Logotipo

Descripción generada automáticamente

**UNIVERSIDAD AUTÓNOMA DE YUCATÁN**

**FACULTAD DE MATEMATICAS**

**Semestre:**

Agosto-diciembre

**Materia:**

Desarrollo Y Mantenimiento de Software

**Proyecto Final:**

Parte 1

**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

Unidad #1

Estándar de conteo

El siguiente estándar de conteo tiene como propósito servir como guía precisa para identificar y clasificar todas las líneas lógicas o físicas de un programa, basado en un paradigma de programación estructurada en el lenguaje de programación de Python.

|  |  |  |
| --- | --- | --- |
| **Tipo de Conteo**  **Físico / Lógico** | **Tipo** | **Lenguaje** |
| Físico | Python |
| **Tipo de sentencia** | **Incluir** | **Comentarios** |
| **Ejecutables** | Sí |  |
| **No ejecutables:** |  |  |
| Declaraciones | Si |  |
| Directivas del compilador | Si |
| **Comentarios:** |  |  |
| -En una sola línea | No |  |
| -Varias líneas | No |
| -Líneas vacías | No |
| -Comentarios en código | No |
| Importaciones | Si |  |
| Decoradores | Si |  |
| Palabras reservadas | Si |  |

|  |  |  |
| --- | --- | --- |
| **Tipo de conteo**  **Físico / Lógico** | **Tipo** | **Lenguaje** |
| **Lógico** | Python |
| **Tipo de sentencia** | **Incluir** | **Comentarios** |
| **Ejecutables** |  |  |
| if | Sí |  |
| elif | No |  |
| else | No |  |
| for | Sí |  |
| while | Sí |  |
| def | Sí |  |
| class | No | No es programación orientada a objetos, por tanto se espera que no haya declaración class. |
| try | Sí |  |
| except | No |  |
| with | Sí |  |
| **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 |

**Ejemplo de Conteo de Líneas:**

* Archivo con líneas vacías y comentarios:

Texto

Descripción generada automáticamente

* Líneas físicas: 10
* Líneas lógicas: 3

Estándar de codificación

El estándar de codificación tiene como principal objetivo garantizar la calidad, mantenibilidad y uniformidad del código fuente en todos y cada uno de los proyectos realizados por el equipo de desarrollo, promoviendo de esta manera buenas prácticas y eficientes.

**Longitud de líneas:**

La longitud de todas las líneas de código deberá estar en el rango máximo de 80 caracteres; Se pueden hacer excepciones para líneas de código que necesiten unos caracteres más cuando sea completamente inevitable. Los espacios en blanco no son contabilizados como carácter así como los saltos en línea.

**Comentarios:**

Los comentarios deben ser frases completas y concisas explicando las decisiones que se tomaron durante el desarrollo del sistema.

Los comentarios de línea comienzan con un “#”, luego aplicar un espacio para empezar a escribir el comentario.

Siempre deben empezar con una letra en mayúscula, evitar empezar con el nombre de un identificador.

Los comentarios dentro de líneas con código esta permitido siempre y cuando sea un comentario corto y significativo, mayormente los comentarios de preferencia deben ir en una línea aparte.

Usar triple comillas para comentarios en bloques no está permitido; Usaremos # en cada línea del bloque de comentario, iniciando la primera palabrea con mayúscula y un espacio después de #.

***Espacios en blanco:***

Debemos utilizar espacios en blanco entre elementos que están separados por comas; por ejemplo, las propiedades de una función. 

**Saltos de línea e indentación**

Las funciones tendrán dos líneas vacías arriba de y debajo de las mismas.

Las estructuras de control tendrán una línea vacía antes de su declaración y después de que hayan sido terminadas.

Se usarán 4 espacios para hacer cada indentación; queda prohibido el uso de las tabulaciones.

**Importaciones*:***

Las Importaciones deben ser declaradas en líneas distintas usando el siguiente orden de importación: librería estándar, librerías de terceros, librerías propias o locales.

Se recomienda siempre usar importaciones absolutas y evitar el uso de las importaciones relativas.

**Identificadores:**

Cuando nombramos una variable el nombre debe ser corto y explicito; la variable debe estar en minúsculas. Cuando declaramos una variable de dos o más palabras, la primera empieza en minúscula, y las demás empiezan en minúsculas o mayúsculas y van separadas por un “\_” sin espacio (funcion\_ejemplo).

Los identificadores de las funciones y las clases siempre empiezan con mayúsculas a diferencia de las variables, al igual que las variables si son conformadas por una o más palabras se separan por un “\_”, en caso de ser dos palabras la segunda después de \_ puede ir en mayúscula o minúscula.

**Estructura de Control:**

solo puede haber una declaración de estructura de control por línea, es decir, no pueden haber anidadas más de una estructura de control por línea.

Manual de usuario

**Introducción:**

Este script permite analizar archivos Python para verificar si cumplen con ciertos estándares de codificación y contar las líneas físicas y lógicas del código.

**Requisitos Previos:**

* Python instalado en tu equipo (versión 3.6 o superior recomendada).
* Biblioteca tkinter incluida (suele venir con Python por defecto).

**Características Principales:**

* Verificar estándares de codificación:

Longitud de líneas (máximo 80 caracteres).

Comentarios correctamente formateados.

Indentación consistente (múltiplos de 4 espacios).

Identificadores en minúsculas.

Nombres funciones en mayúsculas al inicio.

* Contar líneas físicas (líneas que aparecen en el archivo).
* Contar líneas lógicas (aquellas que representan acciones de control o definiciones clave).

**Instrucciones de Uso**

1. Ejecutar el Script:

Guarda el código en un archivo .py (por ejemplo, proyecto1.py).

Opciones de ejecución:

* **Run Python File**: Se puede ejecutar con el botón de Run Python File si se está utilizando un IDE que soporte archivos Python.
* **Terminal:** Abrir una terminal en la ruta donde este guardado el archivo proyecto1.py e ingresar el comando *“python proyecto1.py”.*

2. Ejecución:

- Seleccionar un Archivo:

Se abrirá un cuadro de diálogo para seleccionar un archivo Python. Navega hasta la ubicación del archivo que deseas analizar y selecciónalo.

3. Resultados:

Si el archivo cumple con los estándares, verás un mensaje indicando que no se encontraron errores.

Si hay errores, se mostrará una descripción detallada de cada problema, indicando la línea y el tipo de error.

Adicionalmente, el programa mostrará:

* Número de líneas físicas.
* Número de líneas lógicas.

4. Errores y Manejo

Si seleccionas un archivo no válido o inexistente, el programa notificará el error.

Otros errores inesperados serán capturados y mostrados en pantalla.

**Descripción de Funciones**

1. Verificar\_estandares\_archivo(archivo).

Analiza cada línea del archivo para identificar:

* Longitud mayor a 80 caracteres.
* Comentarios incorrectamente formateados (sin un espacio después de #, en minúsculas o vacíos).
* Indentación no múltiplo de 4 espacios.
* Uso incorrecto de comas (sin espacio después o con espacio antes).
* Identificadores que no están en minúsculas.
* Funciones sin nombres en minúsculas.

Parámetro:

* archivo: Ruta del archivo a analizar.

Salida:

* Imprime los errores encontrados (si existen) o confirma que no hay problemas.

2. Contar\_lineas\_fisicas(file\_path).

Cuenta las líneas físicas en el archivo (incluyendo todas menos las vacías o comentarios).

Parámetro:

* file\_path: Ruta del archivo a analizar.

Salida:

* Número de líneas físicas.

3. Contar\_lineas\_logicas(file\_path).

Cuenta las líneas lógicas en el archivo, considerando palabras clave como if, for, while, entre otras.

Parámetro:

* file\_path: Ruta del archivo a analizar.

Salida:

* Número de líneas lógicas.

4. Interfaz con tkinter:

Usa la biblioteca tkinter para mostrar un cuadro de diálogo de selección de archivos, permitiendo al usuario seleccionar fácilmente el archivo a analizar.

**Posibles Errores**

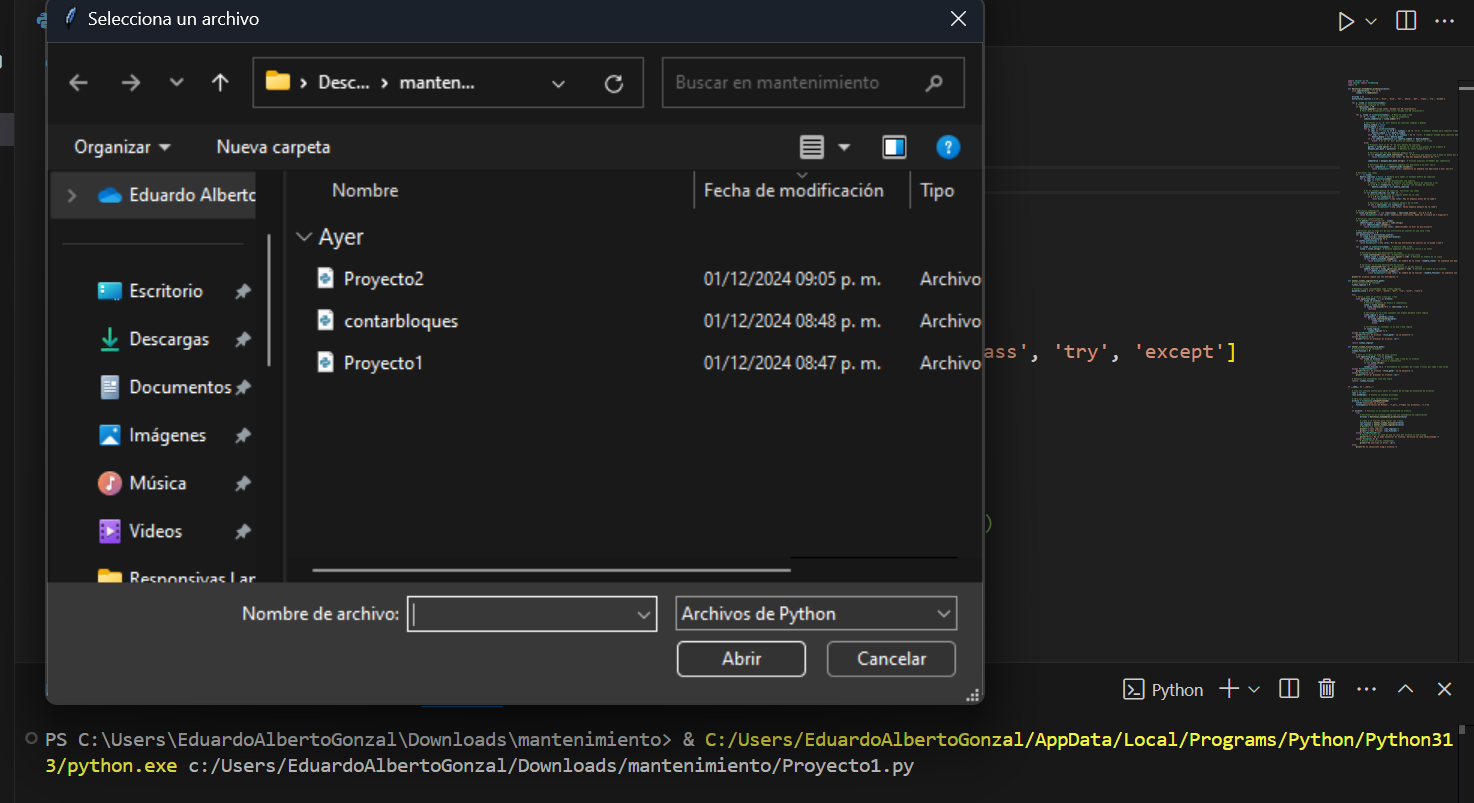
Archivo no encontrado: Ocurre si seleccionas un archivo inexistente.

Errores de formato: El programa reportará errores específicos relacionados con estándares de codificación.

Excepciones generales: Si ocurre un problema inesperado, se mostrará un mensaje con el detalle del error.

**Ejemplo de uso:**

Ejecutamos el programa y nos abrirá una ventana para elegir nuestro archivo a analizar



Una vez seleccionado el archivo el programa nos arrojará el resultado en la consola

Texto

Descripción generada automáticamente

Unidad #2

Casos de prueba

|  |  |  |  |
| --- | --- | --- | --- |
| Pruebas unitarias | | | |
| Archivo | Escenario | Líneas lógicas | Líneas físicas |
| test1.py | Archivo con comentarios validos | 0 | 1 |
| test2.py | Archivo con solo líneas físicas | 0 | 3 |
| test3.py | Identificador mal nombrado | 0 | 0 |
| test4.py | Archivo vacío | 0 | 0 |
| test5.py | Archivo con estructuras lógicas | 2 | 4 |
| Pruebas de integración | | | |
| Archivo | Escenario | Líneas lógicas | Líneas físicas |
| test6.py | Archivo con código | 1 | 3 |
| test7.py | Archivo con varias estructuras | 3 | 8 |
| test8.py | Archivo solo líneas físicas y comentarios | 0 | 5 |
| Pruebas de integración | | | |
| Archivo | Escenario | Líneas lógicas | Líneas físicas |
| test9.py | Código normal | 4 | 12 |
| test9.1.py | Código normal modificado | 4 | 12 |

**Pruebas Unitarias:**

**Objetivo: Validar que cada funcionalidad del analizador funcione correctamente.**

Escenario: Archivo con comentarios válidos.

Archivo: test1.py.

Resultado esperado:

* Líneas Físicas: 1.
* Líneas Lógicas: 0.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Escenario: Archivo con solo líneas físicas.

Archivo: test2.py.

Resultado esperado:

* Líneas Físicas: 3.
* Líneas Lógicas: 0.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Escenario: Archivo con identificador mal escrito.

Archivo: test3.py.

Resultado esperado:

* Líneas Físicas: 0.
* Líneas Lógicas: 0.



Escenario: Archivo vacío.

Archivo: test4.py.

Resultado esperado:

* Líneas Físicas: 0.
* Líneas Lógicas: 0.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Escenario: Archivo con estructuras lógicas.

Archivo: test5.py.

Resultado esperado:

* Líneas Físicas: 4.
* Líneas Lógicas: 2.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

**Pruebas de integración:**

**Propósito: Validar que el programa funcione correctamente al analizar archivos proporcionados por el usuario.**

Caso: analizar archivos test6.py, test7.py, test8.py

test6.py

Líneas Físicas: 3, Líneas Lógicas: 1.

test7.py

Líneas Físicas: 8, Líneas Lógicas: 3.

test8.py

Líneas Físicas: 5, Líneas Lógicas: 0.

Utilizando el código de TestAutomatico.py para múltiples pruebas, tenemos que:

**Texto

Descripción generada automáticamente**

**Pruebas de regresión:**

**Propósito:** asegurar que se mantengan los resultados consistentes tras cambios irrelevantes al código fuente a evaluar.

Caso:

Ejecutar test9.py y después volver a ejecutarlo pero con pequeñas modificaciones agregando líneas en blanco y comentarios extras.

Validar que los resultados son idénticos.

Primera evaluación:

Líneas físicas: 12

Líneas lógicas: 4

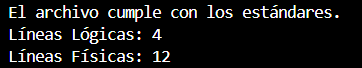
Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Segunda evaluación modificado:

Líneas físicas: 12

Líneas lógicas: 4



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:

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

Valores posibles para dar:

0-Irrelevante o sin influencia

1-Incidental

2-Moderado

3-Medio

4-Significativo

5-Esencial

|  |  |  |
| --- | --- | --- |
| Componentes del factor de complejidad técnica | | |
|  | Fiabilidad de la copia de seguridad y  recuperación |  |
|  | Funciones distribuidas |  |
|  | Configuración utilizada |  |
|  | Facilidad operativa |  |
|  | Complejidad de interfaz |  |
|  | Reutilización |  |
|  | Instalaciones múltiples |  |
|  | Comunicaciones de datos |  |
|  | Desempeño |  |
|  | Entrada de datos en línea |  |
|  | Actualización en línea |  |
|  | Procesamiento complejo |  |
|  | Facilidad de instalación |  |
|  | Facilidad de cambio |  |
| Total | |  |

Ecuación para el factor de complejidad técnica:

Ecuación para el conteo de Puntos Funcionales Ajustado

Por cada punto funcional encontrado se considerará:

**Descripción del sistema1:**  
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.

Parte 1: Elementos identificados:

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

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 | 1 | 0 | 0 |
| Total |  | | 19 |

|  |  |  |
| --- | --- | --- |
| Componentes del factor de complejidad técnica | | |
|  | Fiabilidad de la copia de seguridad y  recuperación | 0 |
|  | Funciones distribuidas | 3 |
|  | Configuración utilizada | 0 |
|  | Facilidad operativa | 0 |
|  | Complejidad de interfaz | 0 |
|  | Reutilización | 3 |
|  | Instalaciones múltiples | 0 |
|  | Comunicaciones de datos | 0 |
|  | Desempeño | 0 |
|  | Entrada de datos en línea | 0 |
|  | Actualización en línea | 0 |
|  | Procesamiento complejo | 0 |
|  | Facilidad de instalación | 1 |
|  | Facilidad de cambio | 3 |
| Total | | 10 |

Tiempo esperado que puede tomar realizar el sistema:

Unidad #5

Aseguramiento de la calidad

**Objetivo:**  
Definir de forma clara y documentada el alcance, los requisitos y los entregables del proyecto, utilizando como base estándares de calidad reconocidos, adaptados a las necesidades específicas del equipo de desarrollo. Los estándares seleccionados son:

1. **Moprosoft**: Orientado a la gestión de riesgos y la planificación de recursos.
2. **CMMI**: Enfocado en la gestión de requisitos para asegurar la alineación con las expectativas del cliente y las partes interesadas.

**1. Planificación del Proyecto:**

* Elaborar un plan detallado que incluya las fases del ciclo de vida del proyecto, entregables parciales y cronogramas específicos.
* Estimar el esfuerzo del proyecto mediante el uso del método de puntos funcionales.
* Identificar riesgos de manera continua, integrando análisis preventivos y correctivos.

**2. Desarrollo y Control de Calidad del sistema:**

* Implementar control de versiones utilizando Git, estableciendo ramas específicas para cada nueva característica.
* Incorporar pruebas de software como una parte central del desarrollo, abarcando pruebas unitarias, de integración y de regresión, con una cobertura mínima del 80%.
* Mantener revisiones manuales de código para detectar errores, asegurar calidad y verificar cumplimiento de estándares.
* Documentar el código mediante el uso de comentarios claros para facilitar su comprensión.
* Adherirse al estándar de codificación definido para Python (Estándar de codificación), realizando revisiones periódicas.

**3. Gestión mediante Git:**

* Gestionar el código fuente y sus versiones a través de Git, con etiquetado y registro de cambios detallado para cada versión.
* Implementar automatizaciones con herramientas como GitHub Actions para integrar pruebas, despliegues y asegurar calidad en cada etapa. Sustituido por la creación de un script .py para realizar varias pruebas a la vez
* Mantener un historial de configuraciones y versiones accesible y organizado.

**4. Gestión de Incidentes y Defectos:**

* Llevar un registro sistemático de errores e incidentes, utilizando herramientas o métodos personalizados según las necesidades del equipo.
* Realizar análisis de causas raíz para identificar patrones, resolver problemas recurrentes y mejorar los procesos internos.