Tecnológico de Monterrey

4.2 Ejercicio de programación 1

Alumno

A01794692 - Edwin David Hernández Alejandre

Pruebas de software y aseguramiento de calidad | Fecha: 4/02/2025

Objetivo

Explicar la importancia del estilo de la codificación de un sistema, así como reconocer los atributos de un estándar de codificación útil para identificar errores utilizando estándares de codificación reconocidos en la industria y sus implicaciones.

Desarrollo.

Problema 1: Compute Statistics

Salida inicial de Pylint:

La ejecución de `pylint` en el archivo `compute_statistics.py` generó varios comentarios que resaltaban áreas de mejora en el código. Los principales problemas identificados incluyeron:

1. Líneas demasiado largas: Varias líneas excedieron el límite de 100 caracteres.

2. Cadenas de documentación faltantes: Faltaban cadenas de documentación en el módulo y las funciones.

3. Nombre del módulo que no se ajusta a `snake_case`: El nombre del archivo `computeStatistics.py` no se ajustaba al estilo `snake_case`.

4. Demasiadas variables locales: La función `main` contenía demasiadas variables locales.

5. Captura de excepciones demasiado general: La `Exception` se estaba capturando de manera demasiado amplia.

6. Nueva línea faltante al final del archivo: Faltaba una nueva línea al final del archivo.

Actualizaciones realizadas.

Para mejorar el código y alinearlo con las pautas de PEP8, se implementaron las siguientes actualizaciones:

1. División de líneas largas: Las líneas que excedían el límite de 100 caracteres se dividían en varias líneas.

2. Agregar cadenas de documentación: Se incluyeron cadenas de documentación en el módulo y en todas las funciones para aclarar su propósito.

3. Cambio de nombre del módulo: El archivo `computeStatistics.py` se renombró como `compute_statistics.py` para cumplir con el estilo `snake_case`.

4. Refactorización de la función `main`: La lógica de procesamiento de archivos y cálculo de estadísticas se trasladó a una nueva función, `process_file_and_compute_statistics`, para reducir la cantidad de variables locales en `main`.

5. Captura de excepciones específicas: Las excepciones que se pueden capturar (`FileNotFoundError`, `OSError` y `Exception`) se especificaron explícitamente.

6. Nueva línea agregada al final del archivo: Se agregó una nueva línea al final del archivo.

Problema 2: Converter

Resultados Iniciales de Pylint

Al ejecutar pylint en el archivo convert_numbers.py, se generaron varios comentarios indicando áreas de mejora en el código. Los principales problemas identificados fueron:

1. **Líneas demasiado largas**: Varias líneas excedían el límite de 100 caracteres.
2. **Falta de docstrings**: Faltaban docstrings en el módulo.
3. **Captura de excepciones demasiado generales**: Se estaba capturando la excepción Exception de manera demasiado general.
4. **Falta de nueva línea al final del archivo**: Faltaba una nueva línea al final del archivo.

Actualizaciones Realizadas

Para mejorar el código y cumplir con las directrices de PEP8, se realizaron las siguientes actualizaciones:

1. **División de líneas largas**:
   - Las líneas que excedían el límite de 100 caracteres se dividieron en varias líneas para cumplir con el límite de 100 caracteres por línea.
2. **Añadido de docstrings**:
   - Se añadió un docstring al módulo para describir su propósito.
3. **Especificación de excepciones**:
   - Se especificaron las excepciones que se pueden capturar (FileNotFoundError, OSError y Exception), en lugar de capturar la excepción Exception de manera general.
4. **Añadido de nueva línea al final del archivo**:
   - Se añadió una nueva línea al final del archivo para cumplir con las directrices de PEP8.

Resultados finales de Pylint.

Después de las actualizaciones, se volvió a ejecutar `pylint` y se observó una mejora notable en la puntuación del código. La puntuación final fue **10/10**, lo que indica que el código ahora cumple en gran medida con las pautas PEP8.

Problema 3: Count Words

Resultados Iniciales de Pylint.

Al ejecutar pylint en el archivo count_words.py, se generaron varios comentarios indicando áreas de mejora en el código. Los principales problemas identificados fueron:

1. Líneas demasiado largas: Varias líneas excedían el límite de 100 caracteres.
2. Falta de nueva línea al final del archivo.
3. Captura de excepciones demasiado generales: Se estaba capturando la excepción Exception de manera demasiado general.
4. Nombre del módulo no conforme a snake_case: El nombre del archivo countWords.py no seguía el estilo snake_case.

Actualizaciones Realizadas.

Para mejorar el código y cumplir con las directrices de PEP8, se realizaron las siguientes actualizaciones:

1. **División de líneas largas:**
   - Las líneas que excedían el límite de 100 caracteres se dividieron en varias líneas para cumplir con el límite de 100 caracteres por línea.
2. **Añadido de nueva línea al final del archivo:**
   - Se añadió una nueva línea al final del archivo para cumplir con las directrices de PEP8.
3. **Especificación de excepciones:**
   - Se especificaron las excepciones que se pueden capturar (FileNotFoundError, OSError y Exception), en lugar de capturar la excepción Exception de manera general.
4. **Renombrado del archivo:**
   - El archivo countWords.py se renombró a count_words.py para seguir el estilo snake_case.

Resultados Finales de Pylint.

Después de realizar las actualizaciones, se volvió a ejecutar pylint y se observó una mejora significativa en el puntaje del código. El puntaje final fue de 10.00/10, lo que indica que el código ahora cumple completamente con las directrices de PEP8.

Conclusión General

Al reflexionar sobre lo aprendido, me doy cuenta de que seguir las pautas PEP8 no solo mejora la legibilidad del código, sino que también facilita su mantenimiento y la colaboración con otros desarrolladores. Incluir cadenas de documentación en módulos y funciones me ha ayudado a proporcionar descripciones más claras sobre su propósito y funcionalidad, lo que hace que el código sea más comprensible para mí y para otros. Además, la refactorización del código me ha permitido reducir la complejidad y la cantidad de variables locales, lo que se traduce en una mayor claridad y facilidad de mantenimiento. Aprendí que capturar excepciones específicas, en lugar de depender de excepciones generales, mejora significativamente la solidez y la claridad en el manejo de errores. Finalmente, el uso de herramientas de análisis estático como pylintha sido invaluable para identificar problemas y asegurar que cumpla con las mejores prácticas de codificación. En conjunto, estas lecciones no solo mejoraron mi puntuación en pylint, sino que también me ayudarán a escribir un código más limpio, claro y fácil de mantener.

Referencias

- *PEP 0 – Index of Python Enhancement Proposals (PEPS) | Peps.python.org*. (n.d.). Python Enhancement Proposals (PEPs). https://peps.python.org/
- The Python tutorial. (n.d.). Python Documentation.

  https://docs.python.org/3/tutorial/index.html