**1. Análisis del Programa**

Tareas Repetitivas o que pueden encapsularse en funciones

Se identificaron las siguientes tareas que se repiten o que pueden organizarse en funciones separadas:

Agregar estudiantes: Se repiten las mismas instrucciones para pedir nombre y calificación y luego almacenarlas.

Mostrar la lista de estudiantes: Se recorre la lista para imprimir los datos.

Calcular promedios: Se suman todas las calificaciones y se divide por la cantidad de estudiantes.

Encontrar al estudiante con la calificación más alta: Se busca la calificación más alta y se imprime el nombre del estudiante correspondiente.

Al convertir estas tareas en funciones, el código se vuelve más limpio y fácil de leer.

**Variables Locales vs Variables Globales**

Uso de variables globales

Las variables globales son útiles para almacenar datos que deben ser accesibles en todo el programa. En este caso, las siguientes variables pueden ser globales:

**List<string> estudiantes**: Almacena los nombres de los estudiantes.

**List<double> calificaciones**: Almacena las calificaciones correspondientes.

Ambas listas deben ser accesibles desde varias funciones, por lo que es recomendable declararlas fuera de las funciones.

Uso de variables locales

Las variables locales deben usarse dentro de funciones cuando los datos solo son relevantes en un contexto específico. Ejemplos:

**double suma = 0** dentro de la función **CalcularPromedio()** Solo se usa para sumar las calificaciones temporalmente.

**double maxCalificacion = 0** dentro de la función **MostrarEstudianteConMaxCalificacion()** Se usa solo para encontrar la mejor nota.

Limitar el uso de variables globales reduce el riesgo de errores y hace que el código sea más seguro y fácil de depurar.

**2. Modularización**

Convertir el Programa en Funciones

El programa se puede dividir en funciones para mejorar su organización. Se recomienda crear las siguientes funciones:

**AgregarEstudiante()**

Solicita el nombre y la calificación de un estudiante.

Valida que la calificación sea un número válido antes de agregarlo a la lista.

**MostrarEstudiantes()**

Recorre la lista e imprime los estudiantes con sus calificaciones.

**CalcularPromedio()**

Suma todas las calificaciones y las divide entre el número total de estudiantes.

**MostrarEstudianteConMaxCalificacion()**

Encuentra al estudiante con la calificación más alta y lo imprime.

Definir Variables Locales y Globales

Variables globales:

**List<string> estudiantes**

**List<double> calificaciones**

**Variables locales:**

**double suma = 0** en **CalcularPromedio()**

**double maxCalificacion = 0** en **MostrarEstudianteConMaxCalificacion()**

**int opcion** dentro del **while** del menú

El uso de variables locales dentro de las funciones evita conflictos y errores al modificar datos globales accidentalmente.

**3. Preguntas Guía**

1. **¿Qué ventajas tiene dividir el código en funciones?**

* **Reutilización:** Podemos llamar las funciones varias veces sin repetir código.
* **Legibilidad:** El código es más limpio y fácil de entender.
* **Mantenimiento:** Si hay un error en el cálculo del promedio, solo se corrige en una función en lugar de en varios lugares.
* **Depuración:** Es más fácil encontrar errores en funciones pequeñas que en un código grande y desorganizado.

**2. ¿Por qué es importante limitar el uso de variables globales?**

* **Mayor riesgo de errores:** Cualquier función puede modificar una variable global accidentalmente.
* **Dificultad en la depuración:** Si un error cambia un valor inesperadamente, es más difícil encontrar el problema.
* **Menos reutilización:** Funciones con muchas variables globales pueden ser menos flexibles para usarlas en otros contextos.

Se recomienda que las variables globales solo almacenen datos esenciales y que las funciones trabajen con variables locales cuando sea posible.

**3. ¿Cómo se puede mejorar la legibilidad del código?**

* **Nombres descriptivos:** En lugar de suma, usar **sumaCalificaciones.**
* **Comentarios claros:** Explicar qué hace cada función y cada bloque de código importante.
* **Estructura ordenada:** Separar las funciones y dejar espacios entre bloques para una mejor visualización.
* **Formato correcto:** Usar sangrías y alineaciones adecuadas.

**4. Mejoras Adicionales**

Validación de Entradas del Usuario

Para evitar errores cuando el usuario ingresa texto en lugar de números, se puede mejorar el código con validaciones.

**Ejemplo:**

**Console.Write("Ingrese la calificación del estudiante: ");**

**if (!double.TryParse(Console.ReadLine(), out double calificacion) || calificacion < 0 || calificacion > 100)**

**{**

**Console.WriteLine("Error: Ingrese una calificación válida entre 0 y 100.");**

**return;**

**}**

Esto evita que el programa falle si el usuario ingresa texto o valores fuera de rango.

Conclusión

* Se mejoró la estructura del programa dividiéndolo en funciones.
* Se analizaron las diferencias entre variables globales y locales.
* Se destacó la importancia de modularizar el código para mejorar su mantenimiento.
* Se agregaron validaciones para evitar errores en las entradas del usuario.

Con estas mejoras, el programa es más eficiente, claro y fácil de mantener.