Software Engineering (IT314)
Lab : 5

ID : 202001176
Name : Jenish Mangukiya

Date : 16 / 03 / 2023

- Static Analysis tool : **pylint**

S.No	Message Object	Expansion	Explanation
1.	C	Convention	It is displayed when the program is not following the standard rules.
2.	R	Refactor	It is displayed for bad code smell
3.	W	Warning	It is displayed for python specific problems
4.	E	Error	It is displayed when that particular line execution results some error
5.	F	Fatal	It is displayed when pylint has no access to further process that line.

Let’s discuss some techniques to improve your score.

- ID **C0326** suggests a bad-white space error means we need to give a whitespace between a and = symbol. This rule is applicable to all declarations where an operator is used immediately after an identifier.
- ID **C0304** comes under missing-new-line suggestion which means we have to add a blank line when we complete our code.
- ID **C0114** comes under missing-module-docstring suggestion which means we need to add a docstring at the top which refers to the use of the program written below that.
- ID **C0103** comes under invalid-name suggestion which can be avoided by writing the identifiers starting with a capital letter. But, we usually believe that class names use CamelCasing i.e class names start with an upper-case letter. To avoid this suggestion we will add a regular expression to pylint that actually accepts all the variables in the lowercase letters. We will discuss this more in the further examples.

  Here is the references

1 : Repo link

Code:

```python
def add(a: float, b: float) -> float:

    return a + b

if __name__ == "__main__":

    a = 5

    b = 6

    print(f"The sum of {a} + {b} is {add(a, b)}")
```

Output:

```
PS C:\Users\student\Desktop\lab-5> py -m pylint test.py
************* Module test
test.py:8:0: C0304: Final newline missing (missing-final-newline)
test.py:1:0: C0114: Missing module docstring (missing-module-docstring)
test.py:2:0: C0116: Missing function or method docstring (missing-function-docstring)
test.py:2:8: C0103: Argument name "a" doesn't conform to snake_case naming style (invalid-name)
test.py:2:18: C0103: Argument name "b" doesn't conform to snake_case naming style (invalid-name)
test.py:2:8: W0621: Redefining name 'a' from outer scope (line 6) (redefined-outer-name)
test.py:2:18: W0621: Redefining name 'b' from outer scope (line 7) (redefined-outer-name)
test.py:6:4: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
test.py:7:4: C0103: Constant name "b" doesn't conform to UPPER_CASE naming style (invalid-name)

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

PS C:\Users\student\Desktop\lab-5>
```

Analysis:

- All errors are true

Error types:

- **Missing final newline.**
  - We have to add a new line when our code is complete.
- **Missing module docstring**
  - At the start of the code we add a string(comment) which indicates the use of our programme.
- **Missing function docstring**
  - At the start of the function we add a string(comment) which indicates the use of our function.
- **Snake_case naming style**
  - It shows that we have to name our variable in proper format.
- **Redefined outer name**
  - It shows that we give different variables the same name.
- **UPPER_CASE naming style**
  - It shows that we have to name our variable in proper format.

After Improvement

Code:

```
"Programme for add two numbers"

def add(num1: float, num2: float) -> float:

    "Function for add two numbers"

    return num1 + num2

if __name__ == "__main__":

    N_1 = 5

    N_2 = 6

    print(f"The sum of {N_1} + {N_2} is {add(N_1, N_2)}")
```

Output:

```
PS C:\Users\student\Desktop\lab-5> py -m pylint test.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 6.67/10, +3.33)

PS C:\Users\student\Desktop\lab-5>
```

2 : [Repo link]

Code:

```
from math import pi

def arc_length(angle: int, radius: int) -> float:

    """

    >>> arc_length(45, 5)

    3.9269908169872414

    >>> arc_length(120, 15)

    31.415926535897928

    """

    return 2 * pi * radius * (angle / 360)

if __name__ == "__main__":

    print(arc_length(90, 10))
```

Output:

```
PS C:\Users\student\Desktop\lab-5> py -m pylint test.py
************* Module test
test.py:15:0: C0304: Final newline missing (missing-final-newline)
test.py:1:0: C0114: Missing module docstring (missing-module-docstring)

-------------------------------------------------------------------
Your code has been rated at 6.00/10 (previous run: 0.00/10, +6.00)

PS C:\Users\student\Desktop\lab-5>
```

Analysis:

- All errors are true

Error types:

- **Missing final newline.**
  - We have to add a new line when our code is complete.
- **Missing module docstring**
  - At the start of the code we add a string(comment) which indicates the use of our programme.

3 : [Repo link]

Code:

```
"""

Approximates the area under the curve using the trapezoidal rule

"""

from __future__ import annotations

from collections.abc import Callable

def trapezoidal_area(

    fnc: Callable[[int | float], int | float],

    x_start: int | float,

    x_end: int | float,

    steps: int = 100,

) -> float:

    """

    Treats curve as a collection of linear lines and sums the
area of the

    trapezium shape they form

    :param fnc: a function which defines a curve
```

```
    :param x_start: left end point to indicate the start of line
segment

    :param x_end: right end point to indicate end of line
segment

    :param steps: an accuracy gauge; more steps increases the
accuracy

    :return: a float representing the length of the curve

    >>> def f(x):

    ...     return 5

    >>> f"{trapezoidal_area(f, 12.0, 14.0, 1000):.3f}"

    '10.000'

    >>> def f(x):

    ...     return 9*x**2

    >>> f"{trapezoidal_area(f, -4.0, 0, 10000):.4f}"

    '192.0000'

    >>> f"{trapezoidal_area(f, -4.0, 4.0, 10000):.4f}"

    '384.0000'

    """

    x1 = x_start
```

```
    fx1 = fnc(x_start)

    area = 0.0

    for _ in range(steps):

        # Approximates small segments of curve as linear and
solve

        # for trapezoidal area

        x2 = (x_end - x_start) / steps + x1

        fx2 = fnc(x2)

        area += abs(fx2 + fx1) * (x2 - x1) / 2

        # Increment step

        x1 = x2

        fx1 = fx2

    return area

if __name__ == "__main__":

    def f(x):

        return x**3 + x**2

    print("f(x) = x^3 + x^2")

    print("The area between the curve, x = -5, x = 5 and the x
axis is:")
```

```
    i = 10

    while i <= 100000:

        print(f"with {i} steps: {trapezoidal_area(f, -5, 5,
i)}")

        i *= 10
```

Output:

```
PS C:\Users\student\Desktop\lab-5> py -m pylint test.py
************* Module test
test.py:60:0: C0304: Final newline missing (missing-final-newline)
test.py:35:4: C0103: Variable name "x1" doesn't conform to snake_case naming style (invalid-name)
test.py:41:8: C0103: Variable name "x2" doesn't conform to snake_case naming style (invalid-name)
test.py:45:8: C0103: Variable name "x1" doesn't conform to snake_case naming style (invalid-name)
test.py:52:4: C0116: Missing function or method docstring (missing-function-docstring)
test.py:52:4: C0103: Function name "f" doesn't conform to snake_case naming style (invalid-name)
test.py:52:10: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 6.82/10 (previous run: 6.00/10, +0.82)

PS C:\Users\student\Desktop\lab-5>
```

Analysis:

- All errors are true

Error types:

- **Missing final newline.**
  - We have to add a new line when our code is complete.
- **Missing function docstring**
  - At the start of the function we add a string(comment) which indicates the use of our function.
- **Snake_case naming style**
  - It shows that we have to name our variable in proper format.

4 : [Repo link]

Code:

```
"""Binary Exponentiation."""

# Time Complexity : O(logn)

def binary_exponentiation(a, n):

    if n == 0:

        return 1

    elif n % 2 == 1:

        return binary_exponentiation(a, n - 1) * a

    else:

        b = binary_exponentiation(a, n / 2)

        return b * b

if __name__ == "__main__":

    try:

        BASE = int(input("Enter Base : ").strip())

        POWER = int(input("Enter Power : ").strip())

    except ValueError:

        print("Invalid literal for integer")
```

```
    RESULT = binary_exponentiation(BASE, POWER)

    print(f"{BASE}^({POWER}) : {RESULT}")
```

Output:

```
PS C:\Users\student\Desktop\lab-5> py -m pylint test.py
************* Module test
test.py:27:0: C0304: Final newline missing (missing-final-newline)
test.py:7:0: C0116: Missing function or method docstring (missing-function-docstring)
test.py:7:26: C0103: Argument name "a" doesn't conform to snake_case naming style (invalid-name)
test.py:7:29: C0103: Argument name "n" doesn't conform to snake_case naming style (invalid-name)
test.py:8:4: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)
test.py:15:8: C0103: Variable name "b" doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 6.00/10 (previous run: 10.00/10, -4.00)

PS C:\Users\student\Desktop\lab-5>
```

Analysis:

- All errors are true

Error types:

- **Missing final newline.**
  - We have to add a new line when our code is complete.
- **Missing function docstring**
  - At the start of the function we add a string(comment) which indicates the use of our function.
- **Snake_case naming style**
  - It shows that we have to name our variable in proper format.
- **No-else-return**
  - In the function there was unnecessary else at the end.

5 : [Repo link]

Code:

```
def bin_exp_mod(a, n, b):

    """

    >>> bin_exp_mod(3, 4, 5)

    1

    >>> bin_exp_mod(7, 13, 10)

    7

    """

    # mod b

    assert b != 0, "This cannot accept modulo that is == 0"

    if n == 0:

        return 1

    if n % 2 == 1:

        return (bin_exp_mod(a, n - 1, b) * a) % b

    r = bin_exp_mod(a, n / 2, b)

    return (r * r) % b

if __name__ == "__main__":
```

```
try:

    BASE = int(input("Enter Base : ").strip())

    POWER = int(input("Enter Power : ").strip())

    MODULO = int(input("Enter Modulo : ").strip())

except ValueError:

    print("Invalid literal for integer")

print(bin_exp_mod(BASE, POWER, MODULO))
```

Output:

```
PS C:\Users\student\Desktop\lab-5> py -m pylint test.py
************* Module test
test.py:28:0: C0304: Final newline missing (missing-final-newline)
test.py:1:0: C0114: Missing module docstring (missing-module-docstring)
test.py:1:16: C0103: Argument name "a" doesn't conform to snake_case naming style (invalid-name)
test.py:1:19: C0103: Argument name "n" doesn't conform to snake_case naming style (invalid-name)
test.py:1:22: C0103: Argument name "b" doesn't conform to snake_case naming style (invalid-name)
test.py:16:4: C0103: Variable name "r" doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 6.25/10 (previous run: 6.00/10, +0.25)

PS C:\Users\student\Desktop\lab-5>
```

Analysis:

- All errors are true

Error types:

- **Missing final newline.**
  - We have to add a new line when our code is complete.
- **Missing module docstring**
  - At the start of the code we add a string(comment) which indicates the use of our programme.
- **Snake_case naming style**
  - It shows that we have to name our variable in proper format.