Clase 005 — VS Code / Cursor para Python y Jupyter

Parte: 0 — Prerrequisitos · Fuente: VS Code Python docs · Cursor docs · The Pragmatic Programmer cap. "Power Editing". · Duración estimada: 60 min.

Clase 005 — VS Code / Cursor para Python y Jupyter

> *Parte: 0 — Prerrequisitos · Fuente: VS Code Python docs · Cursor docs · The Pragmatic Programmer cap. "Power Editing". · Duración estimada: 60 min.*

Objetivo

Que el alumno deje de usar VS Code como Notepad y lo configure como un IDE serio para Python + Jupyter: selector de intérprete, debugger gráfico, linter (ruff), formatter (ruff format), tests integrados y notebooks editables. Bonus: cuándo conviene Cursor (VS Code + IA integrada).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Configurar VS Code con la extensión Python + Jupyter, seleccionando el intérprete del venv del proyecto.
2. Debuggear un script Python paso a paso desde el panel gráfico (breakpoints, watch, call stack).
3. Editar y ejecutar notebooks sin Jupyter web — con autocompletado, type hints y debug de celda.
4. Configurar ruff como linter + formatter (reemplaza black + isort + flake8 en un solo tool).
5. Decidir cuándo usar Cursor (idéntico a VS Code + IA integrada con autorización por chat).

Temas

#	Tema	Por qué importa
1	Selección de intérprete por workspace	El bug "funciona en terminal pero no en VS
2	Debugger gráfico vs print	Breakpoints, watch, evaluación expresiones
3	Notebooks nativos en VS Code	Mejor UX que Jupyter web para edición; mis
4	ruff = linter + formatter en uno	Reemplaza black/isort/flake8/pylint. Más r
5	Tests integrados (pytest)	Run/debug tests con un click; coverage inl
6	Extensiones esenciales	Python, Jupyter, GitLens, ruff, Even Bette
7	Cursor: cuándo sí	Cuando quieres pair-programming con IA sin

Definiciones y características

Workspace

: Concepto de VS Code = una carpeta (o conjunto de carpetas) con configuración asociada en .vscode/settings.json. La configuración del workspace override a la del usuario. Característica: pones .vscode/ en git para que todos los colaboradores hereden la misma config.

Intérprete Python

: Ejecutable concreto (/path/to/.venv/bin/python). VS Code recuerda uno por workspace. Es el origen del 90% de los "funciona en mi máquina" entre IDE y terminal.

ruff

: Linter + formatter en un solo binario, escrito en Rust. Reemplaza black + isort + flake8 + (parte de) pylint con un único tool 10–100× más rápido. Config en [tool.ruff] de pyproject.toml.

Breakpoint

: Marca en una línea (F9) que pausa la ejecución cuando llega ahí. Permite inspeccionar variables, paso a paso, evaluar expresiones — mil veces más eficiente que print.

launch.json

: Config de debug de VS Code. Define perfiles: "debug archivo actual", "debug tests", "debug Django", etc. Cada perfil tiene su program, args, env, justMyCode.

Dataset / recursos

Sin dataset externo. Usamos un script Python con un bug intencional para practicar debug gráfico, y un notebook trivial para verificar autocompletado/type hints.

Ejercicios

1. Selecciona intérprete. En VS Code: Ctrl+Shift+P → "Python: Select Interpreter" → elige el del .venv del proyecto. Verifica con print(sys.executable) en una celda.

2. Debug paso a paso. Toma un script con un bug, pon breakpoint (F9), ejecuta con F5, navega con F10 (next), F11 (step in), Shift+F11 (step out). Inspecciona variables en panel.

3. Configura ruff. En pyproject.toml: [tool.ruff] con line-length = 100, [tool.ruff.lint] con select = ["E", "F", "I", "UP"]. Habilita format on save.

4. Edita un notebook. Abre notebook.ipynb en VS Code, ejecuta una celda, comprueba que el autocompletado funciona con type hints de pandas.

5. Tests con un click. Instala pytest, crea tests/test_simple.py con 2 tests (uno OK, uno FAIL). Usa el panel "Testing" para correr/debuggear.

Homework verificable

Repo con pyproject.toml que incluye [tool.ruff], .vscode/settings.json con python.defaultInterpreterPath y format-on-save habilitado, screenshot del debugger gráfico mostrando un breakpoint activo y variables inspeccionadas.

Criterio de aceptación: Otro alumno clona el repo, abre en VS Code, y al guardar un .py se aplica ruff format automáticamente. El screenshot muestra al menos 1 breakpoint y la variable inspeccionada.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
"Python interpreter is not selected" al ab	Workspace nuevo, VS Code no eligió uno. Fi
Format-on-save no aplica ruff aunque está	Falta declarar ruff como formatter por def
Debugger arranca pero se salta mis breakpo	Estás corriendo el archivo (Ctrl+F5 = sin
Tests no aparecen en el panel "Testing"	VS Code no detectó pytest. Fix: Ctrl+Shift

Cambié interpreter y los imports siguen ro	VS Code cachea symbols del intérprete viej

Preguntas frecuentes

¿VS Code o Cursor?

Cursor = VS Code + IA integrada (chat con contexto del repo, edición multi-archivo). Si pagas Copilot o no te interesa IA, quédate en VS Code. Si quieres pair-programming con IA sin saltar a otra app, Cursor. Las extensiones son las mismas.

¿Debo commitear .vscode/?

Sí la parte compartida: settings.json (interpreter path relativo, formatter, etc.), extensions.json (recomendaciones). No lo personal: .vscode/launch.json con paths absolutos del tester.

¿Notebook en VS Code o en JupyterLab?

VS Code para escribir/refactorizar (autocomplete con type hints, debug por celda, git inline). JupyterLab cuando alguien necesita un navegador y no quiere instalar VS Code (alumno, demo en proyector).

¿Para qué justMyCode: false?

Por default, el debugger se salta código de librerías de terceros (numpy, pandas) — útil para no perderte. Pero a veces el bug viene desde dentro de pandas (datos malformados); con false puedes entrar a ver.

¿Ruff reemplaza todo el stack? ¿No necesito black?

Sí — ruff format es drop-in replacement de black (mismo output prácticamente). Mismo con isort (ruff check --select I --fix) y flake8 (ruff check). Único caso donde aún conviene black: si tu org ya tiene CI con black configurado y no quieres tocar.

Referencias

- VS Code Python docs
- VS Code Jupyter docs
- ruff docs
- Cursor docs

Siguiente clase

Clase 006 — Python: tipos, estructuras, control de flujo

Apéndice: notebook (primer bloque)

VS Code recuerda un intérprete por workspace en .vscode/settings.json. Si no lo configuras, usará el primero que encuentre — generalmente el del sistema. Resultado: import funciona en terminal pero no en VS Code (o al revés). Cómo configurarlo bien: `json

```
import sys
from pathlib import Path

print('Intérprete que ejecuta esta celda:')
```

```
print(f'  {sys.executable}')
print()

in_venv = sys.prefix != sys.base_prefix
print(f'¿Estás en un venv? {in_venv}')
if in_venv:
    print(f'  prefix: {Path(sys.prefix).name}')
else:
    print('  Selecciona un intérprete del venv del proyecto en VS Code.')
```

Archivos complementarios

- notebook.ipynb