Clase 002 — Jupyter y JupyterLab — kernels, magics, debugging, profiling

Parte: 0 — Prerrequisitos · Fuente: VanderPlas, Python Data Science Handbook, cap. 1 — IPython: Beyond Normal Python. · Duración estimada: 90 min.

Clase 002 — Jupyter y JupyterLab — kernels, magics, debugging, profiling

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, Python Data Science Handbook, cap. 1 — IPython: Beyond Normal Python. · Duración estimada: 90 min.*

Objetivo

Que el alumno deje de usar Jupyter como un editor de texto con botón "play" y empiece a usarlo como un entorno exploratorio profesional: con magics que ahorran horas, debugger interactivo (%debug), y profiling real (%timeit, %prun). Al final debe poder diagnosticar por qué un notebook es lento sin adivinar.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar kernel, frontend (Notebook vs JupyterLab vs VS Code) y servidor — y saber qué pasa cuando uno se cuelga.
2. Usar magics esenciales: %timeit, %%time, %run, %load, %matplotlib inline, %debug, %who, %xmode.
3. Conectar un kernel específico a un notebook (ipykernel install --user --name <env>) sin pelearse con el venv equivocado.
4. Debuggear una excepción con %debug y pdb (n, s, c, q, p, l).
5. Profilar código lento con %timeit (microbenchmark) y %prun (line profiler) para decidir dónde optimizar.

Temas

#	Tema	Por qué importa
1	Kernel ↔ frontend ↔ servidor	Saber cuál murió cuando el notebook se cue
2	Modo comando vs modo edición + atajos	Velocidad real: A/B/X/M/Y/Esc/Enter.
3	Magics line (%) vs cell (%%)	El 80% del valor de Jupyter está en las ma
4	%timeit y %%time	Microbenchmark riguroso (varias corridas,
5	%debug + pdb	Inspección post-mortem sin re-correr todo
6	%prun y %lprun	Saber qué función pesa antes de optimizar.
7	Registro de kernels por venv	Cada proyecto, su propio kernel — evita el

Definiciones y características

Kernel

: Proceso Python (u otro lenguaje) que ejecuta el código de las celdas. Vive separado del frontend; si lo matas, pierdes el estado en memoria pero los archivos siguen intactos. Cada notebook se asocia a UN kernel, normalmente el del venv del proyecto.

Frontend

: La interfaz visual (Notebook clásico, JupyterLab, VS Code, Cursor, Colab). Todas hablan el mismo protocolo con el kernel — puedes cambiar de frontend sin perder datos si guardas el .ipynb.

Magic

: Comando especial de IPython, no de Python. Empieza con % (afecta una línea) o %% (afecta la celda entera). Ejemplos: %timeit, %matplotlib inline, %%time, %debug. No funcionan fuera de IPython/Jupyter.

%timeit vs %%time

: %timeit corre la expresión muchas veces, descarta outliers y reporta el mejor → microbenchmark estadísticamente serio. %%time mide una sola corrida del bloque → bueno para operaciones largas donde repetir cuesta. Característica clave: usa %timeit para algo en milisegundos, %%time para algo en segundos.

pdb / %debug

: Debugger interactivo de Python. %debug lo lanza en modo post-mortem después de una excepción — entras al stack en el punto del error sin re-correr nada. Comandos: n siguiente línea, s entra a función, c continúa, p var imprime, u/d sube/baja en stack, q salir.

Dataset / recursos

No requiere dataset externo. Usamos arreglos sintéticos con numpy.random para benchmarks. Para el ejercicio de debug, generamos un ValueError intencional.

Ejercicios

1. Atajos sin mouse. Crea 5 celdas, navega solo con teclado: convierte 2 a markdown, ejecuta todo en orden, borra una, deshaz. Cronométrate.

2. Registra tu kernel. Desde un venv recién creado: python -m ipykernel install --user --name ds-lab-001 --display-name 'DS Lab 001'. Abre Jupyter, selecciona ese kernel, verifica con import sys; sys.executable.

3. Benchmark vectorización. Con %timeit, compara sumar range(10_000) con un for vs np.arange(10_000).sum(). Anota cuántas veces más rápido es NumPy.

4. Post-mortem. Provoca un ZeroDivisionError, luego ejecuta %debug en la siguiente celda y navega el stack con u/d, inspecciona variables con p.

5. Profila una función. Escribe una función que ordene una lista 1000 veces con sort burbuja. Ejecuta %prun -s cumulative tu_func(). Identifica la línea más cara.

Homework verificable

Entrega un notebook homework.ipynb con: (a) celda que muestra sys.executable confirmando que usas un kernel registrado por ti; (b) benchmark %timeit comparando sum(range(N)) vs np.arange(N).sum() para N=10k, 100k, 1M; (c) tabla markdown con los resultados; (d) gráfico simple del speedup.

Criterio de aceptación: El notebook abre con kernel propio (no el global), las 3 mediciones corren sin errores, y la conclusión incluye un número concreto ("NumPy es ~50× más rápido para N=1M").

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ModuleNotFoundError aunque acabo de instal	El kernel activo NO es el venv donde corri
El notebook está "congelado" / la barra di	Una celda quedó atrapada en bucle infinito
Cambié código de un módulo importado y el	Python cachea módulos importados. Fix: %lo
%timeit en una celda con asignación da err	Las variables creadas dentro de %timeit no
Outputs gigantes hacen el .ipynb pesado y	Cada output (imagen, tabla) queda guardado

Preguntas frecuentes

¿Notebook clásico o JupyterLab o VS Code?

Para aprender, VS Code (mismo backend, mejor UX: autocomplete con type hints, debug gráfico, git inline). Para reuniones colaborativas en navegador, JupyterLab. El Notebook clásico es legacy — sigue funcionando pero ya no recibe features.

¿Debo crear un kernel por proyecto o usar uno global?

Uno por proyecto. Cada proyecto tiene dependencias distintas que entran en conflicto: el kernel global tarde o temprano se rompe. Comando: python -m ipykernel install --user --name <proyecto>.

¿Cuándo %timeit no es confiable?

Cuando lo que mides toca disco/red/GPU — la varianza es enorme y el min no representa típico. Usa %%time con varios runs manuales y reporta mediana. Tampoco confiable si la primera corrida hace JIT (numba) — calienta con un run previo.

%debug no funciona, no muestra prompt

Necesita haber ocurrido una excepción en el kernel justo antes. Si la celda falló pero el kernel se reinició, perdiste el stack. También: en VS Code Jupyter, usa el panel de debug en su lugar (más cómodo).

¿Por qué mi notebook tarda 30 segundos en abrir si pesa solo 200 KB?

Probablemente trae outputs binarios grandes (imágenes inline en base64). El JSON parece chico pero al renderizar el navegador procesa MB. Limpia outputs y guarda.

Referencias

- VanderPlas, cap. 1 — IPython: Beyond Normal Python.
- IPython magics reference
- JupyterLab user guide

Siguiente clase

Clase 003 — Git y GitHub para data scientists

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import sys, time
```

```
import numpy as np
print('python:', sys.version.split()[0])
print('exec  :', sys.executable)
print('numpy :', np.__version__)
```

Archivos complementarios

- notebook.ipynb