Sistemas de Gestion de Calidad

EPET N°12

Linters / Profilers

Ing. Eduardo Kunysz

Temas

- Indice
  - Linter
  - Profile

Concepto general

Definición

Los linters son programas que realizan **análisis estático** de código

Originalmente Lint era el nombre de una herramienta de programación del lenguaje C para detectar código sospechoso, confuso o incompatible. Servía para detectar errores de programación que escapan al análisis sintáctico que hace el compilador.

- Uso de variables antes de ser inicializadas o creadas.
- Condiciones que no varían bajo ninguna circunstancia (siempre verdaderas o falsas).
- Cálculos cuyos resultados probablemente caigan fuera del rango permitido por la variable.

Definición

Definición Zen

Los linters son maestros digitales, que en cada ejecución enseñan el camino a la iluminación para escribir código de alta calidad.

En ella, tu par es una máquina, fría y objetiva. Sin prejuicios ni celos. No se enoja porque el día anterior le diste una mala mirada o le terminaste el café de la oficina.

Concepto general

Pylint

Los linters como **Pylint** y **Flake8** son herramientas que analizan el código en Python para detectar errores, malas prácticas y mejorar la calidad

Pylint: Es un linter completo y exhaustivo

- Analiza la calidad del código, sugiriendo mejoras.
- Revisa posibles errores lógicos y de código (por ejemplo, variables no utilizadas, métodos innecesarios).
- Soporte para convenciones como PEP8, pero va más allá al verificar problemas de diseño y rendimiento del código.

Instalación

```
pip install pylint
```

Ejecuciń básica

```
pylint mi_script.py
```

Concepto general

Pylint

```
1 ************* Module mi_script
2 mi_script.py:1:0: C0114: Missing module docstring (missing-module-docstring)
3 mi_script.py:3:0: C0116: Missing function or method docstring (missing-function-docstring)
4 mi_script.py:3:8: C0103: Variable name "x" doesn't conform to snake_case naming style (invalid-name)
5
6 -----------------------------------------------------------------
7 Your code has been rated at 6.67/10 (previous run: 6.67/10, +0.00)
```

Categoría del problema: Errores (E), advertencias (W), convenciones de estilo (C), entre otros.

Código del problema: Un código específico para el tipo de problema (por ejemplo, C0114).

Descripción: Un mensaje que describe el problema encontrado.

Puntuación: Al final, Pylint da una puntuación al código sobre 10. Un puntaje de 10 significa que el código sigue las mejores prácticas.

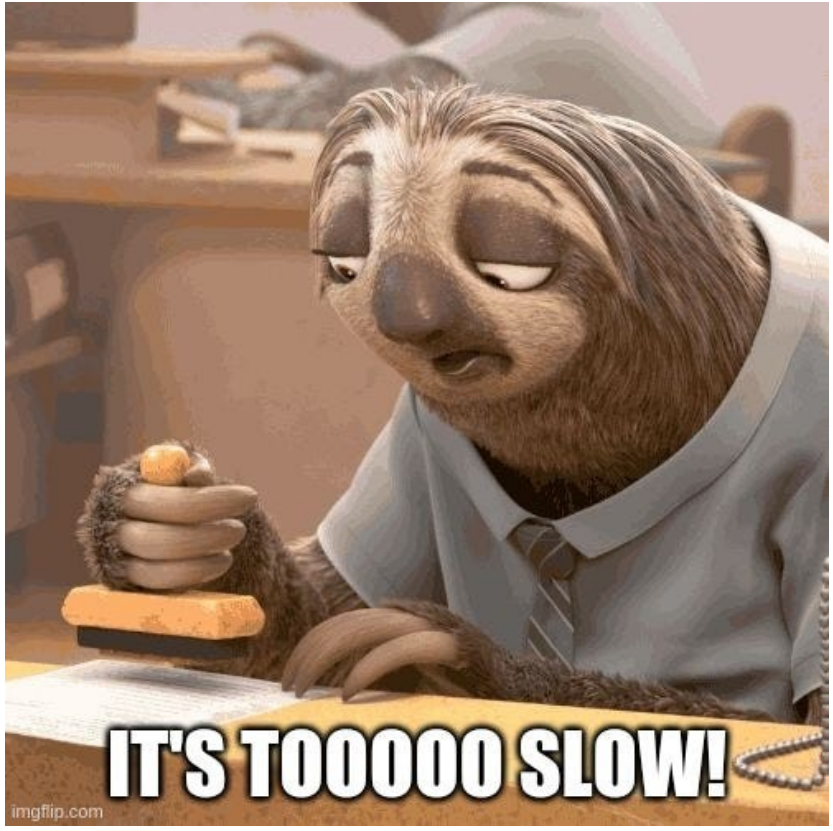
Profile

El módulo **`profile`** permite realizar profiling del código en Python.

Sirve para medir el rendimiento de las funciones y detectar cuellos de botella.

El profiling es el proceso de analizar qué partes del código consumen más tiempo de ejecución.

Se recomienda usarlo para identificar qué funciones optimizar cuando se busca mejorar la eficiencia del código.



Cómo Usar Profile en un Script

Uso Básico de profile

```python
import profile

def ejemplo():
    for i in range(1000000):
        pass

profile.run('ejemplo()')

```

Obtendremos un reporte que muestra el tiempo total y el número de llamadas para cada función.

Interpretación del Reporte de Profile

- **ncalls**: Número de veces que una función fue llamada.
- **tottime**: Tiempo total gastado en la función (excluye las sub-llamadas).
- **percall**: Tiempo promedio por llamada (tottime/ncalls).
- **cumtime**: Tiempo total incluyendo las sub-llamadas hechas por la función.
- **Filename** (function): Ubicación del código fuente y el nombre de la función.

```
         1000003 function calls in 0.253 seconds

   ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
   1    0.000    0.000    0.253    0.253 profile_test.py:4(ejemplo)
```

Consejos para Optimizar y Herramientas Avanzadas

- Se puede utilizar profile para identificar cuellos de botella en funciones específicas.
- Para proyectos grandes, consideremos usar cProfile (versión más rápida).
- Guarda los resultados en un archivo para análisis posterior y n requiere ensuciar el código

```
python3 -m cProfile -o salida.pstat my_script.py
```

Los archivos **salida.pstats** son binarios generados por cProfile con infromación de los resultados del analisis.

Consejos para Optimizar y Herramientas Avanzadas

Se puede utilizar pstats para analizar y ordenar los resultados:

```
import pstats
p = pstats.Stats('salida.pstats')
p.sort_stats('tottime').print_stats(10)
```

Se ordeno por “tottime”

```
Thu Aug 23 13:45:37 2024    salida.pstats

         7 function calls in 6.003 seconds

   Ordered by: internal time
   List reduced from 7 to 5 due to restriction <5>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        3    6.003    2.001    6.003    2.001 mi_script.py:4(slow_function)
        3    0.000    0.000    0.000    0.000 mi_script.py:7(fast_function)
        1    0.000    0.000    6.003    6.003 mi_script.py:9(main)
        1    0.000    0.000    6.003    6.003 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Analisis de `.pstats`

Se puede hacer directamente:

```
python3 -m pstats salida.pstat
```

Esto abrirá una interfaz interactiva donde podrás explorar el archivo. Algunos comandos útiles son:

- **`sort time`**: Ordena los resultados por el tiempo total de ejecución.
- **`sort cumulative`**: Ordena por el tiempo acumulado (funciones que llaman a otras).
- **`stats 10`**: Muestra las 10 primeras funciones con mayor tiempo de ejecución.

Ejemplo:

```
python3 -m pstats salida.pstat
(pstats) sort time
(pstats) stats 10
```

Otras herramientas para el analisis de `.pstats`

snakeviz

```
pip install snakeviz
```

Ejecución:

`snakeviz salida.pstat` → Abre un navegador y muestra resultados de forma grafica

gprof2dot

```
pip install gprof2dot
```

Ejecución:

```
gprof2dot -f pstats salida.pstat | dot -Tpng -o output.png
```

Genera una imagen de grafos con el flujo de llamadas