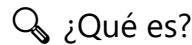


CHEATSHEET de Debugging (Python, VS Code, AI, Crashes)

1. Python logging module



¿Qué es?

Herramienta para registrar eventos del programa con distintos niveles de severidad.



Niveles de logging:

- **DEBUG** – Información muy detallada
- **INFO** – Eventos normales
- **WARNING** – Algo inesperado, pero el programa sigue
- **ERROR** – Fallo que interrumpe una parte del código
- **CRITICAL** – Error muy grave que puede parar el programa



Configuración básica:

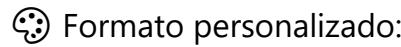
```
import logging

logging.basicConfig(level=logging.DEBUG)
logging.debug("Mensaje debug")
```



Guardar en archivo:

```
logging.basicConfig(filename="app.log", level=logging.INFO)
logging.info("Este mensaje va al archivo")
```



Formato personalizado:

```
logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(message)s",
    level=logging.DEBUG
)
```



Cuándo usar logging:

- Para entender el comportamiento del programa
- Para ver errores sin ensuciar código con `print()`

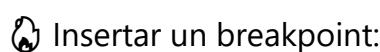
- Para guardar registros persistentes
-

2. Python debugging with pdb



Debugger interactivo incluido en Python. Permite:

- Pausar ejecución
- Revisar variables
- Modificar valores
- Avanzar paso a paso



Insertar un breakpoint:

```
import pdb  
  
pdb.set_trace()
```



Comandos esenciales:

Comando	Acción
n	next: avanza una línea
s	step: entra en funciones
c	continue: sigue hasta otro breakpoint
p var	imprime el valor de var
a	muestra argumentos de función
b num	breakpoint en la línea num
q	salir del debugger
!	ejecutar código Python



Post-mortem debugging:

```
python -m pdb script.py
```

3. Debugging / Breakpoints in VS Code

🔍 ¿Qué es?

Poner puntos donde el programa *se detiene* para inspeccionar su estado.

🔧 Tipos:

- Breakpoint de línea
- Breakpoint condicional
- Breakpoint dentro de loops (solo cuando se cumple la condición)

⭐ Ventajas:

- Inspeccionar variables en vivo
- Ver ejecución paso a paso
- Evitar miles de `print()`
- Mejor para lógica compleja

⚠️ Desventajas:

- Es más lento que ejecutar desde consola
- Requiere configuración inicial (instalar extensiones Python)

4. AI infused debugging & paired programming

💻 ¿Qué es?

Usar herramientas de IA como "segundo programador":

- ChatGPT
- Gemini
- GitHub Copilot
- Code Llama

🤖 ¿Qué pueden hacer?

- Generar código
- Encontrar errores
- Explicar problemas
- Optimizar performance

⚠️ Precaución:

La IA puede dar respuestas incorrectas que *suenan correctas*. Siempre validar.

💻 Paired programming (tradicional):

- **Pilot:** escribe código
- **Copilot:** revisa, mejora, detecta errores

Paired debugging:

- Recorren código juntos
 - El que escribió el código explica su intención
 - El otro ayuda a encontrar el bug
-

5. Resources for debugging crashes

Temas clave:

- Concurrency
- Memoria
- Segmentation faults
- Debugging avanzado

Recursos útiles:

- Artículo: *Speed up your Python program with concurrency*
- Artículo: *Threaded asynchronous magic and how to wield it*
- Lista de causas de segmentation faults

Repos reales para estudiar:

- Flask
- Tornado
- Bottle
- SQLAlchemy
- CherryPy
- Howdoi
- Minecraft-related Python tools

Recomendación:

Para aprender mejor → estudiar **aplicaciones completas** antes que repos gigantes.