Computación Tolerante a Fallas
Profesor: Michel Emanuel Lopez Franco

Herramientas para el manejo de errores: Pylint

—

Reyes Becerra Uzziel

Objetivo:

Aplicar herramientas para el manejo de errores.

Desarrollo:

De todas las herramientas que investigué y encontré, la que me pareció más sencilla y práctica fueron los linters, pues ni siquiera ocupas ejecutar el código para revisarlo.

Como iba a utilizar python, decidí utilizar Pylint, un linter especializado para python, su instalación es tan sencilla como correr un comando en la terminal, y su utilización es igual, solo correr un comando, me resultó complicado utilizarlo porque no estoy acostumbrado a navegar por la terminal, y es necesario situarse en la misma carpeta donde esta el archivo que realizarás, el comando es: python -m pylint .\"tuarchivo,py", y al ejecutarlo en la terminal aparecen los resultados del análisis de pylint, comentarios, sugerencias e incluso una calificación sobré 10 acerca de la estructura de tu código lo cual es creativo e interactivo, realice un código de una cola para probar el pylint, mi código era el siguiente:

```
class Cola:
    def __init__(self):
        self.items = []
    def esta_vacia(self):
        return self.items == []
    def agregar(self, item):
        self.items.insert(0, item)
    def eliminar(self):
        if not self.esta_vacia():
            return self.items.pop()
        else:
            return "La cola está vacía"
    def tamano(self):
        return len(self.items)


    def mostrar(self):
        return self.items
cola = Cola()
cola.agregar(1)
cola.agregar(2)
```

```
cola.agregar(3)

print("Cola actual:", cola.mostrar())

print("Elemento eliminado:", cola.eliminar())

print("Cola actualizada:", cola.mostrar())

print("¿La cola está vacía?", cola.esta_vacia())

print("Tamaño de la cola:", cola.tamano())
```

Después de ejecutar el pylint me dio lo siguiente:

```
PS C:\Users\Mattfreink\Desktop\2024-A\Computación Tolerante a Fallas> python -m pylint .\Prueba.py

************* Module Prueba

Prueba.py:33:0: C0304: Final newline missing (missing-final-newline)

Prueba.py:1:0: C0114: Missing module docstring (missing-module-docstring)

Prueba.py:1:0: C0103: Module name "Prueba" doesn't conform to snake_case naming style (invalid-name)

Prueba.py:1:0: C0115: Missing class docstring (missing-class-docstring)

Prueba.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)

Prueba.py:6:15: C1803: "self.items == []" can be simplified to "not self.items", if it is strictly a
sequence, as an empty list is falsey (use-implicit-booleaness-not-comparison)

Prueba.py:8:4: C0116: Missing function or method docstring (missing-function-docstring)

Prueba.py:11:4: C0116: Missing function or method docstring (missing-function-docstring)

Prueba.py:12:8: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside
it (no-else-return)

Prueba.py:17:4: C0116: Missing function or method docstring (missing-function-docstring)

Prueba.py:20:4: C0116: Missing function or method docstring (missing-function-docstring)

-----------------------------------

Your code has been rated at 5.42/10
```

Una calificación sin duda bastante baja, pero ya teniendo los consejos del lint puedo mejorarla, para seguir probando agregue errores para ver qué decía el lin y qué calificación resultaba pero al agregar “`int su= 28`” me dio un error de sintaxis y no ejecutó el análisis, lo cual fue decepcionante, pensé que independientemente del error correría el análisis, pero supongo que al ser un error tan evidente no se espera.

Ya para terminar ejecute el análisis con otro código de una práctica de otra clase, la cual tuvo una calificación mucho más alta pero estaba llena de sugerencias a cambios en el código, si bien no es la herramienta perfecta, puede ayudar a optimizar el código y ver las implementaciones de respuestas a problemas desde otra perspectiva.