



UNIVERSIDAD AUTÓNOMA DE YUCATÁN

FACULTAD DE MATEMATICAS

SEMESTRE: AGOSTO-DICIEMBRE

Desarrollo Y Mantenimiento de Software

Proyecto Final: parte 1

M. en C. Carlos Benito Mojica Ruiz.

Alumnos:

Jimena Nohemi Cruz Arreola

Luis Manuel Palma Pinto

Juan Pablo Rodríguez Falcon

Angel Mariel Osalde Salazar

Gabriel Precenda Valle

Tabla de contenido

Unidad 1	2
Estándar de Conteo	2
Estándar de Codificación	6
Manual de Usuario	12
Pasos para ejecutar el programa	12
Manejo de excepciones	13
Unidad 2	13
Casos de Prueba	13
Pruebas Unitarias	14
Pruebas de Integración	20
Unidad 3	23
Estimación de Tamaño del Software	23
Unidad 5: Aseguramiento de la calidad	27
Metodología del aseguramiento de la calidad	27
Enfoque	27
Roles y responsabilidades	28
Actividades del aseguramiento de la calidad	29
Calidad administrativa	29
Calidad técnica	32

Unidad 1

Estándar de Conteo

El siguiente estándar de conteo tiene el propósito de servir como guía clara para identificar y clasificar las líneas como lógicas o físicas. Solo se incluyen aquellas sentencias que tienen un 'Sí' en la columna de 'Incluir'.

Estándar de conteo para códigos escritos en Python:

Tipo de conteo Físico / Lógico	Tipo	Lenguaje	
	Físico	Python	
Tipo de sentencia	Incluir	Comentarios	Ejemplo (Opcional)
Ejecutables	Sí		
No ejecutables:			
Declaraciones	Sí	Pueden ser declaraciones de funciones, variables, etc.	<pre>x = 10 # Declaración de variable y = 20</pre>
Declaraciones multilínea	Sí	Las declaraciones multilínea se tomarán en consideración como una línea física en toda su extensión. Es decir, si una declaración utiliza 4 líneas, las 4 serán consideradas como físicas.	<pre>numeros = [1, 2, 3, 4, 5, # Primera fila 6, 7, 8, 9, 10, # Segunda fila 11, 12, 13, 14, 15 # Tercera fila]</pre>
Comentarios		Se tomará en	
- En una sola línea	No	consideración que un comentario utiliza #	

- Multilínea	No	(simple) o “""" """” (multilínea); apenas se encuentre uno de estos indicadores se ignorará la línea del código.	<pre> """ Esta función suma dos números. Parámetros: ----- a : int o float Primer número para sumar. b : int o float Segundo número para sumar. Retorna: ----- int o float La suma de los dos números. Notas: ----- Este comentario multilínea dentro del docstring no es considerado código ejecutable, sí se debe contar como parte de las líneas de código lógicas. """ # Este es un comentario de una sola línea </pre>
Líneas en blanco o vacías	No	Se considerará como línea vacía o en blanco si no tiene ni un solo carácter, incluso si son espacios en blanco o tabulaciones.	6
Importaciones	Sí		<code>import pandas as pd</code>
Decoradores	Sí		<pre> ##### # Código de ejemplo en Python # ##### </pre>
Palabras reservadas	No	Las palabras reservadas como tal no se toman en consideración para agregar al conteo. Caso contrario que sucede en el conteo de líneas lógicas.	<pre> False None True and as assert async </pre>

Tipo de conteo Físico / Lógico	Tipo	Lenguaje	
	Lógico	Python	
Tipo de sentencia	Incluir	Comentarios	Ejemplo (Opcional)
Ejecutables			

if	Sí		<pre>x = 10 if x > 5: print("Es mayor que 5")</pre>
elif	No	No se considerarán estas declaraciones porque son como casos. Similar a las sentencias Switch en otros lenguajes.	<pre>x = 10 if x > 15: print("Mayor que 15") elif x > 5: print("Mayor que 5 pero no mayor que 15")</pre>
else	No		<pre>x = 3 if x > 5: print("Mayor que 5") else: print("No es mayor que 5")</pre>
for	Sí		<pre>for i in range(5): print(i)</pre>
while	Sí		<pre>x = 0 while x < 5: print(x) x += 1</pre>
def	Sí		<pre>def sumar(a, b): return a + b print(sumar(3, 4))</pre>
class	Sí		<pre>class Persona: def __init__(self, nombre): self.nombre = nombre p = Persona("Juan") print(p.nombre)</pre>
try	Sí		<pre>try: x = 10 / 0 except ZeroDivisionError: print("No se puede dividir entre 0")</pre>
except	No		<pre>try: x = 10 / 0 except ZeroDivisionError: print("No se puede dividir entre 0")</pre>
with	Sí		<pre>with open("archivo.txt", "w") as f: f.write("Hola mundo")</pre>
Listas de comprensión	Sí	Estas declaraciones solo serán consideradas como una única línea lógica	
No ejecutables			
Comentarios			

- En una sola línea	No		
- Multilínea	No		
- Líneas en blanco o vacías	No		
Importaciones	No		
Decoradores	No		
Palabras reservadas	No	A excepción de las mencionadas anteriormente no se toman en consideración otro tipo de palabras reservadas	

Ejemplos básicos de cómo se aplica el Conteo de Líneas

Los siguientes ejemplos utilizan ambas tablas definidas anteriormente para poder definir qué se considera línea física y línea lógica.

- Archivo con líneas vacías y comentarios:

```

□ if(a)
□     a = a + 1
□ else
□     a = a - 1

```

- Líneas físicas: 4
- Líneas lógicas: 1

- Archivo con comentarios:

- def suma (a, b):
- # Línea física
- return a + b
- # Línea física
- # Comentario de una línea

- Líneas físicas: 2
- Líneas lógicas: 1

Estándar de Codificación

Este estándar tiene como objetivo garantizar la calidad, mantenibilidad y uniformidad del código fuente en todos los proyectos realizados por el equipo, promoviendo prácticas de desarrollo consistentes y eficientes.

1. Convenciones de Nombres

A continuación, se especifica las convenciones que se deben seguir para nombrar las clases, métodos y variables:

- **Clases**
 - o Se utiliza la convención Upper Camel Case para nombrar las clases.

Ejemplo

Buen uso
<pre>class MiClase: #codigo de la clase Pass</pre>
Mal uso
<pre>class miclase: #codigo de la clase Pass</pre>



- **Métodos**

- o Se emplea la convención snake_case para nombrar los métodos dentro de las clases.

Ejemplo

Buen uso
<pre>class MiClase: def mi_primera_funcion(self): #codigo de la clase Pass</pre>
Mal uso
<pre>class miClase: def miPrimeraFuncion(self): #codigo de la clase Pass</pre>

- **Variables**

- o Para las variables, se aplican la convención snake_case, para toda aquella variable que requiera ser declarada con más de una palabra, solo si esto ayuda al correcto entendimiento de su propósito.

Buen uso
<pre># Variable con un nombre claro y descriptivo mi_variable = 1000</pre>

Mal uso
Variable que no sigue la convención MiVariable = 1000

- **Archivos y Carpetas:**

- o **Archivos:** Los archivos deben de seguir un formato snake_case con mayúscula en cada inicio de palabra.

Buen uso
Mi_Archivo_Principal.py
Mal uso
miArchivo_principal.py

- o **Carpetas:** El nombre de las carpetas que se creen deben de utilizar un formato Upper Camel Case.

Buen uso
MiCarpeta/
Mal uso
miCarpeta/

2. Comentarios en el Código

Las especificaciones que se deben seguir para documentar el código son las siguientes:

- **Comentarios en Línea**
 - o Los comentarios dentro del código se indicarán utilizando el símbolo # seguido de un texto explicativo, el cual, deberá ser breve y relevante.

Buen uso

```
def calcular_suma(a, b):
    # Esta función calcula el total de sumar a con b
    return a + b
```

Mal uso

```
def calcular_suma(a, b):
    # esta función está muy genial
    return a + b
```

- **Docstrings**

Documentar cada función al inicio con una breve descripción de su propósito, los parámetros que recibe y los valores que retorna.

- o Se utilizan docstrings para documentar las clases y los métodos, describiendo su propósito y/o funcionalidad de manera clara y concisa.
- o Los docstrings deben usarse al inicio de **clases**, **funciones** y **módulos**. Ejemplo para funciones:

Buen uso

```
def calcular_suma(a, b):
    """
    Esta funcion calcula la suma de dos números
    Parámetros:
    a(int): El primer numero
    b(int): El segundo numero
    Retorna:
    int: La suma de los 2 numeros
    """
    return a + b
```

Mal uso

```
def calcular_suma(a, b):
    """
    La función realiza la resta de dos números
```

```
Contexto:
Cuando hice esta función estaba lloviendo
El cielo estaba ...
“””
return a + b
```

3. Formato y Estructura

Indica cómo debe organizarse el código.

- **Indentación**

- o Usar 4 espacios para la indentación o 1 tabulación por nivel de indentación.
- o Las estructuras de control (por ejemplo: *if*, *for*, *while*, *etc*) y definiciones (*class*, *def*) deben estar correctamente alineadas para garantizar la legibilidad.

- **Líneas de código**

Limitar la longitud de una línea a 80 caracteres, excepto cuando sea completamente inevitable. Por ejemplo, cuando se presenten líneas con cadenas de texto largas o declaraciones complejas.

- **Separación entre secciones**

Dejar una línea en blanco entre:

- o Métodos dentro de una clase.
- o La definición de clases y otros bloques principales.
- o Comentarios de bloque y el código al que hacen referencia.

- **Bloques Lógicos y de Control**

En estructuras de control como *if* y *for*, emplear una línea por cada declaración, incluso cuando sea breve, para mayor claridad.

Ejemplo:

Buen uso
<pre>if not línea_sin_espacios: continue</pre>
Mal uso

```
if not línea_sin_espacios: continue
```

- **Importaciones**

Colocar las importaciones al inicio del archivo y en el siguiente orden:

- Librerías estándar.
- Librerías externas.
- Módulos locales.

- **4. Manejo de errores**

Utilizar el manejo de errores para ciertas circunstancias que puedan hacer que el programa falle.

- **FileNotFoundError:** Esta excepción ocurrirá si el usuario ha introducido mal el nombre del archivo que desea analizar o si no existe en la carpeta designada para que el analizador pueda acceder a ella.
- **IOError:** Esta excepción ocurrirá si ocurre un problema al leer el archivo, es decir, permisos insuficientes para acceder a la ubicación del archivo o si ocurre un error en el disco.
- **UnicodeDecodeError:** Esta excepción ocurrirá si el archivo a leer no está en un formato utf-8 es decir si no es un archivo válido para el sistema.

Manual de Usuario

Los códigos proporcionados son una herramienta ejecutable diseñada para calcular líneas lógicas y físicas en archivos de código Python.

- **Analiza** archivos de tipo Python para identificar líneas lógicas y físicas excluyendo comentarios y líneas en blanco.
- **Cuenta** las líneas de código físicas y lógicas según el estándar de conteo definido.
- **Gestiona** la interacción entre el usuario y los módulos de análisis mediante una interfaz proporcionada en una terminal.

Pasos para ejecutar el programa

1. Asegúrate de que tengas Python 3.8 o superior instalado en tu computadora
2. Colocar los archivos con extensión ‘.py’ que se quieran analizar en la carpeta ‘/analizador’.
3. Ejecutar el programa
 - Para ejecutar el programa simplemente se le debe de dar doble clic al archivo ejecutable llamado ‘analizador’. Esto nos abrirá una terminal que nos mostrará un texto de cómo se debe de utilizar el programa.
 - Debemos de ingresar el nombre del archivo que deseamos analizar (archivo Python que debe estar en la carpeta señalada anteriormente). El programa se ejecutará sobre el archivo seleccionado y nos mostrará la salida de LOC físicos y lógicos.
 - Después del despliegue del resultado del archivo se nos preguntara si deseamos ingresar otro archivo. En dado caso de no querer el programa finalizara.

Interpretación de los resultados de la aplicación *app.py*

Después de haber ejecutado el código se obtendrá como salida una tabla que contendrá:

- **Programa:** Nombre del archivo analizado
- **LOC (Líneas de Código) Lógicas:** Indica la cantidad de líneas de código ejecutables.

- **LOC (Líneas de Código) Físicas:** Total de líneas presentes en el archivo, excluyendo vacías y comentarios.

Ejemplo de Salida del programa:

Programa	LOC Lógicas	LOC Físicas
-----	-----	-----
archivo_ABC	25	40
archivo_XYZ	30	50

Manejo de excepciones

Al ejecutar el programa pueden darse ciertos escenarios en los que el programa tenga que finalizar debido a un incorrecto uso de este. A continuación, se mencionará cuando el programa se verá forzado a terminar su ejecución. Por cada tipo de excepción (error/fallo) se tendrá un mensaje dando la razón de por qué finalizo.

1. **Abrir un archivo que no existe:** por ejemplo, si el programa intenta leer un archivo que no está en el directorio especificado, se generará este error.
2. **Error de entrada / salida:** esto puede ocurrir por problemas al intentar leer o escribir un archivo.
3. **Problema al decodificar:** cuando el programa intenta decodificar una cadena de bytes en texto Unicode. Esto puede suceder normalmente cuando contiene caracteres que no son compatibles con la codificación especificada.

Unidad 2

Casos de Prueba

Los casos de prueba se diseñan para verificar que el código funcione como se espera en diferentes situaciones. Así como al usuario para la verificación del funcionamiento del

programa, así como la garantía de revisar cada procedimiento que se esté realizando por el equipo:

Instrucciones para el llenado del formato de Registro de Pruebas

Registro de Pruebas (Tipo de prueba)	
Id/Nombre	Identificador único de las pruebas
Objetivo	Mencionar brevemente la razón por la cual se está realizando la prueba
Descripción	Describir los datos (las entradas) y el procesamiento que va a tener el programa sobre esas entradas.
Condiciones (ambiente de la prueba)	Mencionar el ambiente el que se está ejecutando la prueba. Es decir, las herramientas utilizadas con las que se llevó a cabo la prueba.
Resultados esperados	Listar los resultados que la prueba debe producir si corre apropiadamente
Resultados Obtenidos	Listar los resultados que produjo la prueba. Las salidas obtenidas, así como los posibles defectos encontrados.
Comentarios	Escribir posibles comentarios que puedan ayudar al mejor entendimiento de la prueba.
Estado	Señalar el cómo se considera la prueba: <ul style="list-style-type: none"> • Exitosa • Fallida

Pruebas Unitarias

Objetivo: Validar que cada funcionalidad del analizador funcione correctamente.

Funcionalidad: Líneas físicas

Registro de Pruebas (Unitaria)

Id/Nombre	P1 / Prueba_A
Objetivo	Se quiere probar que el código siga correctamente el conteo de líneas físicas
Descripción	<p>El programa tiene preparado 5 test los cuales debe de cumplir para que la prueba sea considerada como exitosa.</p> <p>En cada escenario se tiene un código que cubra únicamente los puntos mencionados a continuación.</p> <ol style="list-style-type: none"> 1. Comentarios: Comentarios (simples y multilíneas) y líneas vacías. 2. Declaraciones: Declaraciones de variables, asignaciones e impresiones en la consola. 3. Importaciones: Llamada a librerías o archivos 4. Lógicas: Declaraciones lógicas 5. Completo: Todos los escenarios anteriores en uno solo
Condiciones (ambiente de la prueba)	<p>En esta prueba se hizo uso de una clase llamada <i>PruebaDeCodigo</i>, esto es con la finalidad de respetar el estándar de codificación establecido.</p> <p>Dentro de esta clase se tienen las funciones que apuntan a la carpeta <i>Test/</i>, carpeta la cual tiene en su interior los archivos Python que serán probados en sus respectivas funciones.</p> <p>En ese sentido, esta es una herramienta automatizada sin uso de librerías externas para realizar pruebas.</p>
Resultados esperados	<p>Resultados esperados por escenario</p> <ol style="list-style-type: none"> 1. Comentarios: Dado que este código únicamente contiene comentarios con un formato mencionado anteriormente se espera que el código no detecte ninguno de estos como líneas físicas. Salida = 0 líneas físicas 2. Declaraciones: Este código contiene declaraciones, asignaciones y uso de comentarios. Se espera que el analizador ignore los comentarios y únicamente cuente las declaraciones. Salida = 14 líneas físicas 3. Importaciones: El contenido de esta prueba son únicamente importaciones de librerías y de la llamada de un archivo. Se espera que el analizador sea capaz de detectarlos como líneas físicas.

	<p>Salida = 4 líneas físicas</p> <p>4. Lógicas: Para esta prueba se hizo uso de declaración de funciones, comentarios, docstrings, impresiones en pantalla y sentencias lógicas. Dado que se está haciendo el conteo de líneas físicas el analizador debería de poder ser capaz de contar sin tomar en consideración como se declaró el conteo de líneas lógicas.</p> <p>Salida = 12 líneas físicas</p> <p>5. Completo: Se combinan los códigos anteriores, se utiliza el contenido de todos en esta prueba. Esto con la finalidad de poder conocer que incluso en presencia de todos los posibles escenarios el analizador es capaz de poder realizar el conteo.</p> <p>Se espera que la salida sea igual a la suma de todas las salidas anteriores.</p> <p>Salida = 30 líneas físicas</p>
Resultados Obtenidos	<p>Salidas obtenidas de los escenarios</p> <ol style="list-style-type: none"> 1. Comentarios: Salida = 0 2. Declaraciones: Salida = 14 3. Importaciones: Salida = 4 4. Lógicas: Salida = 12 5. Completo: Salida = 30
Comentarios	Esta prueba es exclusiva para probar el funcionamiento del conteo de líneas físicas.
Estado	Exitosa

Objetivo: Validar que cada funcionalidad del analizador funcione correctamente.

Funcionalidad: Líneas lógicas

<p align="center">Registro de Pruebas (Unitaria)</p>
--

Id/Nombre	P2/Prueba_B
Objetivo	Se quiere probar que el código siga correctamente el estándar de conteo de líneas lógicas.
Descripción	<p>El programa tiene preparadas 7 pruebas individuales (una por cada estructura: if, for, while, def, class, try, with) y una prueba que combina todas estas estructuras en un solo archivo para evaluar el comportamiento global del analizador.</p> <p>Escenarios por superar:</p> <ul style="list-style-type: none"> • if: Evaluar el conteo de líneas lógicas en estructuras condicionales. • for: Evaluar ciclos for. • while: Evaluar ciclos while. • def: Evaluar declaraciones de funciones. • class: Evaluar declaraciones de clases. • try: Evaluar bloques de manejo de excepciones. • with: Evaluar bloques de contexto. • Combinado: Evaluar un archivo con todas las estructuras anteriores.
Condiciones (ambiente de la prueba)	Se utilizó una clase llamada PruebaDeCodigo, que incluye funciones automatizadas para ejecutar las pruebas. Estas funciones evalúan los archivos en la carpeta Test/, cada uno diseñado para representar un caso específico. No se utilizaron librerías externas para garantizar la independencia de las pruebas.
Resultados esperados	<p>Resultados esperados por escenario</p> <p>if: Contar las líneas lógicas asociadas a la palabra clave if, reconociendo las condiciones evaluadas como una sola línea lógica.</p> <p>for: Identificar ciclos for, donde el encabezado del bucle se cuenta como una línea lógica, sin incluir las operaciones dentro del bloque.</p> <p>while: Contar la línea lógica correspondiente al encabezado de un bucle while, ignorando el contenido del bloque repetitivo.</p> <p>def: Reconocer la línea lógica asociada a la declaración de funciones mediante la palabra clave def. Cada función es una línea lógica.</p>


	<p>class: Contar las líneas lógicas que contienen la declaración de clases iniciadas con la palabra clave class.</p> <p>try: Evaluar las líneas lógicas correspondientes al bloque try que gestiona excepciones, considerando el encabezado try y no los bloques except.</p> <p>with: Contar la línea lógica asociada a bloques with, que gestionan el contexto de operaciones con recursos.</p> <p>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.</p>
Resultados Obtenidos	<p>Salidas obtenidas de los escenarios</p> <ol style="list-style-type: none"> 1. if: Salida = 3. 2. for: Salida = 2. 3. while: Salida = 2. 4. def: Salida = 3. 5. class: Salida = 2. 6. try: Salida = 2. 7. with: Salida = 2. 8. Combinado: Salida = 7.
Comentarios	Las pruebas individuales y combinadas confirmaron que el analizador cumple con los estándares establecidos para el conteo de líneas lógicas.
Estado	Exitosa

Objetivo: validar que el modulo de interfaz funcione de manera correcta.

Funcionalidad: Manejo de archivos

Registro de Pruebas (Unitarias)	
Id/Nombre	P3/ interfaz de usuario.

Objetivo	Esta prueba simula la interacción que tendría el sistema con el usuario , con el objetivo de validar que se manejan de manera correcta las entradas de los archivos a analizar y si se requiere analizar nuevos archivos en la misma sesión.
Descripción	Estas pruebas están diseñadas para simular las entradas en la interfaz del usuario para que posteriormente le envié los archivos al analizador y este realice todo el proceso para devolver el informe a la interfaz quien la desplegara en la terminal Y terminara el proceso de análisis.
Condiciones (ambiente de la prueba)	<p>Configuración del ambiente:</p> <ul style="list-style-type: none"> Se utiliza la clase PruebaDeInterfaz que ejecuta pruebas sobre los archivos en la carpeta Test/. Esta clase automatiza las pruebas sin utilizar librerías externas, asegurando que se ejecuten de manera adecuada. <p>Los archivos fuente deben cumplir con las siguientes condiciones para que la prueba sea efectiva:</p> <ul style="list-style-type: none"> Indentación valida: Uso consistente de espacios y tabulaciones (ver estándar de codificación) Comentarios: comentarios en linea (#) y docstrings(“”). Pueden o no incluirlos Nombres de clases y métodos: Estructurados de manera correcta para ser identificados por la herramienta. Uso correcto de los bloques de control if,for,while,class. Como se establece en el estándar de codificación. Los archivos no deben tener errores de sintaxis.
Resultados esperados	Los resultados esperados son el correcto manejo de los archivos por parte de la interfaz, el despliegue de los informes proporcionados por los módulos y la pregunta si se desea continuar analizando.
Resultados Obtenidos	Los resultados obtenidos son satisfactorios, la prueba simulo la entrada de un usuario, que ingresa un archivo para analizar, este se lo manda al

	<p>módulo y el módulo devolvió el informe con la tabla de los resultados del análisis.</p> 
Comentarios	Con esto queda comprobado que la interfaz puede recibir y gestionar archivos y las interacciones con los usuarios
Estado	Exitosa

Pruebas de Integración

Las pruebas de integración tienen como objetivo verificar que los componentes del sistema (Analizador_De_Codigo.py y app.py) interactúan correctamente.

Objetivo: Validar que el programa funcione correctamente al analizar un archivo Python y devuelva el conteo correcto de líneas de código físicas y lógicas.

Funcionalidad: Ejecución completa del programa.

Registro de Pruebas (Integración)	
Id/Nombre	P5/Prueba_BC
Objetivo	<p>Validar el correcto funcionamiento e interacción del módulo Analizador_De_Codigo para garantizar que es capaz de procesar archivos de Python de forma precisa, contabilizando correctamente las líneas físicas y lógicas de un archivo.</p>
Descripción	<p>El programa tiene preparado para validar 4 escenarios distintos en los que se combinan diferentes situaciones, utiliza un archivo de código fuente como entrada, procesa sus líneas físicas y lógicas, y verifica que los resultados coincidan con los valores esperados previamente definidos. Los resultados de cada prueba indicaran si fue aprobada o no.</p>

Condiciones (ambiente de la prueba)	<p>Se usarán los archivos:</p> <ul style="list-style-type: none"> • Completo1.py • Completo2.py • Completo3.py • Completo4.py <p>Que contendrán diferentes situaciones de uso para el conteo de líneas físicas y lógicas como:</p> <ul style="list-style-type: none"> • Importaciones de módulos • Uso de ciclos, condicionales, casos, definición de funciones. • Comentarios de tipo bloque, en línea con el código y en línea.
Resultados esperados	<p>Con las pruebas realizadas se esperan obtener los siguientes resultados para asegurarnos que el sistema realiza el conteo correctamente de las líneas de código.</p> <ul style="list-style-type: none"> • Prueba 1: Completo1.py <ul style="list-style-type: none"> ○ Líneas físicas: 30 ○ Líneas lógicas: 4 • Prueba 2: Completo2.py <ul style="list-style-type: none"> ○ Líneas físicas: 20 ○ Líneas lógicas:8 • Prueba 3: Completo3.py <ul style="list-style-type: none"> ○ Líneas físicas: 46 ○ Líneas lógicas:14 • Prueba 4: Completo4.py <ul style="list-style-type: none"> ○ Líneas físicas: 64 ○ Líneas lógicas:22
Resultados Obtenidos	<p>Los resultados obtenidos en todas las pruebas coinciden con los valores esperados por lo que se demuestra que el sistema es capaz de contar líneas de código físicas y lógicas en los archivos de tipo Python.</p>

	<ul style="list-style-type: none"> • Prueba 1: Completo1.py <ul style="list-style-type: none"> ○ Líneas físicas: 30 ○ Líneas lógicas: 4 • Prueba 2: Completo2.py <ul style="list-style-type: none"> ○ Líneas físicas: 20 ○ Líneas lógicas:8 • Prueba 3: Completo3.py <ul style="list-style-type: none"> ○ Líneas físicas: 46 ○ Líneas lógicas:14 • Prueba 4: Completo4.py <ul style="list-style-type: none"> ○ Líneas físicas: 64 <p>Líneas lógicas:22</p>
Comentarios	Todas las pruebas fueron exitosas, lo que demuestra que el sistema cumple con los criterios esperados en cada caso de prueba. Este resultado confirma que el sistema es capaz de identificar y contar correctamente las líneas físicas y lógicas en diferentes condiciones.
Estado	Exitosa

Unidad 3

Estimación de Tamaño del Software

Se utilizará como métrica Puntos Funcionales para obtener el tamaño y complejidad del sistema descrito.

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

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

Para la asignación del grado de complejidad se usará como base 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

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

Se utilizará la siguiente plantilla para calcular el factor de complejidad técnica para los puntos funcionales sin ajustar

- 0: Irrelevante o sin influencia
- 1: Incidental

- 2: Moderado
- 3: Medio
- 4: Significativo
- 5: Esencia

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		

Ecuación para el factor de complejidad técnica

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

Ecuación para el conteo de Puntos Funcionales Ajustado

$$FP = UFC \cdot TFC$$

Por cada punto funcional encontrado se considerará:

1 punto funcional tarda aprox 11.35 hrs

Descripción del sistema solicitado

Escribir un programa para contar las líneas lógicas y líneas físicas en un programa, omitiendo comentarios y líneas en blanco. Utilizar y proveer el estándar de conteo y codificación utilizado.

Programa	LOC Lógicas	LOC Físicas
ABC	20	123
XYZ	34	345

Requisitos identificados

1. Entrada de uno o varios nombres de archivos Python al que se le realizara el conteo de líneas físicas y lógicas.
2. Entrada de querer continuar el programa o terminarlo.
3. Salida de tabla de número de líneas físicas y lógicas encontradas en el archivo analizado.
4. Archivo interno que contiene el estándar de codificación que permitirá conocer que se considera como línea física y lógica.

Conteo de los Puntos Funcionales sin ajuste

Elemento	Cantidad	Peso	Total
Entradas Externas	2	4	8
Salidas Externas	1	5	5

Consultas Externas	0	0	0
Archivos Lógicos Externos	0	0	0
Archivos Lógicos Internos	1	7	7
Total			20

$$UFC = (2 * 4) + (1 * 5) + (1 * 7) = 20$$

$$UFC = 20$$

Componentes del factor de complejidad técnica		
F_1	Fiabilidad de la copia de seguridad y recuperación	0
F_2	Funciones distribuidas	2
F_3	Configuración utilizada	0
F_4	Facilidad operativa	0
F_5	Complejidad de interfaz	0
F_6	Reutilización	0
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		6

$$TFC = 0.65 + 0.01(6) = 0.71$$

$$FP = 20 \cdot 0.71 = 14.2 \approx 15$$

Tiempo esperado que puede tomar realizar el sistema:

$$Tiempo_{hrs} = 15 \cdot 11.35 \text{ hrs} = 170.25 \text{ hrs} \approx 171 \text{ hrs}$$

$$Tiempo_{dias} = \frac{171}{24} = 7.125 \text{ días} \approx 8 \text{ días}$$

Considerando que tendremos al menos 2 personas que se encargaran del desarrollo del código este valor puede reducirse a el siguiente dato

$$Tiempo_{dias} \approx \frac{8 \text{ días}}{2 \text{ desarrolladores}} =$$

4 días por cada desarrollador

Unidad 5: Aseguramiento de la calidad

El objetivo principal del aseguramiento de la calidad es garantizar que el sistema:

- Cumpla con los requisitos funcionales y no funcionales definidos.
- Entregue resultados precisos y consistentes en diferentes entornos de uso.
- Sea confiable, mantenible y eficiente.

Además, este apartado tiene como propósito detallar las actividades, roles, procesos y herramientas que se utilizaron para asegurar tanto la calidad técnica del producto como la calidad administrativa del proyecto.

Metodología del aseguramiento de la calidad.

Esta sección describe cómo se implementará el aseguramiento de la calidad en el proyecto.

Enfoque

El enfoque de calidad en este proyecto se abordará en dos niveles: administrativo y técnico, con el objetivo de asegurar que tanto el proceso de desarrollo como el producto final cumplan con los estándares establecidos.

Calidad administrativa

En el nivel administrativo, se priorizará la planificación adecuada, la gestión eficiente de recursos y el seguimiento constante del progreso del proyecto. Esto incluirá la definición clara de roles y responsabilidades, la organización de inspecciones periódicas del proyecto, y la supervisión de los plazos y recursos. Las revisiones formales se llevarán a cabo para evaluar el cumplimiento de los estándares, asegurando que el proyecto avance de acuerdo con lo planificado.

Calidad técnica

A nivel técnico, se implementarán dos principales estrategias para garantizar la calidad del software:

- **Inspecciones:** Las inspecciones formales del código serán realizadas de manera regular para identificar defectos en el diseño, la lógica del software y la calidad del código fuente. Estas inspecciones ayudarán a detectar problemas tempranos en el ciclo de desarrollo, permitiendo que se corrijan.
- **Pruebas:** El sistema será sometido a una serie de pruebas técnicas, que incluyen:
 - **Pruebas unitarias** para validar el correcto funcionamiento de cada componente individual del sistema.
 - **Pruebas de integración** para asegurar que las distintas partes del sistema trabajen juntas de manera coherente.
 - **Pruebas de aceptación** para verificar que el sistema cumpla con los requisitos establecidos y entregue los resultados esperados.

Ambas estrategias, inspecciones y pruebas, estarán interrelacionadas y se realizarán a lo largo de todo el ciclo de vida del desarrollo, garantizando la detección y corrección temprana de defectos, así como la validación continua de que el producto cumple con los estándares de calidad.

Roles y responsabilidades

A continuación, se muestra los roles que tendrá cada persona del equipo y sus responsabilidades en el proceso del aseguramiento de la calidad.

- **Líder de calidad:** Jimena Nohemí Cruz Arreola
Sera el responsable de planificar y supervisar las actividades de QA.
- **Moderador:** Se encargará de coordinar las inspecciones de calidad.
- **Inspectores:** Serán los que revisaran el código fuente y reportaran defectos.
- **Autor del código fuente:** Persona que desarrolló el módulo que se está inspeccionando.
- **Escriba:** Documenta los hallazgos durante las inspecciones.

Actividades del aseguramiento de la calidad

En este apartado se mostrará las actividades/procesos que se llevaron a cabo durante todo el proceso de desarrollo del proyecto.

Calidad administrativa

Num . de revisi ón	Fecha de la revisión	Participan tes de la revisión	Tema revisado	Problema detectado	Impacto potencial	Acción propuesta
1	15/11/2024	Todo el equipo	Estándar de conteo	Falta de documentación formal sobre las definiciones de líneas lógicas y físicas.	Una mala interpretación de lo que el sistema debe contar, lo que puede llevar a no cumplir con el objetivo	Actualizar la documentación con lo que se acordó que se contaría como líneas físicas y lógicas, poniendo ejemplos

					del sistema.	para que sea claro.
2	19/11/2024	Líder de Calidad: Jimena Cruz Arreola Líder del proyecto: Luis Palma Pinto	Cronograma de revisiones con el profesor.	Se detectó que el cronograma de revisiones con el profesor debe ser actualizado debido a falta de fechas para entregas.	Un cronograma desactualizado puede llevar a entregar mal el producto de manera apresurada.	Actualizar el cronograma moviendo la primera revisión con el profesor una semana después de lo que se tenía planeado.
3	22/11/2024	Moderador: Luis Palma Pinto Revisor: Gabriel Precenda Autor: Jimena Cruz Arreola	Estándar de codificación	Falta de actualización del estándar de codificación	No mantener el estándar de codificación actualizado puede ocasionar malas prácticas en el desarrollo del código fuente.	Actualizar el estándar de codificación de acuerdo con lo que se ha establecido hasta el momento.

4	23/11/2024	<p>Líder de Calidad: Jimena Cruz Arreola</p> <p>Moderador : Luis Palma Pinto</p> <p>Encargado de pruebas: Juan Rodríguez</p>	Documentación de pruebas	Falta de una correcta documentación de pruebas.	Llevar una mala documentación de pruebas puede ocasionar confusiones o un incorrecto entendimiento de las pruebas realizadas	Actualizar la documentación de pruebas usando un formato que facilite su comprensión y entendimiento.
5	24/11/2024	<p>Revisor: Luis Palma Pinto</p> <p>Autor: Jimena Cruz Arreola</p>	Documentación de las Inspecciones.	Falta de un correcto formato para documentar las inspecciones realizadas.	No documentar correctamente las inspecciones puede llevar a cometer los mismos errores, no llevar un control de lo que se debe	Documentar de manera clara y correcta las inspecciones, cuidando tener un formato bien definido que ayude a su

					corregir y llevar a que no se tenga una buena calidad del producto.	comprensión.
--	--	--	--	--	---	--------------

Calidad técnica

Inspección 1

- Fecha de la inspección:
- Artefacto inspeccionado: Modulo de conteo de líneas lógicas y físicas
- Tipo de inspección: Revisión de código
- Objetivo de la inspección: Identificar defectos en la lógica del algoritmo de conteo de líneas físicas y lógicas.

Participantes y roles:

- o Moderador: Jimena Cruz Arreola.
- o Autor: Ángel Osalde Salazar.
- o Inspectores: Luis Palma Pinto y Juan Rodríguez Falcon.
- o Escriba: Gabriel Precenda Valle.

Resumen de la Inspección

- o Número total de defectos encontrados: 2
- o Detalles de los Defectos
 1. Defecto #1:
 - Descripción: El algoritmo no considera a las estructuras de control como líneas lógicas.
 - Impacto: Puede generar conteos incorrectos.
 - Acción propuesta: Actualizar la lógica para contar correctamente, como se acordó, las estructuras de control como una sola línea lógica por estructura.
 - Responsable: Ángel Osalde Salazar.

2. Defecto #2:

- Descripción: El algoritmo considera los comentarios como líneas físicas.
- Impacto: Puede generar conteos incorrectos.
- Acción propuesta: Actualizar el algoritmo para que ya no cuente los comentarios como líneas físicas.
- Responsable: Ángel Osalde Salazar.

o Plazo estimado para las correcciones: (1 día).

o Revisión de seguimiento:

Resultados de Seguimiento

o Estado de los defectos: Resuelto.

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

Inspección 2

- Fecha de la inspección:
- Artefacto inspeccionado: Modulo de conteo de líneas lógicas y físicas
- Tipo de inspección: Revisión de código
- Objetivo de la inspección: Identificar defectos en la lógica del algoritmo de conteo de líneas físicas y lógicas.

Participantes y roles:

- o Moderador: Jimena Cruz Arreola.
- o Autor: Ángel Osalde Salazar.
- o Inspectores: Luis Palma Pinto y Juan Rodríguez Falcon.
- o Escriba: Gabriel Precenda Valle.

Resumen de la Inspección

- o Número total de defectos encontrados: 2
- o Detalles de los Defectos

1. Defecto #1:

- Descripción: El algoritmo no debería de contar los comentarios en bloque como líneas físicas
- Impacto: Puede generar conteos incorrectos.
- Acción propuesta: Agregar una o unas estructuras de control para que se ignoren los comentarios en bloque
- Responsable: Ángel Osalde Salazar.

2. Defecto #2:

- Descripción: El programa debería de poder contar código multilínea de acuerdo con lo establecido en el estándar de conteo.
- Impacto: Puede generar conteos incorrectos y una falta de alineación a los estándares establecidos.
- Acción propuesta: Actualizar el algoritmo para que sea capaz de contar código multilínea ajustándose al estándar establecido.
- Responsable: Ángel Osalde Salazar.

o Plazo estimado para las correcciones: (1 día).

o Revisión de seguimiento:

Resultados de Seguimiento

o Estado de los defectos: Resuelto.

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

Inspección 3

- Fecha de la inspección:
- Artefacto inspeccionado: Pruebas
- Tipo de inspección: Revisión de las pruebas realizadas.
- Objetivo de la inspección: Identificar defectos en la ejecución de las pruebas correspondientes del sistema.

Participantes y roles:

- o Moderador: Gabriel Precenda Valle.
- o Autor: Juan Rodríguez Falcon.
- o Inspectores: Luis Palma Pinto y Jimena Cruz Arreola.
- o Escriba: Ángel Osalde Salazar

Resumen de la Inspección

- o Número total de defectos encontrados: 2
- o Detalles de los Defectos
 1. Defecto #1:
 - Descripción: Las pruebas unitarias deberían de haberse realizado acorde a lo que una prueba unitaria dicta, es decir, por módulos.
 - Impacto: Puede generar una incorrecta documentación de las pruebas y que el producto no sea realmente correcto.
 - Acción propuesta: Rehacer las pruebas unitarias ajustándose a lo que estas indican.
 - Responsable: Juan Rodríguez Falcon.
 2. Defecto #2:
 - Descripción: Las pruebas de integración están incompletas.
 - Impacto: Puede generar que el producto no este realmente bien integrado.
 - Acción propuesta: Agregar las pruebas de integración necesarias para asegurar que no hay ninguna falla en la integración de los módulos.
 - Responsable: Juan Rodríguez Falcon.
- o Plazo estimado para las correcciones: (2 días).
- o Revisión de seguimiento:

Resultados de Seguimiento

- o Estado de los defectos: Resuelto.
- o Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.

Inspección 4

- Fecha de la inspección:
- Artefacto inspeccionado: Puntos Funcionales
- Tipo de inspección: Revisión de la sección de conteo de puntos funcionales.
- Objetivo de la inspección: Identificar defectos en el conteo de puntos funcionales del sistema.

Participantes y roles:

- o Moderador: Gabriel Precenda Valle.
- o Autor: Luis Palma Pinto.
- o Inspectores: Ángel Osalde Salazar y Jimena Cruz Arreola.
- o Escriba: Juan Rodríguez Falcon.

Resumen de la Inspección

- o Número total de defectos encontrados: 2
- o Detalles de los Defectos
 - 3. Defecto #1:
 - Descripción: Falta de consideración del estándar de codificación como un archivo lógico interno.
 - Impacto: Puede generar una incorrecta interpretación.
 - Acción propuesta: Agregar este archivo y hacer las modificaciones, a los cálculos, necesarias derivadas de esto.
 - Responsable: Luis Palma Pinto.
 - 4. Defecto #2:
 - Descripción: Valores muy bajos en el cálculo final.
 - Impacto: Puede generar valores erróneos.
 - Acción propuesta: Revisar si se le está asignando correctamente los valores, por ejemplo, que estén como simples y realmente sean promedio, o si hay errores en los cálculos que lleve a tener números bajos.
 - Responsable: Luis Palma Pinto.

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

Resultados de Seguimiento

- o Estado de los defectos: Resuelto.
- o Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.

Inspección 5

- Fecha de la inspección:
- Artefacto inspeccionado: Código fuente
- Tipo de inspección: Revisión del código fuente.
- Objetivo de la inspección: Identificar malas prácticas que no vayan acorde al estándar de codificación.

Participantes y roles:

- o Moderador: Luis Palma Pinto.
- o Autor: Ángel Osalde Salazar.
- o Inspectores: Gabriel Precenda Valle y Juan Rodríguez Falcon.
- o Escriba: Jimena Cruz Arreola.

Resumen de la Inspección

- o Número total de defectos encontrados: 2
- o Detalles de los Defectos
 1. Defecto #1:
 - Descripción: el nombre de una función y una clase no cumplían con lo establecido en el estándar de codificación.
 - Impacto: Puede generar problemas futuros para su mantenimiento.
 - Acción propuesta: Reescribir el nombre para que cumpla con el estándar.
 - Responsable: Ángel Osalde Salazar.

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

Resultados de Seguimiento

- o Estado de los defectos: Resuelto.
- o Notas adicionales: Se realizó una segunda revisión y el problema fue corregido con éxito.