1. **¿Qué es el principio SRP (Single Responsibility Principle)?**

El **principio SRP (Single Responsibility Principle)** es uno de los cinco principios fundamentales de la programación orientada a objetos, conocidos como SOLID. Este principio establece que una clase o módulo debe tener una única responsabilidad, es decir, debe enfocarse en una sola tarea o función específica. El propósito de aplicar SRP es hacer que el código sea más fácil de **mantener**, **entender** y **modificar**. Al centrarse en una única preocupación, el código también se vuelve más **modular** y **testable**, lo que simplifica las pruebas y la depuración.

2. **¿Por qué es importante validar la entrada del usuario? Relaciónalo con OWASP.**

La **validación de la entrada del usuario** es crucial para proteger una aplicación contra posibles **vulnerabilidades de seguridad**. Si no validamos adecuadamente los datos ingresados, podemos ser susceptibles a ataques como **inyección SQL**, **XSS (Cross-Site Scripting)**, o la ejecución de código malicioso. La organización **OWASP (Open Web Application Security Project)**, que se especializa en mejorar la seguridad de las aplicaciones web, destaca que uno de los principales enfoques para proteger las aplicaciones es **validar todas las entradas** de los usuarios. Esto asegura que solo se procesen datos esperados y seguros, evitando que actores malintencionados inyecten información que pueda comprometer la seguridad de la aplicación.

3. **¿Qué beneficios tiene separar el código en funciones o clases?**

Separar el código en **funciones** o **clases** ofrece múltiples ventajas:

- **Reusabilidad**: Al modularizar el código, se facilita su reutilización en distintas partes de la aplicación, reduciendo la duplicación.
- **Mantenibilidad**: Cada función o clase tiene una responsabilidad claramente definida, lo que facilita la identificación de errores y la implementación de cambios sin afectar otras partes del sistema.
- **Escalabilidad**: El código bien estructurado es más fácil de extender sin generar efectos secundarios no deseados en otras partes del proyecto.
- **Facilita la colaboración**: Al dividir el código en unidades pequeñas, diferentes desarrolladores pueden trabajar en diferentes funciones o clases de manera independiente, lo que mejora la **productividad** y reduce los conflictos.

4. **¿Qué es Clean Code y cómo se diferencia de un código que solo "funciona"?**

Clean Code es un conjunto de prácticas de programación que buscan producir código **legible**, **bien estructurado** y **fácil de mantener**. Se caracteriza por su organización, el uso de **nombres descriptivos** para variables y funciones, y la aplicación de principios que mejoran su comprensión y extensión. A diferencia de un código que "solo funciona", que puede ser **funcional** pero desordenado, difícil de leer o sin comentarios, **Clean Code** está diseñado con vistas al **futuro**. Se enfoca en la **sostenibilidad** del código, su **colaboración** entre desarrolladores y su facilidad de depuración. En resumen, un código limpio no solo cumple su función, sino que también es fácil de modificar, extender y probar con el tiempo.

5. **Enumera tres errores comunes que debemos evitar al pedir datos por consola.**

   No validar la entrada del usuario: Es esencial asegurarse de que los datos ingresados sean correctos y correspondan al tipo esperado. De lo contrario, pueden ocurrir **errores de ejecución** o **vulnerabilidades** en el programa.

   No manejar excepciones o errores: Si el usuario ingresa un dato inesperado, como texto cuando se esperaba un número, el programa podría fallar. Es fundamental implementar **manejo de excepciones** y proporcionar **mensajes claros** para que el usuario pueda corregir la entrada.

   No proporcionar instrucciones claras: Cuando las instrucciones no son lo suficientemente específicas, el usuario puede ingresar datos incorrectos. Por ejemplo, si se pide un número de teléfono, especificar claramente el formato esperado evita entradas erróneas y mejora la interacción.