



Software: Python

This guide will walk you through the process of installing **Anaconda**, a Python distribution that simplifies the setup of data science tools and packages, including **Jupyter Notebooks**.

Anaconda provides everything you need to get started with scientific computing and interactive notebooks in one package — no need to install Python and libraries separately.

For the **CEE6501** course, you'll use Jupyter notebooks to follow live coding examples, solve practice problems, and interact with course content.

Learning Objectives

By the end of this guide, you will:

- Understand what Anaconda is and why we use it
- Install Anaconda on your computer
- Verify that the installation works
- Understand what conda environments are and create a test environment
- Launch Jupyter Lab to begin working with notebooks



Installing Anaconda



Windows

1. Visit the [Anaconda Downloads page](#) :

<https://www.anaconda.com/products/distribution>

2. Download the **Windows 64-bit (exe installer)**.

3. Run the installer and follow the instructions:

- Click **Next** to continue through setup screens.
- When prompted, select **Install for All Users (requires admin privileges)**.
- **Keep default install location.** (this should install Anaconda to <C:\ProgramData\Anaconda3>)
- Check "**Add Anaconda to my PATH environment variable**" (optional but convenient).
- Click **Install**.



macOS

1. Go to the [Anaconda Downloads page](#) :

<https://www.anaconda.com/products/distribution>

2. Download the **macOS (64-bit Graphical Installer)**.

3. Open the downloaded [.pkg](#) file and follow the steps to install.

Verifying Your Anaconda Installation

After installing Anaconda, we'll verify that everything is working properly by:

1. Opening the command-line interface
2. Checking that `conda` is available
3. Creating a test environment
4. Running Jupyter Lab



Windows

Anaconda includes its own terminal called **Anaconda Prompt**. You can find it in the Start Menu after installation.

1. Click the **Windows** key and search for **Anaconda Prompt**
2. Open it and type the following:

```
conda --version  
python --version
```

If both commands return a version number, Anaconda is installed correctly!



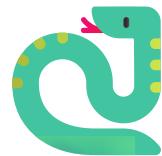
macOS

On macOS, Anaconda integrates with your default terminal.

1. Open **Terminal** (**⌘ + Space**, then search for "Terminal")
2. Type the following:

```
conda --version  
python --version
```

You should see version numbers, confirming that Anaconda is set up properly.



Conda Environments

A conda environment is an isolated workspace that contains its own installation of Python and any packages you install.

This allows you to avoid version conflicts and keeps your projects clean and reproducible.

For example, one project may require Python 3.9 and another may require Python 3.11 — using environments allows both to coexist on your machine.

To keep things clean, we will later be creating a new environment specifically for this course. This ensures that your setup matches the environment used in the lectures and examples.

Creating a Test Conda Environment

To test your Anaconda installation, let's create a simple environment

In your terminal (Windows: Anaconda Prompt, macOS: Terminal), run:

```
conda create -n test_environment python=3.11 jupyterlab
```

You'll be prompted to proceed — type y and press Enter.

Once created, activate the environment with:

```
conda activate test_environment
```

You should see the environment name appear in parentheses at the start of your terminal prompt, like this:

```
(test_environment) C:\Users\YourName>
```

Creating the Course Conda Environment

Before you can run any of the course notebooks, you need to create a dedicated Conda environment with the required Python version and packages.

Step 1: Navigate to the course repository

Open a terminal (Windows: Anaconda Prompt, macOS: Terminal)

Navigate to the folder where you cloned the course repository.

Windows example:

```
cd C:\Users\YourName\Documents\GitHub\CEE6501
```

macOS example:

```
cd ~/Documents/GitHub/CEE6501
```

 *Tip: You can drag the course folder directly into the terminal window to automatically insert the correct path.*

Step 2: Create and activate the course environment

From the **root of the course repository**, run:

```
conda env create -f environment-student.yml  
conda activate CEE6501-student
```

This will:

- Create a new Conda environment named `CEE6501-student`
- Install all required packages and exact versions used in the course

The environment only needs to be created **once**.

Step 3: Verify the environment

To confirm that the environment was created successfully, list all Conda environments:

```
conda env list
```

The active environment will be marked with an asterisk (*).

To see all installed packages and versions in the active environment, run:

```
conda list
```

If these commands run without errors, your Python environment is ready for the course.

```
Anaconda Prompt  X + ▾

(base) C:\Users\ebbruun3>cd C:\Users\ebbruun3\Documents\GitHub\CEE6501

(base) C:\Users\ebbruun3\Documents\GitHub\CEE6501>conda activate CEE6501-student

(CEE6501-student) C:\Users\ebbruun3\Documents\GitHub\CEE6501>conda env list

# conda environments:
#
base                  C:\ProgramData\anaconda3
CEE6501-dev           C:\Users\ebbruun3\.conda\envs\CEE6501-dev
CEE6501-student        * C:\Users\ebbruun3\.conda\envs\CEE6501-student
bar_robot_cell_v2      C:\Users\ebbruun3\.conda\envs\bar_robot_cell_v2
bar_v2_rh7              C:\Users\ebbruun3\.conda\envs\bar_v2_rh7
bar_v2_rh8              C:\Users\ebbruun3\.conda\envs\bar_v2_rh8
compas_dev             C:\Users\ebbruun3\.conda\envs\compas_dev
compas_fab_dev          C:\Users\ebbruun3\.conda\envs\compas_fab_dev
website_doc             C:\Users\ebbruun3\.conda\envs\website_doc
website_doc2            C:\Users\ebbruun3\.conda\envs\website_doc2

(CEE6501-student) C:\Users\ebbruun3\Documents\GitHub\CEE6501>conda list
# packages in environment at C:\Users\ebbruun3\.conda\envs\CEE6501-student:
#
#           Name                Version         Build  Channel
_openmp_mutex           4.5                  2_gnu    conda-forge
_python_abi3_support     1.0                  hd8ed1ab_2  conda-forge
anyio                   4.12.0               pyhcf101f3_0  conda-forge
argon2-cffi             25.1.0               pyhd8ed1ab_0  conda-forge
argon2-cffi-bindings    25.1.0               py314h5a2d7ad_2  conda-forge
arrow                    1.4.0               pyhcf101f3_0  conda-forge
```

Additional useful conda commands

If you add packages to your environment, you can export a new environment .yml file with:

```
conda env export > environment.yml
```

If you want to delete a conda environment:

```
conda env remove --name CEE6501-student  
conda env remove --name CEE6501-dev
```



JupyterLab: Working with Notebooks

Most of the course materials and examples are provided as **Jupyter notebooks** (`.ipynb` files).

Jupyter Notebooks are **interactive computational documents** that combine code, visual output, text, equations, and more in a single file (`.ipynb`). They're widely used in data science, education, scientific research, and rapid prototyping.

The name **Jupyter** comes from **Julia**, **Python**, and **R**, though the project now supports many languages via **kernels**. In this course, we'll be using Python exclusively.

JupyterLab is an interactive environment that allows you to open, run, and edit these notebooks.

In this course, JupyterLab will run **inside the Conda environment** you just created.

Key Features

-  **Executable code** in small chunks (cells)
-  **Markdown** cells for formatting, descriptions, and embedded LaTeX math
-  **Inline visualizations** using libraries like `matplotlib`, `plotly`, etc.
-  **Immediate feedback**: great for learning, debugging, and exploration

All in a single, interactive file.

Why Use Jupyter in This Course?

- Allows you to write code in steps and test incrementally
- Encourages **documentation** and **code explanation** side by side
- Makes it easier to **present and share** your work (e.g., in reports or papers)
- Ideal for **data exploration, plotting, and prototyping** algorithms
- Fully integrated with **VS Code**, giving you an IDE experience



Types of Notebook Cells

There are two main cell types:

Markdown Cells

These are for formatted text. They use Markdown syntax and can contain:

Headers (#, ##, etc.)

Lists

Bold, italic

inline code

LaTeX math:

Links and Images

```
In [1]: ### **Code Cells**
```

These are for Python code. You run them to get output, plots, errors, etc.

```
print("Write something and run!")
a = 30
b = a + 10
print(b)
```

Write something and run!

40

Launching JupyterLab

Make sure your course environment is activated:

```
conda activate CEE6501-student
```

Then launch JupyterLab by running:

```
jupyter lab
```

This command will:

- Start a local JupyterLab server
- Automatically open JupyterLab in your web browser

You should see a file browser where you can navigate to the course notebooks and open them.

Open the `L1_software_python.ipynb` file

The screenshot shows the JupyterLab interface with several panels:

- File Browser:** On the left, it lists files in the directory `/Lectures / L1 /`. The `L1_python.ipynb` file is selected.
- Code Cell:** In the center, a code cell contains the following Python code:

```
[3]: # This is an example of an interactive cell
a = 5
b = 10

print("This equals: ", a*b)
This equals: 50
```
- Terminal:** On the right, the terminal window shows logs from the `jupyter lab` command. A red oval highlights the first few lines of the log output.

Terminal Log Output (highlighted area):

```
[I 2026-01-04 18:33:50.432 ServerApp]
To access the server, open this file in a browser:
  file:///C:/Users/ebrunr3/AppData/Roaming/Jupyter/runtime/jpserver-3440-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/lab?token=aala744e15d1b43ala8350e56121953657fc7fa2b335e2dd
  http://127.0.0.1:8888/lab?token=aala744e15d1b43ala8350e56121953657fc7fa2b335e2dd
[I 2026-01-04 18:33:50.521 ServerApp] Skipped non-installed server(s): basedpyright, bash-language-server-nodejs, javascript-typescript-langsServer, jedi-language-server, julia-language-server, python-language-server, python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-fied-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languge-server
[W 2026-01-04 18:33:53.147 LabApp] Could not determine jupyterlab build status without nodejs
[W 2026-01-04 18:34:02.893 ServerApp] Notebook Lectures/L1/L1_python.ipynb is not trusted
[T 2026-01-04 18:34:03.193 ServerApp] Kernel started: cfbbba0c5-4d1e-4a84-b60f-891ab4914013
[T 2026-01-04 18:34:04.235 ServerApp] Adapting from protocol version 5.3 (kernel cfbbba0c5-4d1e-4a84-0.5.4 (client)).
[W 2026-01-04 18:34:04.236 ServerApp] Connecting to kernel cfbbba0c5-4d1e-4a84-b60f-891ab4914013.
[W 2026-01-04 18:34:04.242 ServerApp] The websocket_ping_timeout (90000) cannot be longer than the wait (30000).
  Setting websocket_ping_timeout=30000
[I 2026-01-04 18:34:04.244 ServerApp] Adapting from protocol version 5.3 (kernel cfbbba0c5-4d1e-4a84-0.5.4 (client)).
[I 2026-01-04 18:34:04.244 ServerApp] Connecting to kernel cfbbba0c5-4d1e-4a84-b60f-891ab4914013.
[I 2026-01-04 18:34:04.251 ServerApp] Adapting from protocol version 5.3 (kernel cfbbba0c5-4d1e-4a84-0.5.4 (client)).
[I 2026-01-04 18:34:04.252 ServerApp] Connecting to kernel cfbbba0c5-4d1e-4a84-b60f-891ab4914013.
[I 2026-01-04 18:36:02.967 ServerApp] Saving file at /Lectures/L1/L1_python.ipynb
[I 2026-01-04 18:36:04.929 ServerApp] Saving file at /Lectures/L1/L1_python.ipynb
```

```
In [7]: # This is an example of an interactive cell  
# press shift + enter to run  
  
a = 5  
b = 10  
  
print("This equals: ", a*b)
```

This equals: 50



Markdown Syntax Quick Guide



Headers

```
# H1 - Main Title  
## H2 - Section  
### H3 - Subsection
```



Text Formatting

- **Bold** text: `**bold text**`
- *Italic* text: `*italic text*`
- ***Bold and Italic***: `***bold and italic***`

Example: This is **bold**, this is *italic*, and this is ***both***.



Lists

Unordered List: Use a hyphen `-` or asterisk `*`

Example:

- Apple
- Banana
- Cherry

Ordered List: Use numbers followed by a period `1.`

Example:

1. First item
2. Second item
3. Third item

Inline Code

Use backticks ``` to format inline code.

Example: The variable has a value of 20, `a = 20`.

Code Blocks

Use triple backticks ````` to create multi-line code blocks.

Example:

```
def greet(name):
    return f"Hello, {name}!"
```



LaTeX Math

Write math using `$...$` for inline or `$$...$$` for block equations.

Inline Example: `$E = mc^2$` → $E = mc^2$

Block Example: `$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$`

Renders as:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



Links

You can embed clickable links and display images using Markdown.

Syntax: [Google](https://www.google.com/)

Example: [Google](#)



Images

Syntax: ! [Alt Text]

(https://upload.wikimedia.org/wikipedia/commons/3/38/Jupyter_logo.svg)



Example:

Export A Notebook (Terminal)

You can export a notebook from the command line using:

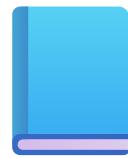
```
jupyter nbconvert MyNotebook.ipynb --to pdf  
jupyter nbconvert MyNotebook.ipynb --to html  
jupyter nbconvert MyNotebook.ipynb --to script
```

export as html slides:

```
jupyter nbconvert --to slides "MyNotebook.ipynb" --post serve
```

> To export as PDF:

- **macOS**: `brew install --cask mactex`
- **Windows/Linux**: install MiKTeX or TeX Live



What's Next

Now that you have Python and Conda installed and can activate the course environment, you're ready to start working with the course materials themselves. In the next tutorial, we'll set up Visual Studio Code as a unified workspace for editing code, running Jupyter notebooks, and managing version control in one place.