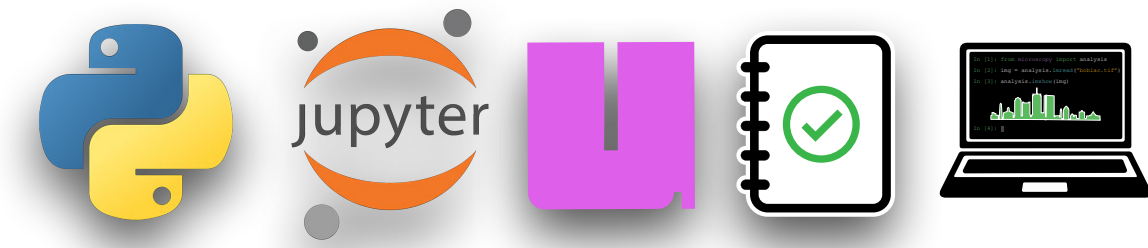


github.com/manzt/juv



Introduction to Python

Part II - Jupyter Notebooks and *juv*



Q Search

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.

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

Extra Practice

- [Working with Bioimages](#)
- [Classic Segmentation](#)

Previous

< [Notebook 2: Do two different protein clusters co-localize?](#)

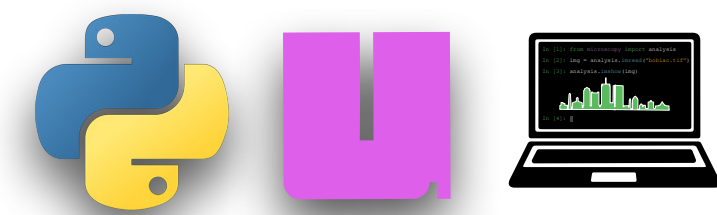


juv by Trevor Manz

github.com/manzt/juv



Introduction to Python



Part III - Python and uv

The screenshot shows the BoBiAC course website. The header includes the BoBiAC logo (Boston Bioimage Analysis Course | 2026) and logos for the Image Analysis Collaboratory, CITE Center for Imaging Technology & Education, and Harvard Medical School. A search bar is present with a magnifying glass icon and a keyboard shortcut icon (⌘ + K). Below the search bar is a 'Home' button. The 'Course Material' section lists several items, with '01 - Getting Started with Python' highlighted in green and enclosed in a blue hand-drawn box. Underneath it, 'Introduction to Python' and 'Getting started with uv' are also listed. The main content area features a green header for '1. Terminal Commands'. The text below explains that this is a brief introduction to uv, a tool used in the course, and refers to the official uv documentation. It then discusses terminal commands, defining a terminal as a text-based interface for interacting with the operating system. It notes that on macOS and Linux, the terminal is simply called the Terminal, while on Windows, PowerShell is recommended instead of the older cmd Prompt. A right-hand sidebar contains a 'Contents' menu with five items: '1. Terminal Commands', '2. Installing uv', '3. Virtual Environments with uv', '4. The Python Files and uv', and '5. Summary Table'. The first item is highlighted in green.

BoBiAC
Boston Bioimage Analysis Course | 2026

Image Analysis Collaboratory CITE HARVARD MEDICAL SCHOOL

Search ⌘ + K

Home

Course Material

- 00 - Introduction to BoBiAC
- 01 - Getting Started with Python
 - Introduction to Python
 - Getting started with uv
- Jupyter Notebooks and jupyter
- Glossary

Getting started with uv

This is not an extensive tutorial on `uv`, but rather a **brief introduction** to some of the tools that we will use in this course. For a more complete guide, check out the [official uv documentation](#).

1. Terminal Commands

Before we start, let's go over some of the `terminal` commands that could be useful during the course.

First of all, what is a `terminal`? A `terminal` is a text-based interface that allows you to interact with your computer's operating system. You can use it to run commands, navigate the file system, and manage files and directories.

On **macOS** and **Linux**, this program is simply called the **Terminal**. On **Windows**, we suggest using **PowerShell** (instead of the older `cmd` Prompt) since most of the commands we will use behave the same way as on macOS and Linux.

Contents

1. Terminal Commands
2. Installing `uv`
3. Virtual Environments with uv
4. The Python Files and `uv`
5. Summary Table

