

**Laboratorio 8**  
**Exploración y Uso Avanzado de Plataformas IA, Repositorios**  
**Profesionales y Herramientas Globales para el desarrollo de IA**  
**y de SW**

**Materia:**

Profundización de inteligencia artificial

**Participantes:**

Ana Maria Navarro Fernandez

**Profesor:**

Carlos Betancourt correa

Universidad de Manizales  
ingeniería en sistemas y telecomunicaciones  
Manizales, Caldas, Colombia

## 2.13 Claude Code



### 1. Introducción

En esta actividad se utilizó Claude Code para analizar y mejorar un archivo de Python. El objetivo fue comprobar la capacidad de Claude para detectar errores, sugerir optimizaciones y aplicar buenas prácticas de programación.

### 2. Archivo original

```
codigo_original.py X
C: > Users > Ana Navarro > Downloads > codigo_original.py > ...
1 def calcular_promedio(notas):
2     suma = 0
3     for n in notas:
4         suma = suma + n
5     promedio = suma / len(notas)
6     return promedio
7
8 notas = [3.5, 4.0, 2.8, 3.9]
9 print(calcular_promedio(notas))
10
```

Se partió del archivo `codigo\_original.py`, el cual contenía una función simple para calcular el promedio de una lista de notas. El código funcionaba, pero presentaba problemas de legibilidad, uso de bucles innecesarios y falta de validación.

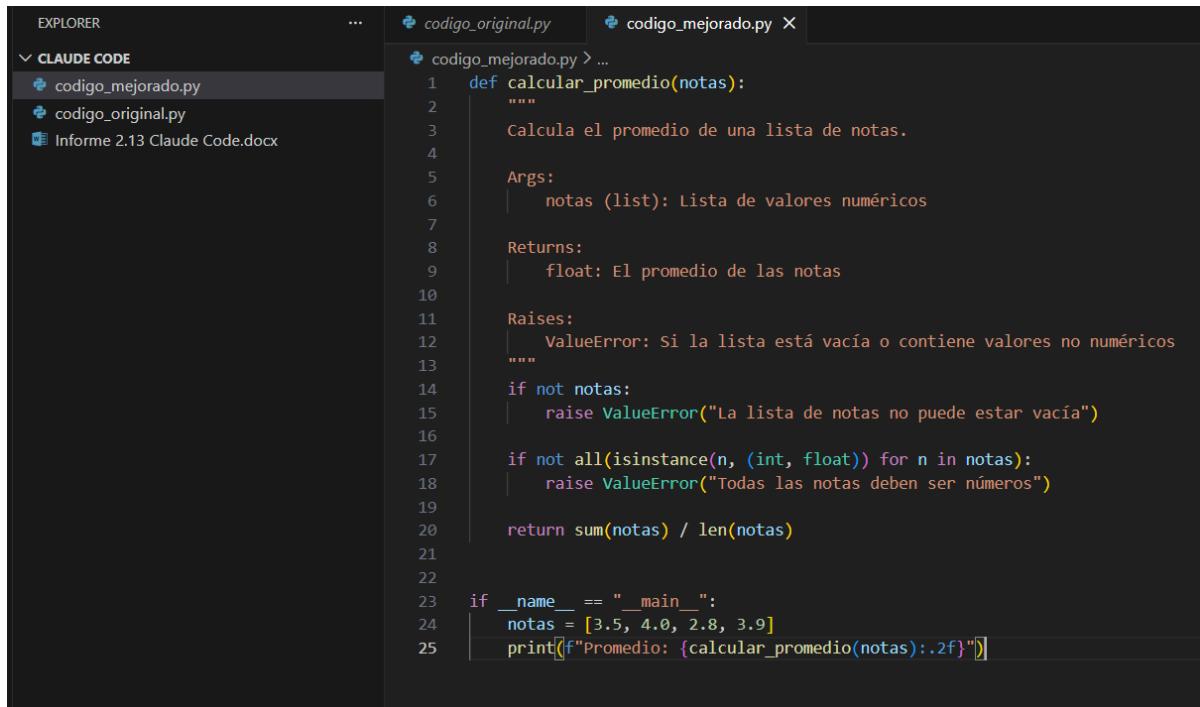
### 3. Análisis de Claude

Claude identificó los siguientes puntos:

- Uso manual de sumas que puede reemplazarse por `sum()`.
- Falta de validación para listas vacías.
- Falta de estructura en el bloque principal.
- No se formateaba la salida.

- Pocas o nulas buenas prácticas (PEP8).

#### 4. Código mejorado



```

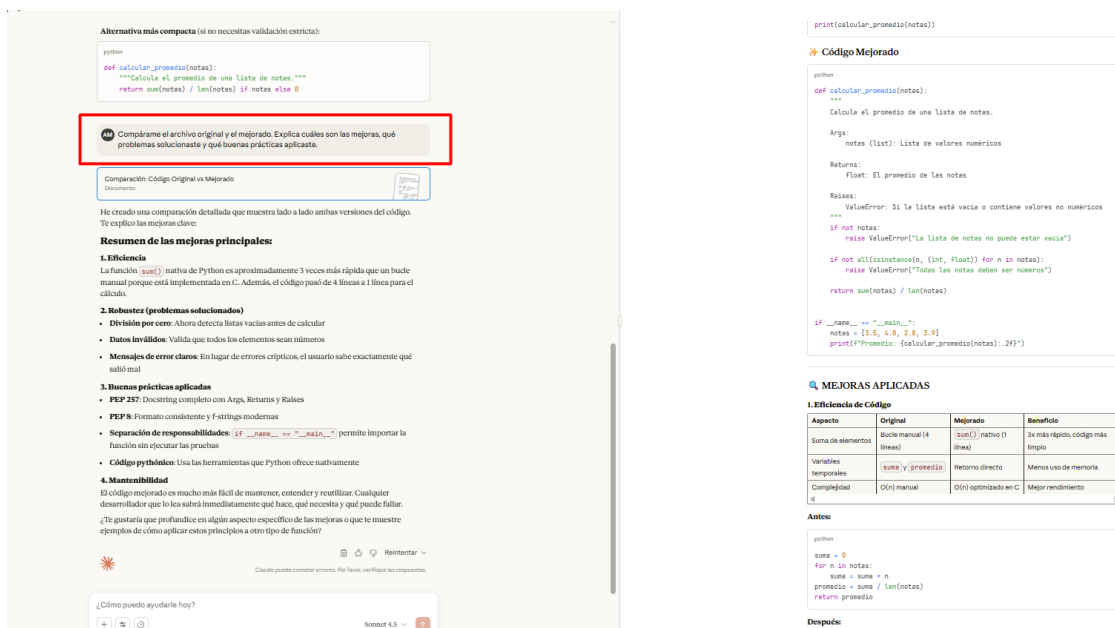
EXPLORER
  CLAUDE CODE
    codigo_mejorado.py
    codigo_original.py
    Informe 2.13 Claude Code.docx

codigo_mejorado.py > ...
1  def calcular_promedio(notas):
2      """
3      Calcula el promedio de una lista de notas.
4
5      Args:
6      |   notas (list): Lista de valores numéricos
7
8      Returns:
9      |   float: El promedio de las notas
10
11     Raises:
12     |   ValueError: Si la lista está vacía o contiene valores no numéricos
13     """
14     if not notas:
15         raise ValueError("La lista de notas no puede estar vacía")
16
17     if not all(isinstance(n, (int, float)) for n in notas):
18         raise ValueError("Todas las notas deben ser números")
19
20     return sum(notas) / len(notas)
21
22
23 if __name__ == "__main__":
24     notas = [3.5, 4.0, 2.8, 3.9]
25     print(f"Promedio: {calcular_promedio(notas):.2f}")
  
```

Claude generó una versión optimizada, guardada como `codigo\_mejorado.py`, que incluye:

- Validación de entradas.
- Reducción del código a una línea usando `sum()`.
- Estructura `if \_\_name\_\_ == "\_\_main\_\_":`
- Formateo profesional de salida.
- Comentarios y nombres claros.

#### 5. Comparación entre versiones



**Alternativa más compacta** (si no necesitas validación estricta):

```
python
def calcular_promedio(notas):
    """Calcula el promedio de una lista de notas."""
    return sum(notas) / len(notas) if notas else 0
```

**Comparación: Código Original vs. Mejorado**

He creado una comparación detallada que muestra lado a lado ambas versiones del código. Te explico las mejoras clave:

**Resumen de las mejoras principales:**

- 1. Eficiencia:** La función `sum()` nativa de Python es aproximadamente 3 veces más rápida que un bucle manual porque está implementada en C. Además, el código pasó de 4 líneas a 1 línea para el cálculo.
- 2. Robustez (problemas solucionados):**
  - División por cero:** Ahora detecta listas vacías antes de calcular.
  - Datos inválidos:** Valida que todos los elementos sean números.
  - Mensajes de error claros:** En lugar de errores crípticos, el usuario sabe exactamente qué salió mal.
- 3. Buenas prácticas aplicadas:**
  - PEP 257:** Docstring completo con Args, Returns y Raises.
  - PEP 8:** Formato consistente y f-strings modernas.
  - Separación de responsabilidades:** `if __name__ == "__main__":` permite importar la función sin ejecutar las pruebas.
  - Código pythonico:** Usa las herramientas que Python ofrece nativamente.
- 4. Mantenibilidad:** El código mejorado es mucho más fácil de mantener, entender y reutilizar. Cualquier desarrollador que lo lea sabrá inmediatamente qué hace, qué necesita y qué puede fallar. ¿Te gustaría que profundice en algún aspecto específico de las mejoras o que te muestre ejemplos de cómo aplicar estos principios a otro tipo de función?

**Código Mejorado**

```
python
def calcular_promedio(notas):
    """
    Calcula el promedio de una lista de notas.

    Args:
        notas (list): Lista de valores numéricos

    Returns:
        float: El promedio de las notas

    Raises:
        ValueError: Si la lista está vacía o contiene valores no numéricos
    """
    if not notas:
        raise ValueError("La lista de notas no puede estar vacía")

    if not all(isinstance(n, (int, float)) for n in notas):
        raise ValueError("Todas las notas deben ser números")

    return sum(notas) / len(notas)

if __name__ == "__main__":
    notas = [3.5, 4.0, 2.8, 3.9]
    print(f"Promedio: {calcular_promedio(notas):.2f}")
```

**MEJORAS APLICADAS**

**1. Eficiencia de Código**

Aspecto	Original	Mejorado	Beneficio
Suma de elementos	Bucle manual (4 líneas)	<code>sum()</code> nativo (1 línea)	3x más rápido, código más limpio
Variables temporales	<code>suma</code> y <code>promedio</code>	Retorno directo	Menos uso de memoria
Complejidad	O(n) manual	O(n) optimizado en C	Mejor rendimiento

**Antes:**

```
python
suma = 0
for n in notas:
    suma = suma + n
promedio = suma / len(notas)
return promedio
```

**Después:**

```
python
def calcular_promedio(notas):
    return sum(notas) / len(notas) if notas else 0
```

Claude señaló mejoras sustanciales en:

- Eficiencia
- Legibilidad
- Claridad de funciones
- Buenas prácticas
- Robustez ante errores

### 6. Conclusiones

Claude Code demostró ser una herramienta útil para revisión y optimización rápida de código. Puede detectar problemas comunes de estilo, lógica y estructura. Sin embargo, no reemplaza la revisión humana en casos complejos, especialmente cuando se requiere conocimiento del contexto del proyecto o validación profunda.

Analizo tu código de Python:

### Análisis del código

#### ✓ Lo que está bien

- El código funciona correctamente y calcula el promedio
- La lógica es clara y directa
- Los nombres de variables son descriptivos en español

#### ⚠ Posibles errores

1. **División por cero:** Si `notas` está vacía, `len(notas)` será 0 y causará un `ZeroDivisionError`
2. **Validación de entrada:** No verifica que los elementos sean números

#### 🔧 Mejoras de eficiencia

1. Usar `sum()` integrado: Python tiene una función nativa más eficiente que el bucle manual
2. Evitar operaciones innecesarias: La acumulación manual es menos óptima

#### 📖 Problemas de legibilidad

1. **Falta docstring:** No hay documentación de qué hace la función
2. **Falta validación explícita:** No es obvio qué pasa con listas vacías
3. **Código suelto:** Las pruebas deberían estar en un bloque `if __name__ == "__main__":`

#### 🌟 Buenas prácticas recomendadas

```
python
def calcular_promedio(notas):
    """
    Calcula el promedio de una lista de notas.

    Args:
        notas (list): Lista de valores numéricos

    Returns:
        float: El promedio de las notas

    Raises:
        ValueError: Si la lista está vacía o contiene valores no numéricos
    """
    if not notas:
        raise ValueError("La lista de notas no puede estar vacía")

    if not all(isinstance(n, (int, float)) for n in notas):
        raise ValueError("Todas las notas deben ser números")

    return sum(notas) / len(notas)
```

¿Cómo puedo ayudarle hoy?



Sonnet 4.5



Analizo tu código de Python:

### Análisis del código

#### ✓ Lo que está bien

- El código funciona correctamente y calcula el promedio
- La lógica es clara y directa
- Los nombres de variables son descriptivos en español

#### ⚠ Posibles errores

1. **División por cero:** Si `notas` está vacía, `len(notas)` será 0 y causará un `ZeroDivisionError`
2. **Validación de entrada:** No verifica que los elementos sean números

#### 🔧 Mejoras de eficiencia

1. Usar `sum()` integrado: Python tiene una función nativa más eficiente que el bucle manual
2. Evitar operaciones innecesarias: La acumulación manual es menos óptima

#### 📖 Problemas de legibilidad

1. **Falta docstring:** No hay documentación de qué hace la función
2. **Falta validación explícita:** No es obvio qué pasa con listas vacías
3. **Código suelto:** Las pruebas deberían estar en un bloque `if __name__ == "__main__":`

#### 🌟 Buenas prácticas recomendadas

```
python
def calcular_promedio(notas):
    """
    Calcula el promedio de una lista de notas.

    Args:
        notas (list): Lista de valores numéricos

    Returns:
        float: El promedio de las notas

    Raises:
        ValueError: Si la lista está vacía o contiene valores no numéricos
    """
    if not notas:
        raise ValueError("La lista de notas no puede estar vacía")

    if not all(isinstance(n, (int, float)) for n in notas):
        raise ValueError("Todas las notas deben ser números")

    return sum(notas) / len(notas)
```

¿Cómo puedo ayudarle hoy?



Sonnet 4.5

