😊 Pylint and Unit Test Library

Pylint

Pylint is a static code analysis tool in Python that helps improve code quality. It scans your code without running it and detects errors, style violations, and potential issues. It checks for syntax errors, unused variables, incorrect naming conventions, and adherence to coding standards like PEP 8. Pylint assigns a score to your code, indicating how well it follows best practices. This helps developers write cleaner, more maintainable, and error-free code.

Use Cases of Pylint:

- Detecting syntax errors before execution
- Enforcing coding style guidelines
- Identifying unused variables or imports
- Ensuring better readability and maintainability

Unit Testing

Unit testing is a method of testing individual components (such as functions or classes) of a program to ensure they work correctly. Python's built-in unittest module provides a framework for writing test cases, running them automatically, and checking expected vs. actual results. If a function does not return the expected output, the test fails, allowing developers to detect and fix bugs early.

Use Cases of Unit Testing:

- Validating that a function returns correct results
- Detecting bugs before deploying code
- Automating repetitive testing processes
- Ensuring that future code changes do not break existing functionality

Benefits of Using Pylint and Unit Testing:

- **Early Bug Detection:** Identifies issues before execution (Pylint) and during testing (Unit Testing).
- **Code Quality Improvement:** Enforces clean, maintainable, and efficient code.
- **Saves Time and Effort:** Reduces debugging time by catching errors early.
- **Enhances Collaboration:** Ensures consistency across team-based development.
- **Prevents Regression Issues:** Ensures changes do not break existing features.

Both Pylint and Unit Testing are essential tools in software development, improving the reliability and maintainability of Python applications.

😄 To begin with the Lab

1. We start by installing **Pylint** in our Jupyter Notebook. Then we wrote a simple file where we created an error while printing.
2. This script creates a file named "simple1.py" with two variables: one named "a" set to 1 and another named "b" set to 2. It then prints the value of "a" correctly but attempts to print "B". Since Python is case-sensitive, "B" is not defined (only "b" is defined), leading to an error. Finally, the script runs pylint on the file, which will flag the undefined variable "B" along with any other style or error issues.

[1]:
```
!pip install pylint
```

```
Requirement already satisfied: pylint in c:\users\pulkit\anaconda3\lib\site-packages (2.16.2)
Requirement already satisfied: platformdirs>=2.2.0 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (3.10.0)
Requirement already satisfied: astroid<=2.16.0-dev0,>=2.14.2 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (2.14.2)
Requirement already satisfied: isort<6,>=4.2.5 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (5.9.3)
Requirement already satisfied: mccabe<0.8,>=0.6 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (0.7.0)
Requirement already satisfied: tomlkit>=0.10.1 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (0.11.1)
Requirement already satisfied: dill>=0.3.6 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (0.3.6)
Requirement already satisfied: colorama>=0.4.5 in c:\users\pulkit\anaconda3\lib\site-packages (from pylint) (0.4.6)
Requirement already satisfied: lazy-object-proxy>=1.4.0 in c:\users\pulkit\anaconda3\lib\site-packages (from astroid<=2.16.0-dev0,>=2.14.2->pylint) (1.6.
0)
Requirement already satisfied: wrapt<2,>=1.14 in c:\users\pulkit\anaconda3\lib\site-packages (from astroid<=2.16.0-dev0,>=2.14.2->pylint) (1.14.1)
```

[3]:
```
%%writefile simple1.py
a = 1
b = 2
print(a)
print(B)
```

```
Writing simple1.py
```

[5]:
```
! pylint simple1.py
```

```
************* Module simple1
simple1.py:1:0: C0114: Missing module docstring (missing-module-docstring)
simple1.py:1:0: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
simple1.py:2:0: C0103: Constant name "b" doesn't conform to UPPER_CASE naming style (invalid-name)
simple1.py:4:6: E0602: Undefined variable 'B' (undefined-variable)

-----------------------------------
Your code has been rated at 0.00/10
```

3. Pylint analyzes your code against a set of coding standards and best practices, such as proper naming conventions, code structure, and potential error detection. It assigns a rating (typically on a scale from 0 to 10) to indicate how well your code follows these guidelines. A lower rating suggests areas where your code could be improved for better readability, maintainability, and reliability.
4. In the code below, you can see that we have defined the proper syntax as we rewrote our file, and we get a 10/10 rating from Pylint.

[7]:
```
%%writefile simple1.py
"""
A very simple script.
"""

def myfunc():
    """
    An extremely simple function.
    """
    first = 1
    second = 2
    print(first)
    print(second)

myfunc()
```

```
Overwriting simple1.py
```

[9]:
```
! pylint simple1.py
```

```
--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 0.00/10, +10.00)
```

5. In this script, you have defined a simple function with proper docstrings and then called it. The function assigns values to two variables: one named "first" and another "second." It prints the value of "first," but instead of printing the value of the variable "second," it prints the literal string "second." This might be intentional or an oversight if you meant to display the value of the variable.
6. Based on that Pylint has rated our code.

[11]:
```
%%writefile simple2.py
"""
A very simple script.
"""


def myfunc():
    """
    An extremely simple function.
    """
    first = 1
    second = 2
    print(first)
    print('second')

myfunc()
```

```
Writing simple2.py
```

[13]:
```
! pylint simple2.py
```

```
************* Module simple2
simple2.py:10:4: W0612: Unused variable 'second' (unused-variable)

-----------------------------------
Your code has been rated at 8.33/10
```

7. This script is a **unit test** file for a module named "cap." It uses Python's unittest framework to define test cases for the function "cap_text" from the cap module. Two tests are provided: one checks that a single lowercase word is capitalized correctly, and the other checks that multiple words are capitalized appropriately. When you run the script, it executes the tests and reports any failures or errors.
8. The unit tests are run using the **unittest** framework. The first test, which passes a single word, succeeds because "python" becomes "Python" using the capitalize () method. However, the second test fails because the capitalize () method only capitalizes the first letter of the string and lowercases the rest. Therefore, "monty python" becomes "Monty python" (with a lowercase "p"), which does not match the expected "Monty Python".

[15]:
```
%%writefile cap.py
def cap_text(text):
    return text.capitalize()
```

```
Writing cap.py
```

[17]:
```
%%writefile test_cap.py
import unittest
import cap

class TestCap(unittest.TestCase):

    def test_one_word(self):
        text = 'python'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Python')

    def test_multiple_words(self):
        text = 'monty python'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Monty Python')

if __name__ == '__main__':
    unittest.main()
```

```
Writing test_cap.py
```

[19]:
```
! python test_cap.py
```

```
F.
======================================================================
FAIL: test_multiple_words (__main__.TestCap.test_multiple_words)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\PULKIT\project\test_cap.py", line 14, in test_multiple_words
    self.assertEqual(result, 'Monty Python')
AssertionError: 'Monty python' != 'Monty Python'
- Monty python
?       ^
+ Monty Python
?       ^


----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

9. By replacing the call to .capitalize () with .title () in cap.py, the cap_text function now capitalizes the first letter of each word. When you run the test file, both tests pass

because the function returns "Python" for the single word and "Monty Python" for the multiple-word input.

[21]:
```
%%writefile cap.py
def cap_text(text):
    return text.title()  # replace .capitalize() with .title()
```

```
Overwriting cap.py
```

[23]:
```
! python test_cap.py
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

This test file defines three test cases using Python's unittest framework to verify the behavior of the cap_text function from cap.py. The tests check that:

1. A single word ("python") becomes "Python".
2. A two-word phrase ("monty python") becomes "Monty Python".
3. A phrase with an apostrophe ("monty python's flying circus") becomes "Monty Python's Flying Circus".

When you run the test file, all tests should pass, confirming that the cap_text function correctly converts text to title case.

[25]:
```
%%writefile test_cap.py
import unittest
import cap

class TestCap(unittest.TestCase):

    def test_one_word(self):
        text = 'python'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Python')

    def test_multiple_words(self):
        text = 'monty python'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Monty Python')

    def test_with_apostrophes(self):
        text = "monty python's flying circus"
        result = cap.cap_text(text)
        self.assertEqual(result, "Monty Python's Flying Circus")

if __name__ == '__main__':
    unittest.main()
```

```
Overwriting test_cap.py
```

[27]:
```
! python test_cap.py
```

```
..F
======================================================================
FAIL: test_with_apostrophes (__main__.TestCap.test_with_apostrophes)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\PULKIT\project\test_cap.py", line 19, in test_with_apostrophes
    self.assertEqual(result, "Monty Python's Flying Circus")
AssertionError: "Monty Python'S Flying Circus" != "Monty Python's Flying Circus"
- Monty Python'S Flying Circus
?              ^
+ Monty Python's Flying Circus
?              ^


----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1)
```