UNIVERSIDAD AUTÓNOMA DE YUCATÁN

FACULTAD DE MATEMATICAS

Semestre:

Agosto-diciembre

Materia:

Desarrollo Y Mantenimiento de Software

Proyecto Final:

Parte 2

Docente:

M. en C. Carlos Benito Mojica Ruiz.

Alumnos:

Santiago Efrain Itzincab Poot

Rogerio Emmanuel Canto Romero

Eduardo Alberto Gonzalez Ortega

Jafet Andree Mena Solis

Contenido

Unidad #1

Manual de usuario

Este manual proporciona las instrucciones necesarias para utilizar el script que analiza un archivo Python, valida estándares de codificación, y cuenta las líneas físicas y lógicas.

Requisitos Previos

1. Sistema y Herramientas:

- Sistema operativo con Python 3.6 o superior instalado.
- Biblioteca tabulate instalada (puede instalarla usando: pip install tabulate).

2. Archivos de Entrada:

- Un archivo Python (.py) que será analizado.

3. Funcionalidades del Script:

- Verifica estándares de codificación comunes (longitud de líneas, indentación, comentarios, etc.).
- Identifica funciones y bloques principales en el código.
- Cuenta las líneas físicas y lógicas del archivo.
- Presenta un resumen detallado en formato tabular.

Cómo Usar el Script

1. Ejecución Inicial

- Guarde el script en un archivo .py (por ejemplo, analizador_codigo.py).
- Ejecute el script desde la terminal o un IDE:

python analizador_codigo.py

2. Selección de Archivo

- Aparecerá una ventana de selección de archivos.
- Seleccione el archivo Python que desea analizar.
- Si no selecciona un archivo, el programa mostrará un mensaje y se cerrará.

3. Resultados

El script genera los siguientes resultados en la consola:

1. Verificación de Estándares:

- Identifica errores relacionados con:
- Longitud de líneas (>79 caracteres).
- Formato de comentarios.
- Indentación (múltiplos de 4 espacios).
- Identificadores (nombres en minúscula).
- Estructuras de control y definiciones de funciones.

2. Resumen de Funciones y Bloque Principal:

- Muestra las siguientes columnas en formato tabular:
- Programa: Identifica si es una función o el bloque principal.
- Función / Procedimiento: Nombre de la función (si aplica).
- Total de métodos: Número de funciones en el programa.
- LOC físicas: Cantidad de líneas físicas no vacías.
- LOC lógicas: Cantidad de instrucciones clave.

3. Totales del Código Completo:

- Total de líneas físicas y lógicas en el programa.

Flujo de Trabajo del Usuario

1. Inicio:

- Ejecute el script y seleccione el archivo Python a analizar.

2. Análisis:

- El script analiza el archivo línea por línea para verificar estándares y contar líneas.

3. Resultados:

- Verifique en la consola los errores detectados, el desglose de funciones, y los totales de líneas.

Ejemplo de Salida

Archivo Analizado:

Proyecto1.py

Salida Generada:

```
PS C:\Users\EduardoAlbertoGonzal\Downloads\mantenimiento> & C:/Users/EduardoAlbertoGonzal/AppData/Local/Programs/Python/Python313/python.exe c:/Users/EduardoAlbertoGonzal/Downloads/mantenimiento/Proyecto2.py
El archivo cumple con los estándares.
+------------------+-----------------------------+--------------------+---------------+---------------+
|     Programa     |   Función / Procedimiento   |  Total de métodos  |   LOC físicas |   LOC lógicas |
+==================+=============================+====================+===============+===============+
|    Función 1     | Verificar_estandares_archivo |         1          |            62 |            28 |
+------------------+-----------------------------+--------------------+---------------+---------------+
|    Función 2     |    Contar_lineas_logicas    |         1          |            21 |             8 |
+------------------+-----------------------------+--------------------+---------------+---------------+
|    Función 3     |    Contar_lineas_fisicas    |         1          |            13 |             5 |
+------------------+-----------------------------+--------------------+---------------+---------------+
| Bloque Principal |                             |                    |            20 |             3 |
+------------------+-----------------------------+--------------------+---------------+---------------+

Totales del código completo:
Líneas físicas: 119
Líneas lógicas: 44
```

Mensajes de Error Comunes

1. Error: "No se pudo encontrar el archivo"

- Asegúrese de seleccionar un archivo existente.

2. Excepciones Personalizadas:

- Ejemplo: Línea 3: Indentación incorrecta, debe ser múltiplo de 4 espacios.

- Descripción: Detalla el problema y la línea afectada.

Preguntas Frecuentes (FAQ)

1. ¿Qué es una línea lógica?

- Es una línea que incluye instrucciones clave como if, for, def, etc.

2. ¿Qué hacer si encuentro un error en el archivo?

- Revise el mensaje del error y corríjalo en el archivo Python.

3. ¿Se pueden analizar varios archivos a la vez?

- Actualmente, el script solo permite seleccionar un archivo a la vez.

Manual Técnico

Este manual técnico describe en detalle la estructura, funcionalidad y lógica del script de análisis de código Python. Este documento está destinado a desarrolladores y técnicos que necesiten entender, mantener o modificar el código.

1. Propósito del Script

El script permite:

- Verificar estándares de codificación en un archivo Python.
- Identificar funciones y bloques principales en el código.
- Contar líneas físicas y lógicas de código.
- Presentar resultados en un formato tabular.

Estándares Validados

- Longitud de líneas (<80 caracteres).
- Formato de comentarios.
- Indentación (múltiplos de 4 espacios).
- Uso de identificadores en minúsculas.
- Validez de nombres de funciones.
- Correcto uso de estructuras de control.

2. Descripción Técnica

2.1. Estructura del Código

1. Importaciones

- tkinter: Usado para manejar la selección de archivos mediante una GUI.
- filedialog: Permite al usuario seleccionar un archivo desde un cuadro de diálogo.

- tabulate: Genera tablas para los resultados.
- re: Procesa expresiones regulares para la validación de comentarios y estructuras.

2. Funciones Principales

- Verificar_estandares_archivo(archivo):

Valida estándares de codificación línea por línea.

- Leer_codigo(filename):

Divide el archivo en funciones y el bloque principal.

- Contar_lineas_fisicas(codigo):

Cuenta las líneas no vacías ni comentarios.

- Contar_lineas_logicas(codigo):

Cuenta líneas que representan instrucciones clave.

- Imprimir_bloques(funciones, bloque_principal):

Muestra un resumen tabular de las funciones y el bloque principal.

- Contar_lineas_totales(codigo):

Calcula el total de líneas físicas y lógicas del programa.

3. Bloque Principal (if __name__ == "__main__":)

- Gestiona la interacción con el usuario y ejecuta las funciones anteriores.

2.2. Detalle de Funciones

a) Verificar_estandares_archivo(archivo)

- Entrada: Ruta de un archivo .py.
- Procesos:

- Valida longitud de línea, formato de comentarios, indentación, uso de identificadores, y estructuras de control.

- Salida:
  - Mensaje en consola si el archivo cumple estándares o una excepción detallada.

b) Leer_codigo(filename)

- Entrada: Ruta de un archivo .py.
- Procesos:
  - Lee el archivo línea por línea.
  - Clasifica el código en funciones y bloque principal.
- Salida:
  - Lista de funciones con sus líneas y bloque principal.

c) Contar_lineas_fisicas(codigo)

- Entrada: Lista de líneas de código.
- Procesos:
  - Ignora comentarios y líneas vacías.
- Salida: Número total de líneas físicas.

d) Contar_lineas_logicas(codigo)

- Entrada: Lista de líneas de código.
- Procesos:
  - Identifica líneas que comienzan con palabras clave (if, for, while, etc.).
- Salida: Número total de líneas lógicas.

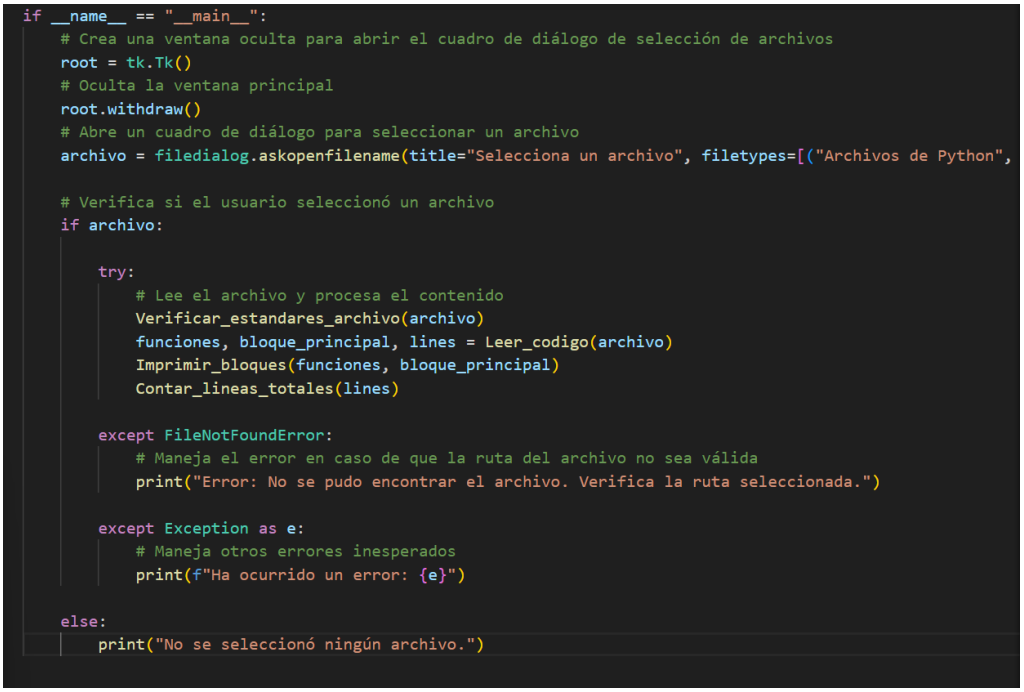
e) Imprimir_bloques(funciones, bloque_principal)

- Entrada: Listas de funciones y bloque principal.
- Procesos:
  - Calcula LOC físicas y lógicas para cada función y bloque.
  - Genera una tabla con los datos.
- Salida: Tabla tabulada impresa en consola.

f) Contar_lineas_totales(codigo)

- Entrada: Lista de líneas del archivo completo.
- Procesos:
  - Llama a las funciones de conteo para líneas físicas y lógicas.
- Salida: Totales de líneas físicas y lógicas impresos en consola.

2.3. Lógica del Bloque Principal



```python
if __name__ == "__main__":
    # Crea una ventana oculta para abrir el cuadro de diálogo de selección de archivos
    root = tk.Tk()
    # Oculta la ventana principal
    root.withdraw()
    # Abre un cuadro de diálogo para seleccionar un archivo
    archivo = filedialog.askopenfilename(title="Selecciona un archivo", filetypes=[("Archivos de Python",

    # Verifica si el usuario seleccionó un archivo
    if archivo:

        try:
            # Lee el archivo y procesa el contenido
            Verificar_estandares_archivo(archivo)
            funciones, bloque_principal, lines = Leer_codigo(archivo)
            Imprimir_bloques(funciones, bloque_principal)
            Contar_lineas_totales(lines)

        except FileNotFoundError:
            # Maneja el error en caso de que la ruta del archivo no sea válida
            print("Error: No se pudo encontrar el archivo. Verifica la ruta seleccionada.")

        except Exception as e:
            # Maneja otros errores inesperados
            print(f"Ha ocurrido un error: {e}")

    else:
        print("No se seleccionó ningún archivo.")
```

3. Herramientas y Configuración

3.1. Requisitos de Software

- Python 3.6 o superior.
- Bibliotecas requeridas:
    - tabulate: Instalar con pip install tabulate.

4. Lógica de Validación

4.1. Validación de Comentarios

- Debe haber un espacio después de #.
- Los comentarios deben iniciar con una mayúscula.

4.2. Validación de Identificadores

- Nombres de variables deben estar en minúsculas.

4.3. Validación de Indentación

- La indentación debe ser un múltiplo de 4 espacios.

5. Salida del Script

a. Verificación de Estándares:

- Mensaje en consola indicando problemas o confirmando que los estándares se cumplen.

b. Resumen Tabular:

Ejemplo:

Programa	Función / Procedimiento	Total de métodos	LOC físicas	LOC lógicas
Función 1	Verificar_estandares_archivo	1	62	28
Función 2	Contar_lineas_logicas	1	21	8
Función 3	Contar_lineas_fisicas	1	13	5
Bloque Principal			20	3

c. Totales del Código Completo:

- Líneas físicas y lógicas totales.

6. Posibles Mejoras

- Soporte para múltiples archivos: Analizar varios archivos en una sola ejecución.
- Interfaz gráfica completa: Reemplazar la interacción por consola con una GUI.

Unidad #2

Casos de prueba

Los casos de prueba están diseñados para verificar que el código funcione sin errores, tal y como se debe esperar en diferentes situaciones. Así como que se verifique el funcionamiento del programa por parte del usuario.

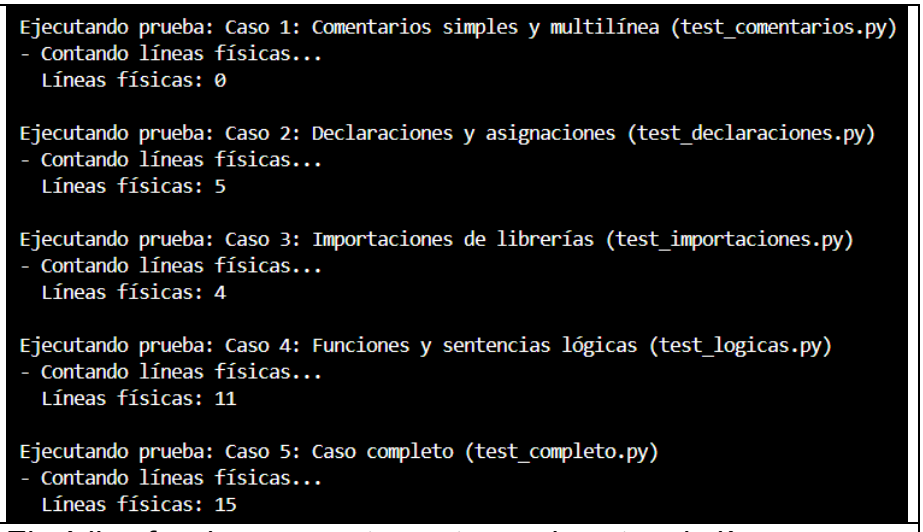
El siguiente cuadro nos da una forma de registro de los casos de prueba:

Tipo de prueba	
Nombre de la prueba/Numero objetivo de la prueba	Identificar único para la prueba
Descripción de la prueba	Describe las entradas y el procesamiento que va a tener el programa.
Condiciones de la prueba	Menciona las herramientas que se utilizaron para llevar a cabo en la prueba.
Resultados esperados	Lista de los resultados que se esperan obtener.
Resultados actuales	Lista de los resultados que se produjo durante la prueba.
Comentarios	Describe posibles comentarios que puedan ayudar a entender las pruebas.
Estado	Indicar el estado de la prueba: - Exitosa 1 - Fallida 0

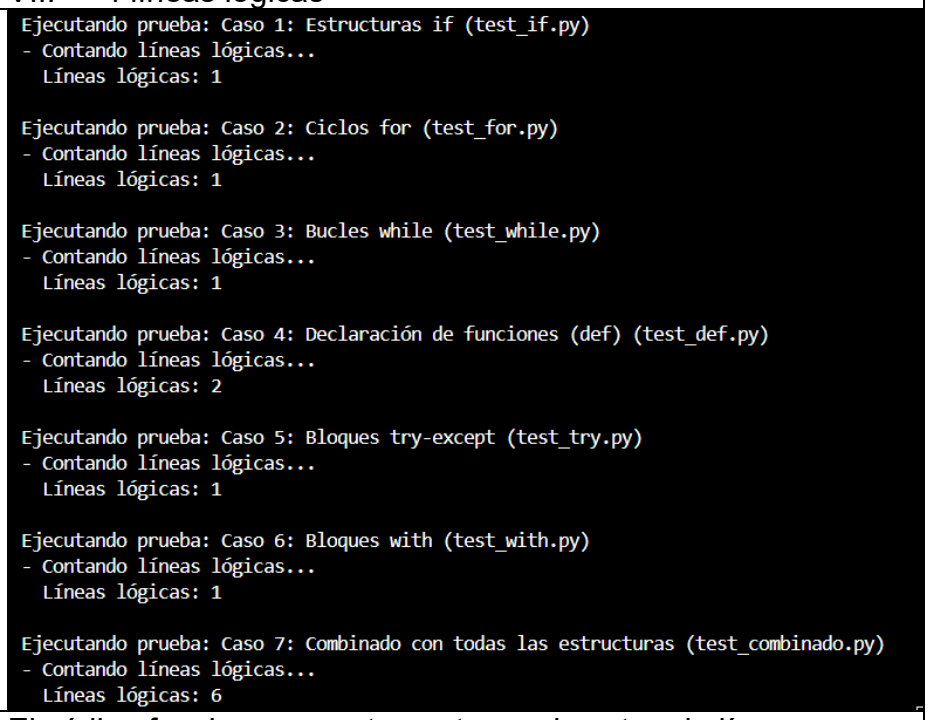
Pruebas Unitarias:

Para los módulos de contar líneas lógicas y el de líneas físicas, usaremos los test del proyecto anterior para verificar que funcionen correctamente después de las modificaciones aplicadas.

Tipo de prueba Prueba unitaria	
Nombre de la prueba/Numero objetivo de la prueba	P1 / Prueba_A Validar que el código sigue correctamente el estándar de conteo de líneas físicas.
Descripción de la prueba	El programa evalúa 5 escenarios predefinidos, diseñados para verificar los casos más relevantes relacionados con líneas físicas. **1. Comentarios:** Solo comentarios simples Se espera que no sean detectados como líneas físicas. **2. Declaraciones:** Declaraciones de variables, asignaciones e impresiones en consola, ignorando comentarios. **3. Importaciones:** Involucra importaciones de librerías o archivos externos, que deben contarse como líneas físicas. **4. Lógicas:** Contiene funciones, sentencias lógicas, impresiones en consola, docstrings y comentarios, sin afectar el conteo. **5. Completo:** Combina todos los escenarios previos en un solo caso, validando la consistencia en presencia de múltiples casos.
Condiciones de la prueba	Un script test4.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una.
Resultados esperados	1. 0 líneas físicas 2. 5 líneas físicas 3. 4 líneas físicas 4. 11 líneas físicas 5. 15 líneas físicas

Resultados actuales	
Comentarios	El código funciona correctamente en el conteo de líneas físicas.
Estado	1

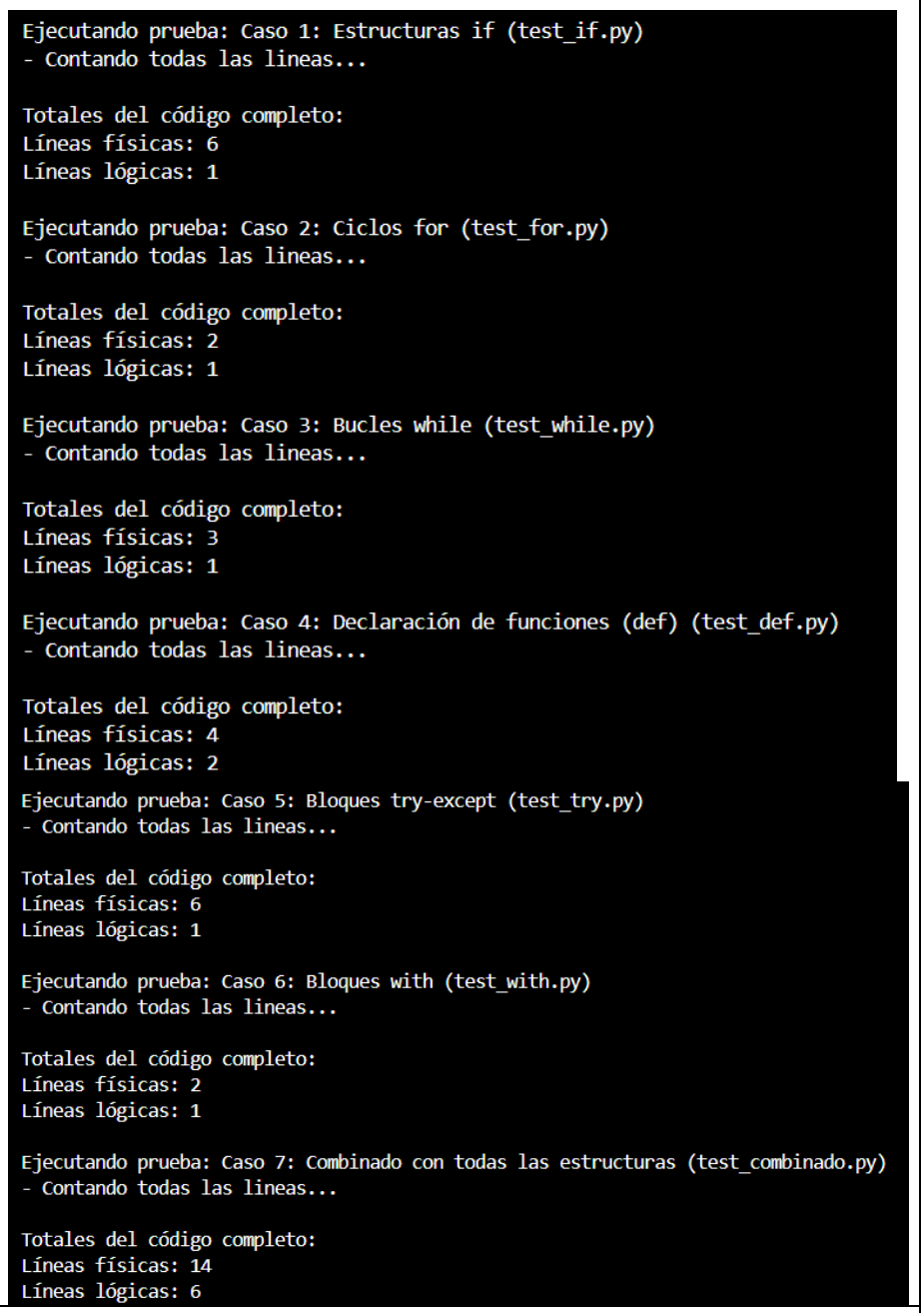
Tipo de prueba Prueba unitaria	
Nombre de la prueba/Numero objetivo de la prueba	P2 / Prueba_B Validar que el código sigue correctamente el estándar de conteo de líneas lógicas.
Descripción de la prueba	Resultados esperados por escenario I. if: Contar las líneas lógicas asociadas a la palabra clave if, reconociendo las condiciones evaluadas como una sola línea lógica. II. for: Identificar ciclos for, donde el encabezado del bucle se cuenta como una línea lógica, sin incluir las operaciones dentro del bloque. III. while: Contar la línea lógica correspondiente al encabezado de un bucle while, ignorando el contenido del bloque repetitivo. IV. def: Reconocer la línea lógica asociada a la declaración de funciones mediante la palabra clave def. Cada función es una línea lógica. V. try: Evaluar las líneas lógicas correspondientes al bloque try que gestiona excepciones, considerando el encabezado try y no los bloques except. VI. with: Contar la línea lógica asociada a bloques with, que gestionan el contexto de operaciones con recursos. VII. Combinado: Evaluar un archivo que contiene todas las estructuras anteriores, verificando que cada encabezado if, for, while, def, class, try, y with sea identificado correctamente como una línea lógica.

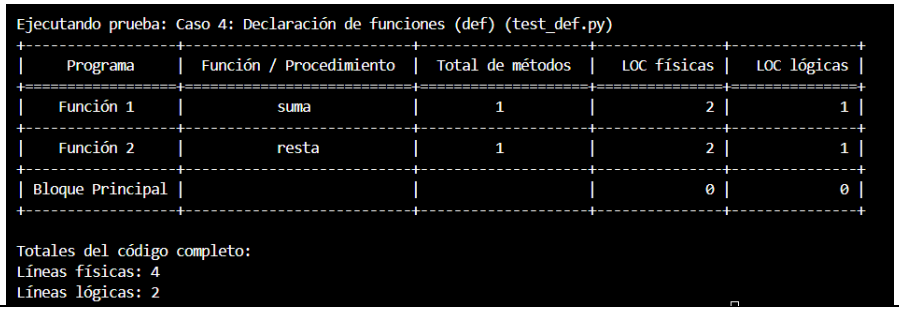
Condiciones de la prueba	Un script test5.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una.
Resultados esperados	I. 1 líneas lógicas II. 1 líneas lógicas III. 1 líneas lógicas IV. 2 líneas lógicas V. 1 líneas lógicas VI. 1 líneas lógicas VII. 4 líneas lógicas
Resultados actuales	
Comentarios	El código funciona correctamente en el conteo de líneas físicas.
Estado	1

Pruebas de integración:

Tipo de prueba Prueba integración	
Nombre de la prueba/Numero objetivo de la prueba	P3 / Prueba_ABC Validar que el código sigue correctamente el estándar de conteo de líneas físicas y lógicas, además que con el bloque C se pretende contar las de todo el programa.
Descripción de la prueba	Resultados esperados por escenario I. if: Contar las líneas lógicas asociadas a la palabra clave if, reconociendo las condiciones evaluadas como una sola línea lógica.

	II. for: Identificar ciclos for, donde el encabezado del bucle se cuenta como una línea lógica, sin incluir las operaciones dentro del bloque. III. while: Contar la línea lógica correspondiente al encabezado de un bucle while, ignorando el contenido del bloque repetitivo. IV. def: Reconocer la línea lógica asociada a la declaración de funciones mediante la palabra clave def. Cada función es una línea lógica. V. try: Evaluar las líneas lógicas correspondientes al bloque try que gestiona excepciones, considerando el encabezado try y no los bloques except. VI. with: Contar la línea lógica asociada a bloques with, que gestionan el contexto de operaciones con recursos. VII. Combinado: Evaluar un archivo que contiene todas las estructuras anteriores, verificando que cada encabezado if, for, while, def, class, try, y with sea identificado correctamente como una línea lógica.
Condiciones de la prueba	Un script test6.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una.
Resultados esperados	1. 6 líneas físicas, 1 líneas lógicas 2. 2 líneas físicas, 1 líneas lógicas 3. 3 líneas físicas, 1 líneas lógicas 4. 4 líneas físicas, 2 líneas lógicas 5. 6 líneas físicas, 1 líneas lógicas 6. 2 líneas físicas, 1 líneas lógicas 7. 14 líneas físicas, 6 líneas lógicas

Resultados actuales	
Comentarios	El código funciona correctamente en el conteo de líneas físicas y lógicas.
Estado	1
Tipo de prueba Prueba integración	
Nombre de la prueba/Numero objetivo de la prueba	P3 / Prueba_completa Validar que el código sigue correctamente el estándar de conteo de líneas físicas y lógicas, dividido en bloques y función principal.

Descripción de la prueba	Escenario: El escenario es un código que cuenta con dos funciones las cuales se espera que el código las detecte como funciones independientes y haga el conteo normal.
Condiciones de la prueba	Un script test7.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una.
Resultados esperados	Función 1: suma, 1, 2, 1 Función 2: resta, 1, 2, 1 Bloque 3: vacío Total Físicas: 4 Total Lógicas: 2
Resultados actuales	
Comentarios	El código funciona correctamente en el conteo de líneas físicas y lógicas dividido en bloques.
Estado	1

Unidad #3

Estimación de tamaño

Se utilizará la métrica de Puntos Funcionales para el tamaño y complejidad del sistema.

Se tomará en consideración el conteo de los siguientes elementos funcionales:

- Entradas lógicas
- Salidas
- Consulta (Querys)
- Archivos Lógicos Internos
- Archivos Lógicos Externos

Asignación del grado de complejidad con base a la siguiente tabla:

Elemento Funcional	Factor de Ponderación		
	Simple	Promedio	Complejo
Entradas Externas	3	4	6
Salidas Externas	4	5	7
Consultas Externas	3	4	6
Archivos Lógicos Externos	7	10	15
Archivos Lógicos Internos	5	7	10

Ecuación para el conteo de Puntos Funcionales sin Ajuste:

$$UFC = \sum Cantidad_{elemento} \cdot Peso_{elemento}$$

La siguiente plantilla se utilizará para calcular el factor de complejidad técnica para los puntos funcionales sin ajustar:

Componentes del factor de complejidad técnica		
F_1	Fiabilidad de la copia de seguridad y recuperación	
F_2	Funciones distribuidas	
F_3	Configuración utilizada	
F_4	Facilidad operativa	
F_5	Complejidad de interfaz	
F_6	Reutilización	
F_7	Instalaciones múltiples	
F_8	Comunicaciones de datos	
F_9	Desempeño	
F_{10}	Entrada de datos en línea	
F_{11}	Actualización en línea	
F_{12}	Procesamiento complejo	
F_{13}	Facilidad de instalación	
F_{14}	Facilidad de cambio	
	Total	

Valores posibles para dar:

0-Irrelevante o sin influencia

1-Incidental

2-Moderado

3-Medio

4-Significativo

Ecuación para el factor de complejidad técnica:

$$\mathbf{TFC} = \mathbf{0.65} + \mathbf{0.01} \sum_{\mathbf{i=1}}^{\mathbf{14}} \mathbf{F_i}$$

Ecuación para el conteo de Puntos Funcionales Ajustado

$$FP = UFC \cdot TFC$$

Por cada punto funcional encontrado se considerará:

$$1pf\ tarda\ aprox\ 10\ hrs$$

Descripción del sistema2:

Escribir un programa para contar las líneas totales de un programa, las líneas de cada clase y el total de métodos que contenga, si usas OO, o las líneas de código de cada función/procedimiento si utilizas programación estructurada. Reutilizar el programa 1.

Programa	Clase	Total de métodos	LOC físicas por clase	Total de LOC físicas del programa
123	ABC	3	86	
	DEF	4	92	
				178
456	…	…	…	

Programa	Función / Procedimiento	Total de métodos	LOC físicas p	Total de LOC físicas del programa
123	ABC	1	86	
	DEF	1	92	
				178
456	…	…	…	

Parte 1: Elementos identificados:

1- Entrada del archivo Python a evaluar en el conteo.
2- Salida de tabla del total de líneas físicas, lógicas y si cumple con el estándar de codificación de cada módulo y el total de líneas al final.

Parte 2: Conteo de los puntos de Función sin ajuste:

Elemento	Cantidad	Peso	Total
Entradas Externas	1	4	4
Salidas Externas	3	5	15
Consultas Externas	0	0	0
Archivos Lógicos Externos	0	0	0

Archivos Lógicos Internos	0	0	0
Total			19

$$UFC = (1 * 4) + (3 * 5) = 19$$

$$UFC = 19$$

Componentes del factor de complejidad técnica		
F_1	Fiabilidad de la copia de seguridad y recuperación	0
F_2	Funciones distribuidas	3
F_3	Configuración utilizada	0
F_4	Facilidad operativa	3
F_5	Complejidad de interfaz	0
F_6	Reutilización	3
F_7	Instalaciones múltiples	0
F_8	Comunicaciones de datos	0
F_9	Desempeño	0
F_{10}	Entrada de datos en línea	0
F_{11}	Actualización en línea	0
F_{12}	Procesamiento complejo	0
F_{13}	Facilidad de instalación	1
F_{14}	Facilidad de cambio	3
	Total	13

$$TFC = 0.65 + 0.01(13) = 0.78$$

$$FP = 19 \cdot 0.78 = 14.82 \approx 15$$

Tiempo esperado que puede tomar realizar el sistema:

$$Tiempo_{hrs} = 15 \cdot 10\ hrs = 150\ hrs\ \approx 150\ hrs$$

$$Tiempo_{dias} = \frac{150}{24} = 6.25\ días\ \approx 6\ días$$

Considerando que dos personas trabajaran en el desarrollo del programa podemos deducir el siguiente valor:

$$Tiempo_{dias} = \frac{6\ dias}{2\ desarrolladores}$$

3 días para cada desarrollador

Unidad #4

Mantenimiento, documentación de las decisiones y modificaciones

Introducción

Este apartado tiene como objetivo detallar las decisiones tomadas respecto a los cambios y modificaciones realizados en el código de la aplicación de conteo de líneas lógicas y físicas de cada módulo de un archivo Python. El propósito de estas modificaciones es mejorar la funcionalidad, optimizar el código y asegurar que el software cumpla con los requisitos actualizados de los establecidos en el proyecto 1 al 2.

Resumen de Cambios

1. Mejora de la Funcionalidad de Cálculo de Líneas

Se ha modificado la lógica para contar las líneas físicas y lógicas del código Python en el archivo seleccionado por el usuario. La metodología ahora permite identificar de manera más precisa las líneas físicas (sin contar las vacías ni los comentarios) y las líneas lógicas (que contienen estructuras de control como 'if', 'for', 'while', entre otras). Este cambio mejora la precisión en el cálculo y facilita la evaluación del código y su evolución.

2. Verificación de Estándares de Codificación

Se ha mejorado la lógica de verificación de estándares de codificación para garantizar que el archivo Python cumpla con las mejores prácticas de programación. Los cambios incluyen la validación de longitud de línea, espacios antes y después de las comas, indentación, y el uso de comentarios. Además, se han añadido checks para asegurarse de que los identificadores sigan las convenciones de estilo y que las funciones estén correctamente nombradas.

3. Optimización de la Identificación de Funciones y Bloques Principales

Se ha implementado una nueva estructura para identificar las funciones y el bloque principal del código. Esto permite una visualización más clara de la estructura del código en la salida del programa, separando correctamente las funciones del bloque principal y proporcionando información detallada de cada uno, como el número de líneas físicas y lógicas.

4. Mejora en el Uso de la Interfaz Gráfica (Tkinter)

La interfaz gráfica para la selección de archivos ha sido simplificada. Ahora, el programa permite al usuario seleccionar un archivo Python a través de un cuadro de diálogo de Tkinter, mejorando la experiencia del usuario.

Motivos de las Modificaciones

1. Claridad y Legibilidad del Código

Las modificaciones realizadas al código tienen como objetivo mejorar la claridad y legibilidad de este mismo. Esto incluye la mejora en la estructura del código para facilitar su mantenimiento y comprensión a largo plazo. La inclusión de comentarios claros y la optimización de la lógica permiten un código más fácil de leer y entender.

2. Cumplimiento de Estándares de Programación

El cumplimiento de los estándares de codificación es crucial para asegurar que el código sea consistente, seguro y fácil de mantener. Se han implementado modificaciones para garantizar que se sigan buenas prácticas como el uso adecuado de indentación, la correcta documentación en los comentarios y el cumplimiento de convenciones de nombres.

3. Mejora de la Eficiencia y Exactitud en los Cálculos

El cálculo de las líneas físicas y lógicas en el archivo Python es una tarea clave en el análisis de código. Las modificaciones mejoran la precisión de estos cálculos, proporcionando resultados más exactos y útiles para el análisis del código.

Conclusión

Las modificaciones realizadas han mejorado tanto la funcionalidad como la eficiencia del programa. La estructura del código es ahora más clara, y el análisis de líneas físicas y lógicas es más preciso. Además, las verificaciones de los estándares de codificación aseguran que el código sea consistente con las mejores prácticas de desarrollo. En general, los cambios implementados aseguran que el programa sea más robusto y fácil de usar.

Unidad #5

Aseguramiento de la calidad

Inspecciones

Inspección 1

- **Fecha de la inspección:**
- **Módulo inspeccionado:** Verificación de estándares en archivo (función Verificar_estandares_archivo).
- **Tipo de inspección:** Revisión de código.
- **Objetivo de la inspección:** Identificar defectos en la implementación de la verificación de estándares del código fuente.
- **Participantes y roles:**
    - **Moderador:** Rogerio Emmanuel Canto Romero.
    - **Inspector:** Eduardo Alberto González Ortega.
    - **Autor:** Jafet Andree Mena Solís.
    - **Secretario:** Santiago Efraín Itzincab Poot.
    - **Lector:** Eduardo Alberto González Ortega.
- **Resumen de la inspección:**
    - **Defectos encontrados:** 2
        1. **Descripción:** No se valida correctamente si un comentario inicia con una letra mayúscula cuando contiene caracteres especiales al inicio.
           Impacto: Puede permitir comentarios mal formateados.
           Acción propuesta: Mejorar la lógica para manejar caracteres

especiales antes de validar la mayúscula.
Responsable: Jafet Andree Mena Solís.

2. **Descripción:** Los comentarios en bloque no se manejan adecuadamente, ya que algunos podrían ser necesarios para la documentación interna.
Impacto: Puede eliminar información útil.
Acción propuesta: Añadir una opción para permitir comentarios en bloque en áreas específicas del código.
Responsable: Santiago Efraín Itzincab Poot.

- Plazo estimado para las correcciones: (1 día).
- Revisión de seguimiento:

Seguimiento:

El código ha sido corregido y evaluado.

Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.

Inspección 2

- **Fecha de la inspección:**
- **Módulo inspeccionado:** División de bloques y funciones (función Leer_codigo).
- **Tipo de inspección:** Revisión de código.
- **Objetivo de la inspección:** Identificar defectos en la lógica para dividir el código en funciones y bloques principales.
- **Participantes y roles:**
    - **Moderador:** Santiago Efraín Itzincab Poot.
    - **Inspector:** Jafet Andree Mena Solís.
    - **Autor:** Rogerio Emmanuel Canto Romero.

    - **Secretario:** Eduardo Alberto González Ortega.
    - **Lector:** Jafet Andree Mena Solís.
- **Resumen de la inspección:**
    - **Defectos encontrados:** 1
        - **Descripción:** Funciones definidas dentro de otras no se detectan correctamente como funciones independientes.
          Impacto: Puede causar un análisis incompleto.
          Acción propuesta: Implementar una verificación para manejar funciones anidadas.
          Responsable: Rogerio Emmanuel Canto Romero.
- Plazo estimado para las correcciones: (1 día).
- Revisión de seguimiento:

Seguimiento:

El código ha sido corregido y evaluado.

Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.

Inspección 3

- **Fecha de la inspección:**
- **Módulo inspeccionado:** Conteo de líneas físicas y lógicas (funciones Contar_lineas_fisicas y Contar_lineas_logicas).
- **Tipo de inspección:** Pruebas de código.
- **Objetivo de la inspección:** Validar el conteo correcto de líneas físicas y lógicas en diferentes escenarios.
- **Participantes y roles:**
    - **Moderador:** Eduardo Alberto González Ortega.

    - **Inspector:** Santiago Efraín Itzincab Poot, Rogerio Emmanuel Canto Romero.
    - **Autor:** Jafet Andree Mena Solís.
    - **Secretario:** Santiago Efraín Itzincab Poot.
    - **Lector:** Rogerio Emmanuel Canto Romero.
- **Resumen de la inspección:**
    - **Defectos encontrados:** 1
        - **Descripción:** Algunas estructuras de control no detectadas al usar diferentes formatos de escritura (if(condition):).
          Impacto: Puede generar un conteo incorrecto de líneas lógicas.
          Acción propuesta: Añadir soporte para formatos comunes en estructuras de control.
          Responsable: Jafet Andree Mena Solís.
- Plazo estimado para las correcciones: (1 día).
- Revisión de seguimiento:

Seguimiento:

El código ha sido corregido y evaluado.

Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.

Inspección 4

- **Fecha de la inspección:**
- **Módulo inspeccionado:** Impresión de bloques y conteo total (funciones Imprimir_bloques y Contar_lineas_totales).
- **Tipo de inspección:** Revisión de código.

- **Objetivo de la inspección:** Verificar que los resultados sean correctos y cumplan con los estándares de presentación.
- **Participantes y roles:**
    - **Moderador:** Jafet Andree Mena Solís.
    - **Inspector:** Eduardo Alberto González Ortega, Santiago Efraín Itzincab Poot.
    - **Autor:** Rogerio Emmanuel Canto Romero.
    - **Secretario:** Eduardo Alberto González Ortega.
    - **Lector:** Santiago Efraín Itzincab Poot.
- **Resumen de la inspección:**
    - **Defectos encontrados:** 2
        1. **Descripción:** Los nombres de las funciones impresas no conservan el formato original del código fuente.
           Impacto: Puede confundir a los desarrolladores al rastrear funciones específicas.
           Acción propuesta: Mejorar la lógica para mantener el formato original de los nombres de funciones.
           Responsable: Rogerio Emmanuel Canto Romero.
        2. **Descripción:** No se incluye un total acumulado de líneas lógicas y físicas por bloque.
           Impacto: Falta claridad en el resumen total del programa.
           Acción propuesta: Añadir un resumen acumulativo al final de la impresión.
           Responsable: Jafet Andree Mena Solís.
- Plazo estimado para las correcciones: (1 día).
- Revisión de seguimiento:

Seguimiento:

El código ha sido corregido y evaluado.

Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.