I’ll show venv first (simple/portable), then Conda. Pick one.

**0) Create your project folder**

mkdir myproject && cd myproject

**1) Make a virtual environment**

**Option A — venv (works everywhere)**

python -m venv .venv

# macOS/Linux

source .venv/bin/activate

# Windows PowerShell

# .\.venv\Scripts\Activate.ps1

python -m pip install --upgrade pip

pip install black ruff pytest ipykernel

**Option B — Conda**

conda create -n myproject python=3.11 -y

conda activate myproject

pip install black ruff pytest ipykernel

**2) Open in VS Code and select the interpreter**

* VS Code → **File → Open Folder…** → select myproject.
* **Ctrl/Cmd+Shift+P → “Python: Select Interpreter”** → pick your .venv (or Conda) env.

Add helpful VS Code settings (create .vscode/settings.json):

{

"python.defaultInterpreterPath": ".venv/bin/python",

"editor.formatOnSave": true,

"editor.codeActionsOnSave": { "source.fixAll": "always", "source.organizeImports": "always" },

"python.analysis.typeCheckingMode": "basic",

"python.testing.pytestEnabled": true,

"python.testing.pytestArgs": ["-q"],

"ruff.enable": true,

"ruff.organizeImports": true,

"ruff.lint.enable": true

}

On Windows, set "python.defaultInterpreterPath": ".venv\\Scripts\\python.exe".

**3) Minimal project files**

myproject/

├─ .vscode/

│ └─ settings.json

├─ app.py

├─ pyproject.toml

├─ .gitignore

└─ requirements.txt

**app.py**

def main():

print("Hello from my project!")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**pyproject.toml** (keep Black & Ruff in sync)

[tool.black]

line-length = 100

target-version = ["py311"]

[tool.ruff]

line-length = 100

target-version = "py311"

lint.select = ["E","F","I","B","UP"]

**.gitignore**

.venv/

\_\_pycache\_\_/

\*.pyc

.ipynb\_checkpoints/

.env

**requirements.txt** (optional, if you like pinning later)

black

ruff

pytest

ipykernel

**4) Initialize Git (local repo)**

git init

git add .

git commit -m "Initial commit: VS Code + Python env scaffolding"

(Recommended) Normalize line endings on Windows/macOS:

git config core.autocrlf input # macOS/Linux

# OR

git config core.autocrlf true # Windows

(Recommended) Main branch:

git branch -M main

**5) Create a remote (GitHub) and push**

**HTTPS (simple):**

git remote add origin https://github.com/<your-user>/<your-repo>.git

git push -u origin main

**SSH (no passwords after setup):**

1. Create an SSH key if you don’t have one yet:

ssh-keygen -t ed25519 -C "<your email>"

# press enter for defaults, add passphrase if you like

1. Copy the **public** key (e.g., ~/.ssh/id\_ed25519.pub) to GitHub → Settings → **SSH and GPG keys**.
2. Then:

git remote add origin git@github.com:<your-user>/<your-repo>.git

git push -u origin main

**6) Run & debug in VS Code**

* Open app.py, set a breakpoint, press **F5** (Run and Debug).
* VS Code will create .vscode/launch.json on first run. You can customize:

{

"version": "0.2.0",

"configurations": [

{

"name": "Run app.py",

"type": "python",

"request": "launch",

"program": "${workspaceFolder}/app.py",

"console": "integratedTerminal",

"envFile": "${workspaceFolder}/.env"

}

]

}

**7) Daily flow (efficient routine)**

1. Activate env (VS Code terminal auto-activates after interpreter selection).
2. Install libs as needed: pip install numpy pandas (etc.).
3. Save often—**format on save** cleans it (Black) and **Ruff** fixes lint/imports.
4. Run tests: pytest -q or the **Testing** sidebar.
5. Commit small changes:
6. git add -p
7. git commit -m "Describe the change"
8. git push

**8) Nice-to-have boosters**

* **pre-commit** (auto run Black/Ruff before each commit):
* pip install pre-commit
* printf "%s\n" \

"repos:

* repo: <https://github.com/psf/black>  
  rev: 24.8.0  
  hooks: [{id: black}]
* repo: <https://github.com/astral-sh/ruff-pre-commit>  
  rev: v0.6.9  
  hooks: [{id: ruff, args: [--fix]}]" > .pre-commit-config.yaml

pre-commit install

* **.env** for secrets/config (VS Code uses it during debug):
* API\_TOKEN=...
* DEBUG=true

**9) Using Conda with VS Code kernels/notebooks**

* Create notebooks/analysis.ipynb.
* Kernel picker → choose your Conda env.
* (Optional) Add the kernel explicitly:
* python -m ipykernel install --user --name myproject --display-name "Python (myproject)"

**10) Cloning an existing repo into a fresh env (common case)**

git clone https://github.com/<user>/<repo>.git

cd repo

python -m venv .venv && source .venv/bin/activate # or conda env create ...

pip install -r requirements.txt # or `pip install -e .` if it’s a package

code .

# VS Code: Python: Select Interpreter → pick the env

If you tell me **venv vs Conda** and your typical stack (e.g., numpy/pandas/nilearn/scikit-learn), I can hand you a tiny, ready-to-run template with those pinned and the VS Code settings pre-wired.

Great—VS Code + Python + a (Conda or venv) environment is a sweet setup. Here’s a fast, reliable way to get going and stay efficient.

**1) One-time setup**

* Install **Python 3.11/3.12**.
* Install **VS Code**.
* In VS Code, install extensions:
  + **Python** (Microsoft)
  + **Pylance** (better IntelliSense)
  + **Jupyter** (if you use notebooks)
  + **GitHub Copilot** or similar (optional)

**2) Create a project + virtual environment**

Pick one of the two common paths.

**Option A — venv (simple, works everywhere)**

# create project

mkdir myproject && cd myproject

# create a local venv \*inside\* the project (keeps things portable)

python -m venv .venv

# activate

# macOS/Linux:

source .venv/bin/activate

# Windows PowerShell:

.\.venv\Scripts\Activate.ps1

# upgrade pip and install a few dev tools

python -m pip install --upgrade pip

pip install black ruff pytest ipykernel

**Option B — Conda (if you already use it)**

# create & activate

conda create -n myproject python=3.11 -y

conda activate myproject

# dev tools

pip install black ruff pytest ipykernel

Tip: put the env **inside** the project (.venv) for venv, or use a **Conda env name that matches the folder**. VS Code will find it automatically.

**3) Open the folder in VS Code**

* **File → Open Folder** → select your project.
* Press **⌘⇧P / Ctrl+Shift+P** → **Python: Select Interpreter** → pick your .venv or Conda env.
* VS Code will now run, lint, debug, and use Jupyter in that env.

**4) Minimal files to add**

Create these in the project root.

**app.py**

def main():

print("Hello, VS Code + Python!")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**requirements.txt (optional if you prefer pip-tools/pyproject.toml)**

black

ruff

pytest

ipykernel

**.gitignore**

.venv/

\_\_pycache\_\_/

.ipynb\_checkpoints/

\*.pyc

.env

**5) VS Code settings for speed (format/lint on save)**

Create .vscode/settings.json:

{

"python.defaultInterpreterPath": ".venv/bin/python",

"editor.formatOnSave": true,

"editor.codeActionsOnSave": {

"source.fixAll": "always",

"source.organizeImports": "always"

},

"python.analysis.typeCheckingMode": "basic",

"python.testing.pytestEnabled": true,

"python.testing.pytestArgs": ["-q"],

"ruff.enable": true,

"ruff.organizeImports": true,

"ruff.lint.enable": true,

"ruff.format.enable": false

}

On Windows, change the interpreter path to .venv\\Scripts\\python.exe.

And add a simple pyproject.toml so ruff and black agree:

[tool.black]

line-length = 100

target-version = ["py311"]

[tool.ruff]

line-length = 100

target-version = "py311"

lint.select = ["E","F","I","B","UP"] # basic + modernizations

**6) Run & debug**

* Open app.py → press **F5** (Run and Debug).
* VS Code will create .vscode/launch.json if needed; you can customize later.

Example launch.json (handy for args and env vars):

{

"version": "0.2.0",

"configurations": [

{

"name": "Run app.py",

"type": "python",

"request": "launch",

"program": "${workspaceFolder}/app.py",

"console": "integratedTerminal",

"envFile": "${workspaceFolder}/.env",

"args": []

}

]

}

**7) Testing quickly**

From the integrated terminal:

pytest -q

Or use the **Testing** sidebar (beaker icon) to discover and run tests.

**8) Jupyter in VS Code (if you use notebooks)**

* Create notebooks/analysis.ipynb.
* Kernel picker (top-right) → select your project env.
* You can export a notebook to .py and back if you prefer scripts.

**9) Typical daily flow (efficient routine)**

1. **Open folder** → VS Code auto-picks your env.
2. **Create/activate env** only once per project.
3. **Install deps**: pip install <pkg> (and update requirements.txt with pip freeze > requirements.txt when you want to lock versions).
4. **Code → Save** → auto-format (Black), lint/fix (Ruff).
5. **Run/Debug** with F5, set breakpoints, inspect variables (left sidebar).
6. **Test** with pytest, watch failures in Problems pane.
7. **Git**: Source Control tab for staging/commits/branches/PRs.

**10) Quality-of-life boosters (optional but great)**

* **pre-commit hooks** (auto format/lint before every commit):
* pip install pre-commit
* pre-commit sample-config > .pre-commit-config.yaml
* # Replace with minimal hooks:
* # - repo: https://github.com/psf/black hook: black
* # - repo: https://github.com/astral-sh/ruff hook: ruff
* pre-commit install
* **Tasks** to automate common commands (.vscode/tasks.json):
* {
* "version": "2.0.0",
* "tasks": [
* { "label": "Tests", "type": "shell", "command": "pytest -q", "group": "test" },
* { "label": "Format", "type": "shell", "command": "black . && ruff check --fix ." }
* ]
* }
* **Environment variables**: put them in .env (VS Code loads it for debug & terminals).
* **Refactoring**: Pylance supports rename symbol (F2), extract method, etc.
* **Notebooks → scripts**: Use “Export as Python” to version code cleanly.

**11) Switching an existing project to a new env**

1. Create/activate the env (venv or Conda).
2. Install deps (pip install -r requirements.txt).
3. VS Code → **Python: Select Interpreter** → pick the new env.
4. Reload window if IntelliSense doesn’t pick up packages.