Ejercicios de uso del depurador (Python)

ETS – Depuración con un IDE (Python).

Objetivo: practicar el uso del depurador para localizar y corregir errores de sintaxis y de lógica, y para comprender el estado del programa (variables, pila de llamadas y flujo de ejecución).

Requisitos previos

- Python 3.10 o superior.
- Un IDE con depurador (recomendado: Visual Studio Code con la extensión de Python, o PyCharm).
- Carpeta de ejercicios adjunta (cada ejercicio incluye un fichero .py).

Indicaciones generales de depuración

- Ejecuta cada ejercicio en modo depuración (no solo 'Run').
- Coloca puntos de ruptura (breakpoints) en las primeras instrucciones y avanza con Step Over/Step Into para observar el cambio de valores.
- Usa el panel de variables y/o 'Watch' para seguir a, b, c, paso, veces, valor, contador, i, etc.
- Cuando se solicite, usa un punto de ruptura condicional (por ejemplo, 'valor > 10_000_000' o 'i == 3').
- Anota el comportamiento observado y justifica la corrección aplicada.

Ejercicio 1 – Seguimiento de variables

Importa/abre el fichero del Ejercicio1 (Python). Es un programa en el que se modifican los valores de varias variables.

1. Coloca un breakpoint en las primeras instrucciones y ejecuta paso a paso.
2. En cada instrucción, observa el valor de las variables a, b y c.
3. Identifica instrucciones en las que el orden de evaluación sea relevante (por ejemplo, asignaciones encadenadas y operaciones compuestas).

Ejercicio 2 – Tabla de multiplicar y equivalentes a ++/--

Tu programa debe mostrar correctamente la tabla de multiplicar del 6. El fichero contiene errores de sintaxis y de lógica, y además usa operaciones tipo '++' que no existen en Python.

4. Arregla los errores de sintaxis para que el programa ejecute.

5. Realiza los cambios necesarios para que la salida sea correcta (de 1×6 a 10×6).
6. Explica, con un ejemplo, la diferencia conceptual entre usar primero el valor y después incrementarlo (post-incremento) frente a incrementarlo antes de usarlo (pre-incremento). En Python debes expresarlo con instrucciones separadas (por ejemplo, imprimir y luego 'paso += 1').
7. Refactoriza el código para usar un bucle (for) y reducir las líneas repetidas.

Ejercicio 3_2 – Bucle infinito, estado de variables y breakpoints condicionales

El programa debería mostrar un mensaje 4 veces y después otro bloque de 4 mensajes. Sin embargo, hay errores.

8. ¿Qué ocurre cuando ejecutas el programa por primera vez? Coloca un breakpoint dentro del 'while' e investiga qué valor tiene la variable 'veces' en cada iteración.
9. Modifica el código para que no se produzca el error (pista: condición de salida del bucle).
10. ¿Hay algún problema con el segundo bucle? Coloca un breakpoint al inicio del 'for'. ¿Qué valor tiene 'veces'? ¿Por qué?
11. Arregla el error para que se ejecuten ambos bloques el número de veces esperado.
12. Modifica el código para que la variable 'veces' la pueda seleccionar el usuario por teclado (input).
13. Crea un breakpoint condicional dentro del 'while' que se active cuando 'veces > 10' y otro breakpoint normal en el 'for'. Introduce primero 5 y luego 15. Describe qué ocurre con ambos puntos de ruptura.

Ejercicio 3 – Breakpoint condicional por umbral

Programa sencillo que realiza un bucle y va incrementando el valor de una variable.

14. Inserta un punto de ruptura condicional en la línea donde se actualiza 'valor' para que la ejecución se detenga cuando 'valor' sea superior a 10 millones.
15. ¿Qué valor tiene el contador ('contador') cuando se para el código? Justifica el resultado.

ENTREGA: capturas del depurador (variables y breakpoints) y breve memoria con explicación de errores encontrados y correcciones.