✅ Actividad 2 — Ejecutar script_1

- ¿En qué lenguaje está escrito?

  Python

- ¿Es compilado o interpretado?

  Interpretado

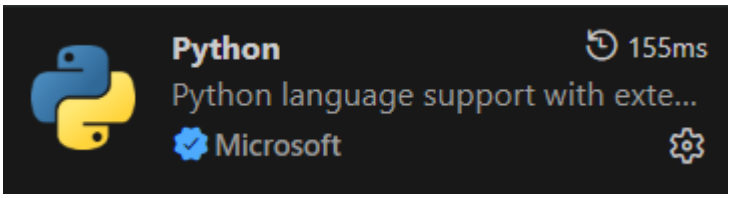
- ¿Se ejecuta directamente al instalar VS Code?

  No, tienes que descargar python

- ¿Qué tuviste que instalar para ejecutarlo?

  Python

- ¿Instalaste algún plugin? ¿Cuál?



2. **Capturas de pantalla** de cada paso.

✅ Actividad 3 — Investigar el lenguaje

1. Características del lenguaje

- Interpretado y de alto nivel.
- Tipado dinámico y fuerte.
- Sintaxis clara y legible, cercana al inglés.
- Multiplataforma (Windows, Linux, macOS).
- Soporta programación procedural, orientada a objetos y funcional.
- Amplia biblioteca estándar y ecosistema de paquetes externos.

2. Puntos fuertes

- Fácil de aprender y leer.
- Gran comunidad y documentación.
- Amplio soporte para ciencia de datos, web, automatización y scripting.
- Compatible con casi cualquier sistema operativo.
- Gran cantidad de librerías y frameworks (Django, Flask, Pandas, NumPy, TensorFlow…).

3. Puntos débiles

- Más lento que lenguajes compilados (C/C++/Java) en ejecución pura.
- Consumo de memoria relativamente alto.
- No ideal para aplicaciones móviles o videojuegos de alto rendimiento nativo.
- Global Interpreter Lock (GIL) limita hilos en CPU.

4. Usos actuales

- Desarrollo web y backend.
- Ciencia de datos, análisis y visualización.
- Inteligencia artificial y machine learning.
- Automatización de tareas y scripts.
- Educación y prototipado rápido.
- Desarrollo de aplicaciones de escritorio y APIs.

5. Instalar un plugin útil en VS Code

Plugin recomendado: **Python (Microsoft)**

Pasos:

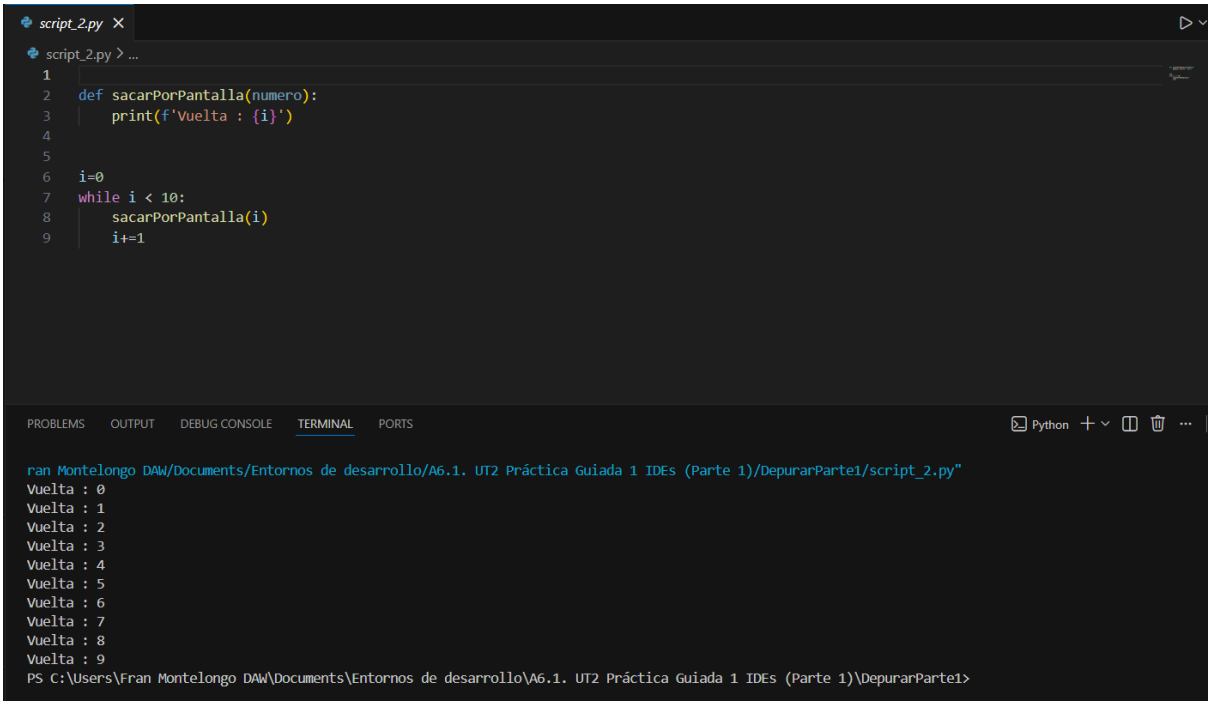
1. Abrir VS Code → Extensiones → Buscar `Python`.
2. Instalar el plugin oficial de Microsoft.
3. Reiniciar VS Code si es necesario.

6. Cómo estaba el entorno antes

- VS Code mostraba el código sin autocompletado avanzado.
- No había detección de errores en tiempo real.
- No podía ejecutar scripts directamente desde VS Code fácilmente.

7. Cómo funciona después del plugin

- Autocompletado de código, sugerencias de librerías y funciones.
- Resaltado de errores y advertencias en tiempo real.
- Posibilidad de ejecutar scripts de Python directamente.
- Soporte para entornos virtuales y depuración integrada.



✅ Actividad 4 — Depuración (Debug)

Aquí debes practicar con los scripts 1 al 7.

4.1 Script 1

Poner breakpoint en `print(f'Vuelta : {i}')`.

Ejecutar modo Debug.

Continue: sigue hasta el siguiente breakpoint o fin del programa.

Step Over: ejecuta la línea actual y pasa a la siguiente, sin entrar en funciones.

Step Into: si la línea llama una función, entra dentro de la función.

4.2 Script 2

Breakpoint: dentro de `sacarPorPantalla`.

Debug:

Step Into: entra en la función y verás que imprime `i` global (mal, debería ser `numero`).

Corrección: `print(f'Vuelta : {numero}')`.

4.3 Script 3

- **Error:** `IndexError` al acceder al último índice.
- **Debug paso a paso:** observa que `i` llega a 5.
- **Corrección:** `for i in range(len(colores)):`

4.4 Script 4

- **Errores:** `IndexError` + `TypeError` (string + int).
- **Debug:** Step Over para ver dónde falla.
- **Corrección:**

```
for i in range(len(colores)):
    print(colores[i])
    resultado = colores[i] + "1"  # si quieres concatenar como string
```

4.5 Script 5

- **Debug:** observar valores de `resultado_intermedio`, `resultado_intermedio_2`, `resultado_final`.
- **Step Into:** entra en la función.
- **Step Over:** ejecuta la línea actual sin entrar en funciones internas.

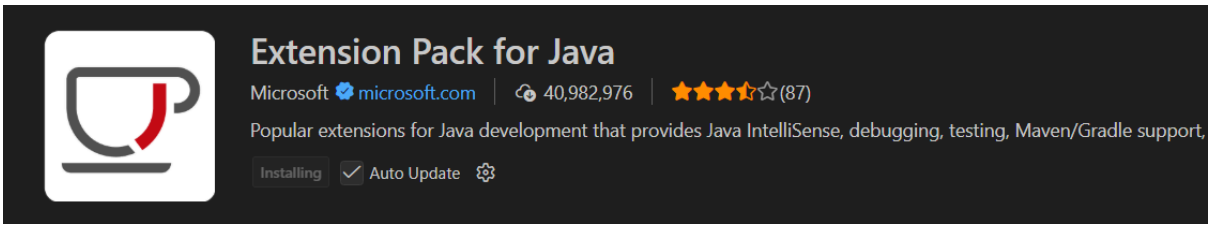
4.6 Script 6

- Introducir números → funciona correctamente.
- Introducir letras → `ValueError` al convertir a int.
- se podría añadir `try/except` para manejar entradas no numéricas.

4.7 Script 7

- Funciona con números.
- Letras en número → captura con `try/except`.
- Mayúsculas (Y/N) → `lower()` corrige el problema.
- Debug permite verificar flujo y valores de `p1` y `p2`.

Actividad 5 — Configurar Java en VS Code



Poner **breakpoints** en `System.out.println(i)` y depurar con:

- **Step Over:** ejecuta la línea actual.
- **Step Into:** entra en métodos si los hubiera.
- **Step Out:** sale de método actual.

Actividad 6 — Compilar Java

- Se genera `Principal.class` → bytecode listo para ejecutar en JVM.
- Abrir `.class` con un editor → símbolos raros porque no es texto legible, es bytecode.
- `javac` traduce código fuente `.java` a bytecode `.class`.

Actividad 7 — IntelliJ IDEA

- Instalar IntelliJ IDEA (Community o Ultimate).
- Abrir carpeta `Depurar1`.
- Ejecutar `Principal.java` → funciona igual que en VS Code.
- Depurar con breakpoints.
- Opciones similares a VS Code: Step Over, Step Into, Step Out, Run to Cursor, Variables y Watches.

Actividad 8 — Depurar2 y Depurar3

Depurar2

- **Error:** `i` no existe fuera del for → Scope limitado.
- **Solución:** declarar `int i = 0;` antes del for o imprimir solo dentro del for.

Depurar3

Sin errores si `texto = "hola"`.

Si `texto = null` → `NullPointerException`.

Debug paso a paso permite ver `texto` y `longitud`.

Actividad 9 — Depurar4

- **Error:** `ArrayIndexOutOfBoundsException`.
- **Solución:**

  Build Project en IntelliJ: genera `.class` en `out/production/...`

  Bytecode generado → se ejecuta en JVM.