IUT PARIS CITE
BUT VCOD
2025-2026

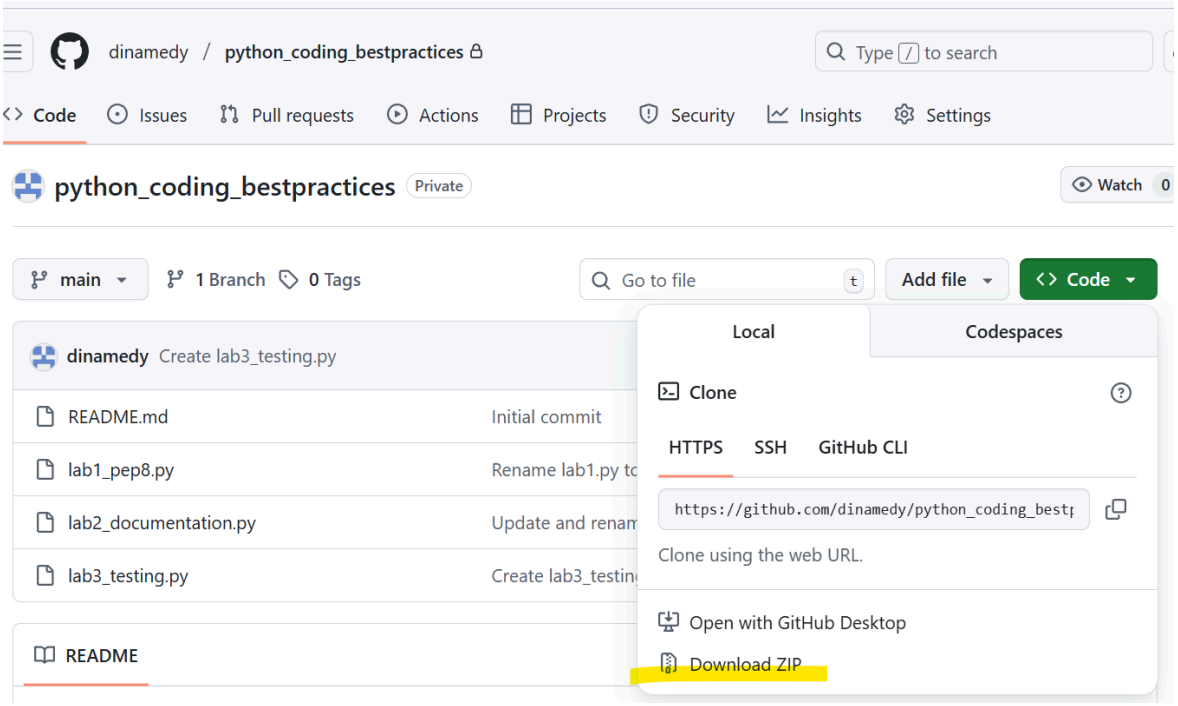
TP : Coding best practices
Applied to Python

Pre-requisites :
This Lab can be done remotely on github, a github account in mandatory.
if you want or need to work locally, you need to install vscode
https://code.visualstudio.com/

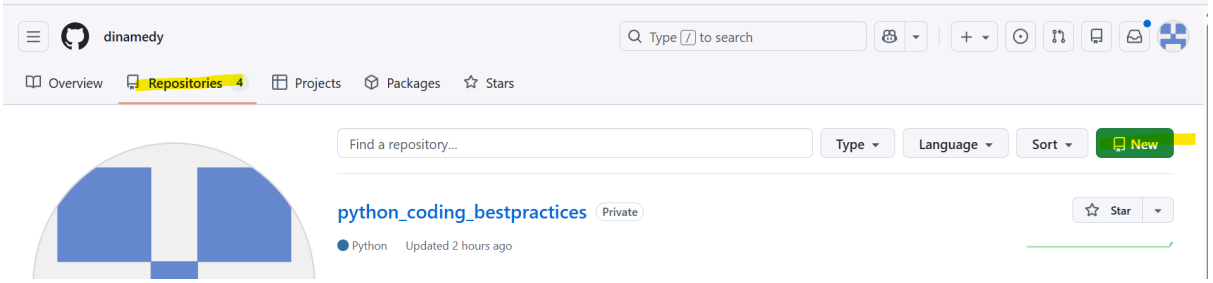
Set-up

1/Connect to github or create your account

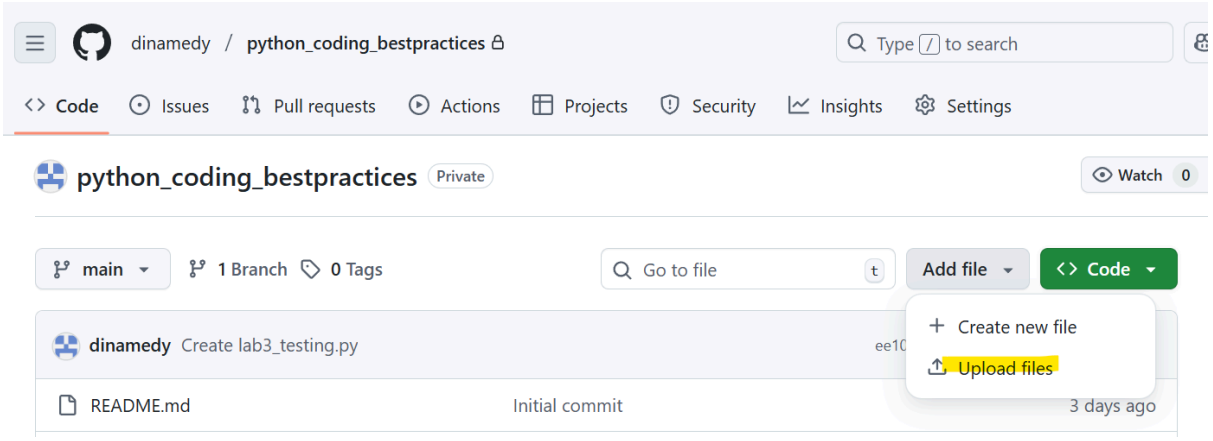
2/ go to https://github.com/dinamedy/python_coding_bestpractices repository, download it and unzip it on your local machine.



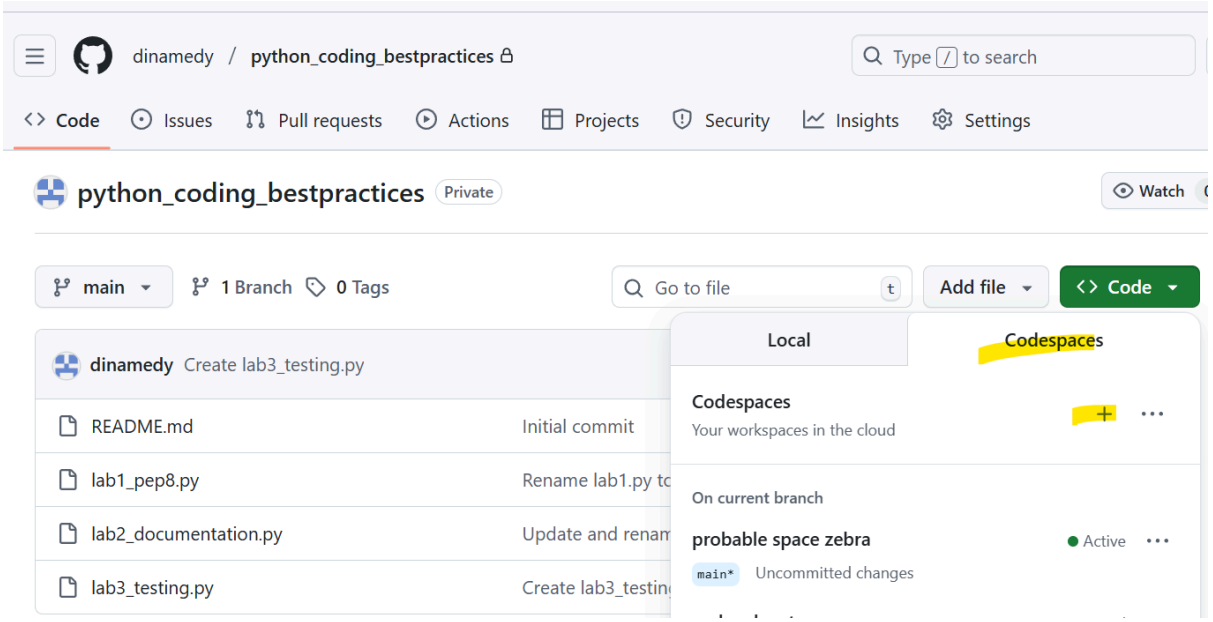
3/On github, go to “repositories” in the menu and create a new repository “**python_conding_bestpractices”**



4/Upload the 3 ".py" files you already download in step 2 :



5/Open a codespace environment on github
this will launch the vscode IDE directly on github



Lab 1: PEP 8 best practices

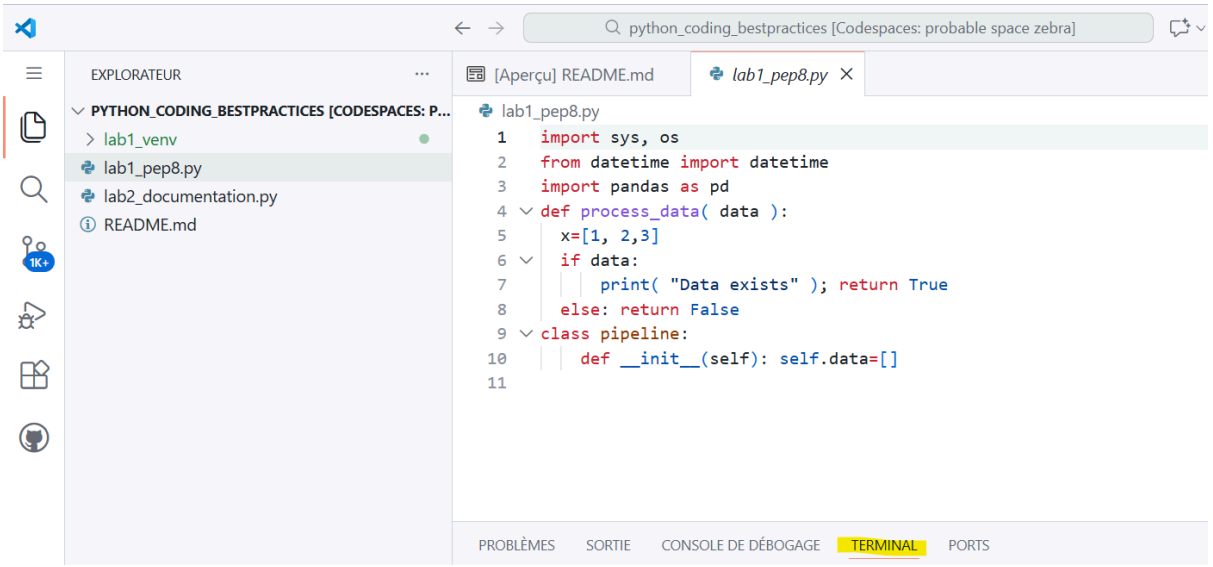
Objective: Experiment with `flake8` (linter), `isort` (import sorter), and `black` (formatter) to enforce PEP 8 standards automatically.

You have received a script called `lab1_pep8.py`. It functions correctly but it violates multiple coding standards. Your task is to clean it up using the tools listed above.

Step 1: Open Github Codespace from the forked "python_coding_bestpractices" repository

This will open your repository in vscode

Open lab1_pep8.py script :



⇒Go to the terminal of vscode

- Create a virtual environment: `python -m venv lab1_venv`
- Install the python libraries : `pip install flake8 isort black`

Step 2: Linting

- Run **Flake8**: `flake8 lab1_pep8.py` if you have an error, test :

  `python -m flake8 lab1_pep8.py`

- Identify and list error codes reported by flake8

Step 3: Formatting

- Run **isort**: `isort lab1_pep8.py (or python -m isort lab1_pep8.py)`
  - Did it separate standard library imports (`sys`, `os`) from third-party (`pandas`)?
- Run **Black**: `black lab1_pep8.py (or python -m black lab1_pep8.py)`
  - Did it fix the spacing around `x=[1, 2,3]`?

Step 4: Final Check

- Run `flake8` again : `python -m flake8 lab1_pep8.py`
  - Do you still see some errors ? Why didn't Black fix this?

Lab 2: Documentation best practices

Objective: Write professional docstrings using the PEP 257 standard and generate a static HTML documentation site using Sphinx.

You are building a shared library module. You need to document it so other developers can use it without needing to read your source code.

Step 1: Open lab2_documentation.py in github codespace

This code uses the Google Style format.

Step 2: Validate Compliance

- Install pydocstyle (library to check PEP 257): `pip install pydocstyle`
- Run: `python -m pydocstyle lab2_documentation.py`
- Fix any errors until the command returns nothing.

Step 3: Generate HTML Documentation

- Install Sphinx: `pip install sphinx`.
- Initialize docs: `sphinx-quickstart docs` (accept defaults by pressing enter to first question)
- Change repository to docs : `cd docs`
- Build**:** `python -m sphinx -b html . _build`
- **Result:** Open `docs/_build/html/index.html` in your browser to see your professional documentation.

Lab 3: Unit Testing

Objective: Write a robust test suite using the built-in `unittest` framework to verify logic and handle edge cases.

You are coding the backend for a bank. You have a class `BankAccount` and you must ensure that money cannot be withdrawn if funds are insufficient.

Step 1: Open the code to test in github codespace `lab3_testing.py`

Step 2: Create a test script

Create a python script named `"test_bank.py"` to test the class `BankAccount` in `lab3_testing.py` using `unittest.TestCase` to ensure that money cannot be withdrawn if funds are insufficient.

Step 3: Experiment

- Run the test: `python test_bank.py`
- Go to `lab3_testing.py` and comment out the `if amount > self.balance` check.
- Run the test again.
- Explain why `test_withdraw_overdraft` failed (it expected a `ValueError` but none was raised).