

Software Engineering (IT314)

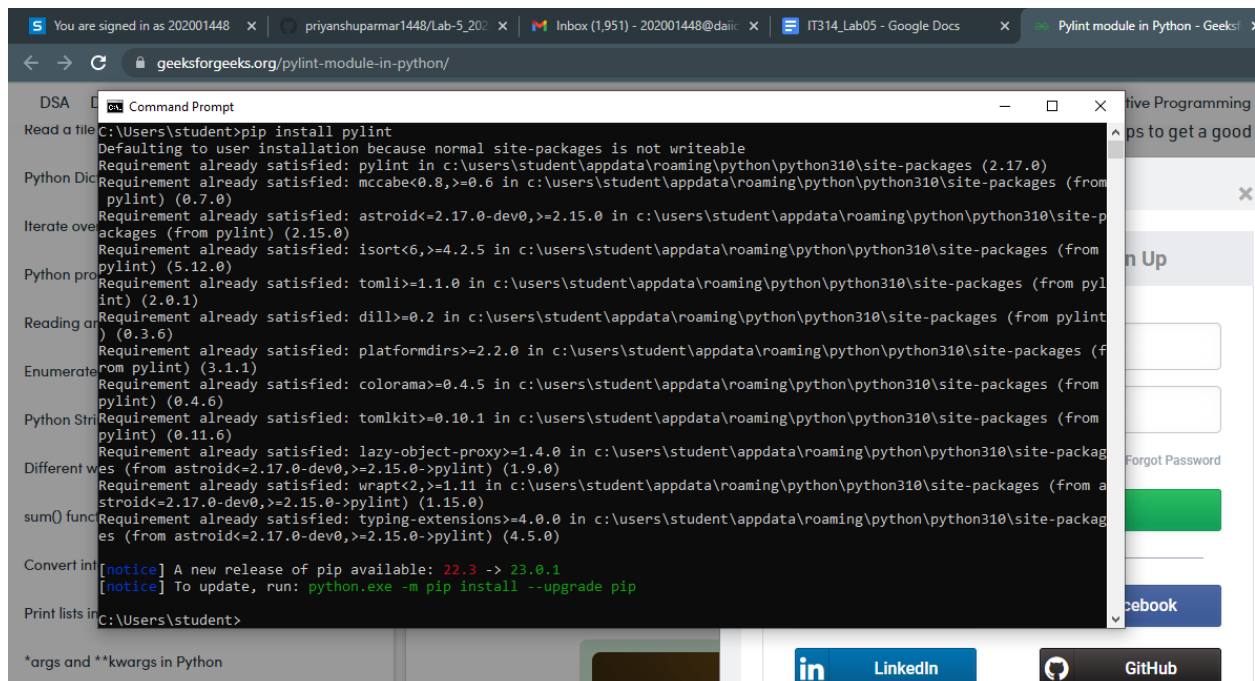
Lab : 5

Name : Priyanshu Parmar
ID : 202001448

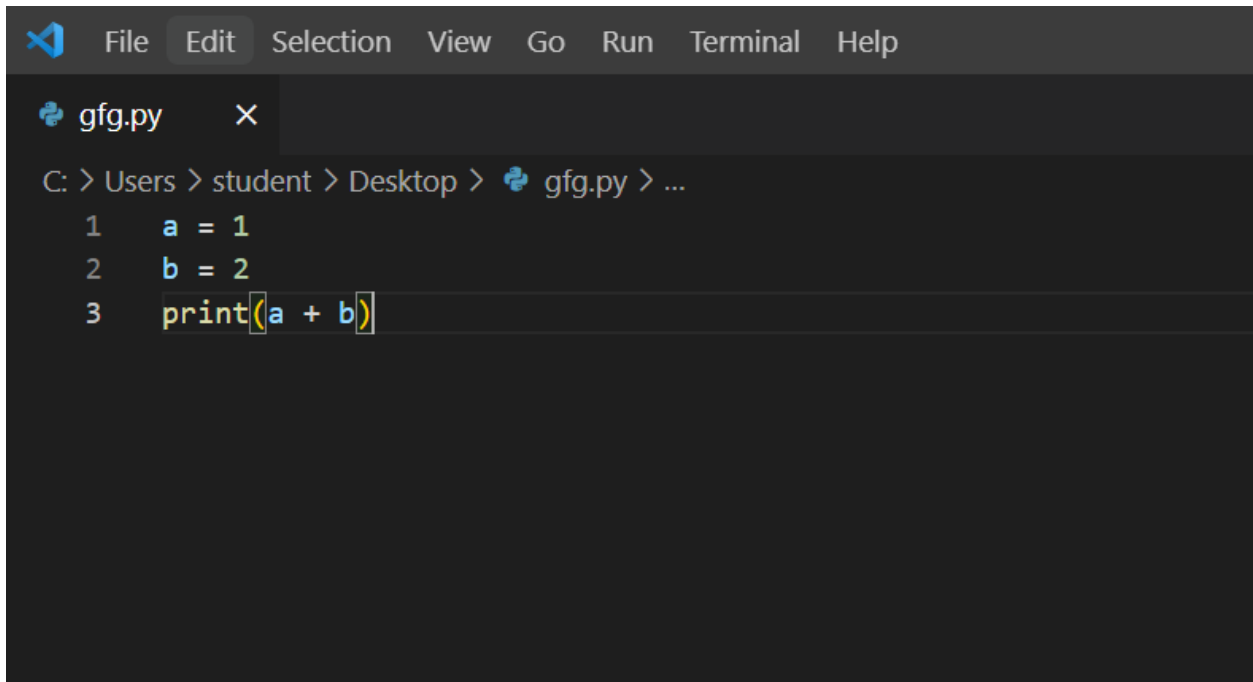
Date : 24 / 03 / 2023

- Static Analysis tool : pylint

```
pip install pylint
```



The screenshot shows a web browser window with the URL <https://www.geeksforgeeks.org/pylint-module-in-python/>. Overlaid on the browser is a Windows Command Prompt window. The Command Prompt shows the execution of the command `pip install pylint`. The output indicates that pylint is already installed at version 2.17.0. It then lists several dependencies that are either already installed or being installed: mccabe (0.8.0), astroid (2.17.0-dev0), isort (6.0.0), pylint (0.7.0), pylint (5.12.0), toml (1.1.0), dill (0.3.6), platformdirs (2.2.0), colorama (0.4.5), tomlkit (0.10.1), lazy-object-proxy (1.4.0), wrapt (2.1.1), typing-extensions (4.0.0), and astroid (2.17.0-dev0). At the bottom, a notice states: "[notice] A new release of pip available: 22.3 -> 23.0.1" and "[notice] To update, run: python.exe -m pip install --upgrade pip".

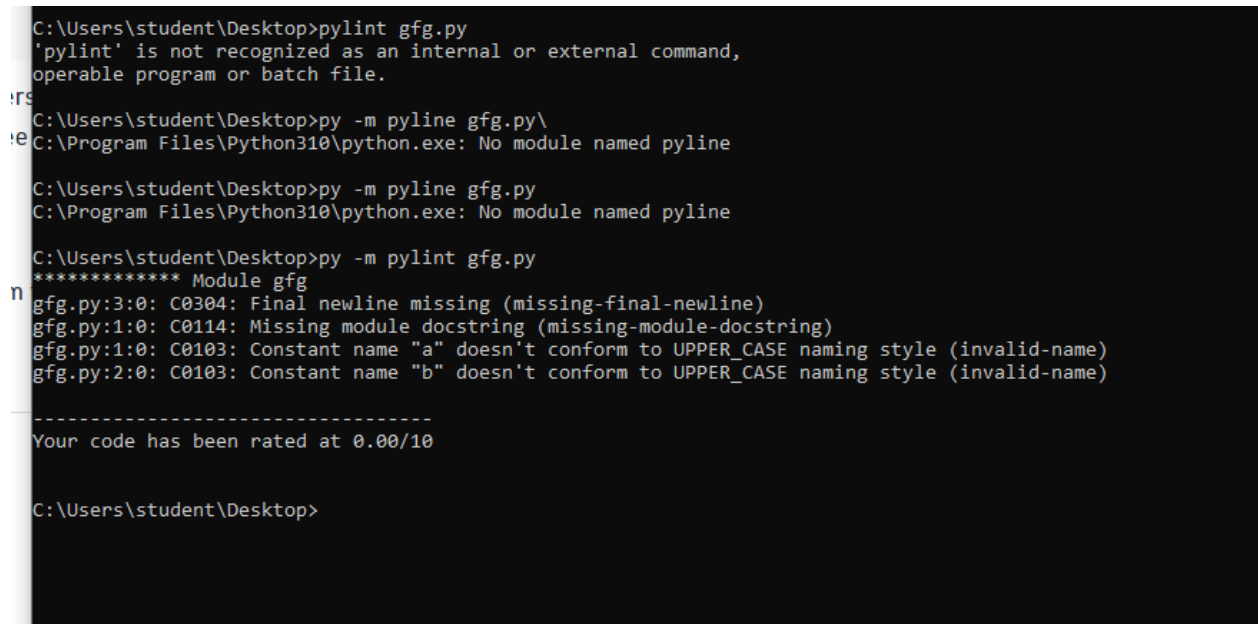


```
File Edit Selection View Go Run Terminal Help

gfg.py x

C: > Users > student > Desktop > gfg.py > ...

1 a = 1
2 b = 2
3 print(a + b)
```



```
C:\Users\student\Desktop>pylint gfg.py
'pylint' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\student\Desktop>py -m pylint gfg.py
C:\Program Files\Python310\python.exe: No module named pylint

C:\Users\student\Desktop>py -m pylint gfg.py
C:\Program Files\Python310\python.exe: No module named pylint

C:\Users\student\Desktop>py -m pylint gfg.py
***** Module gfg
gfg.py:3:0: C0304: Final newline missing (missing-final-newline)
gfg.py:1:0: C0114: Missing module docstring (missing-module-docstring)
gfg.py:1:0: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
gfg.py:2:0: C0103: Constant name "b" doesn't conform to UPPER_CASE naming style (invalid-name)

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

C:\Users\student\Desktop>
```

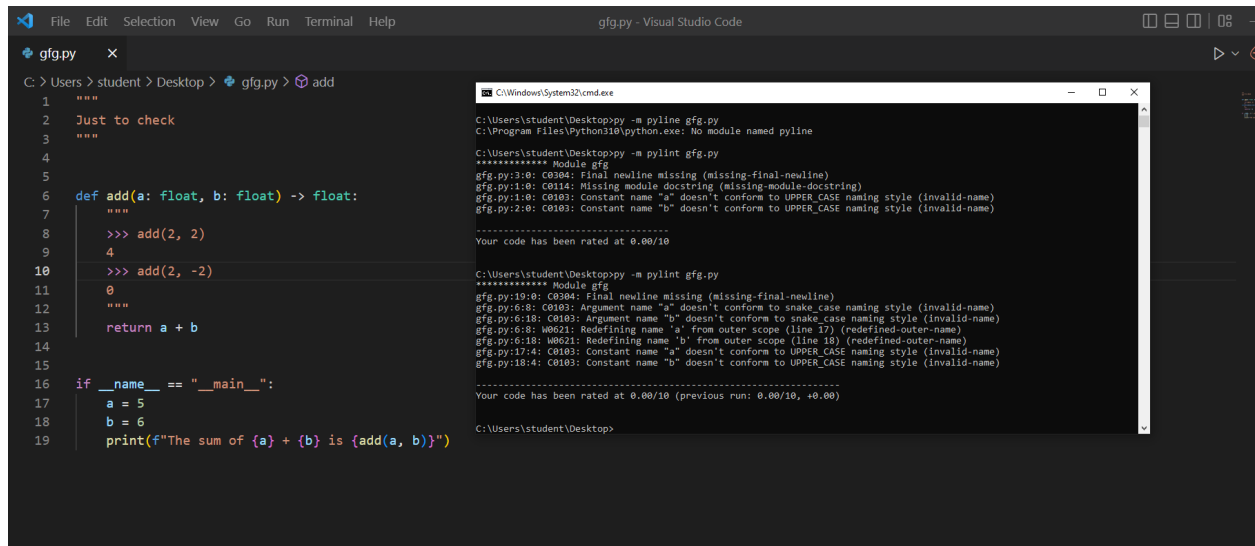
S.No	Message Object	Expansion	Explanation
1.	C	Convention	It is displayed when the program is not following the standard rules.
2.	R	Refactor	It is displayed for bad code smell
3.	W	Warning	It is displayed for python specific problems
4.	E	Error	It is displayed when that particular line execution results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

Here is the [references](#)

Example 1:

repo link:

<https://github.com/TheAlgorithms/Python/blob/master/maths/add.py>



The screenshot shows the Visual Studio Code editor with a file named 'gfg.py' open. The file contains a Python function 'add' and a main block. The terminal window shows the output of running the file with 'pylint', displaying various error messages such as 'Final newline missing', 'Missing module docstring', and 'Constant name doesn't conform to UPPER_CASE naming style'.

```
1 """
2 Just to check
3 """
4
5
6 def add(a: float, b: float) -> float:
7     """
8     >>> add(2, 2)
9     4
10    >>> add(2, -2)
11    0
12    """
13     return a + b
14
15
16 if __name__ == "__main__":
17     a = 5
18     b = 6
19     print(f"The sum of {a} + {b} is {add(a, b)}")
```

```
C:\Users\student\Desktop> python gfg.py
C:\Program Files\Python310\python.exe: No module named pylint

C:\Users\student\Desktop> python -m pylint gfg.py
***** Module gfg
gfg.py:3:0: C0304: Final newline missing (missing-final-newline)
gfg.py:1:0: C0114: Missing module docstring (missing-module-docstring)
gfg.py:11:0: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
gfg.py:12:0: C0103: Constant name "b" doesn't conform to UPPER_CASE naming style (invalid-name)

Your code has been rated at 0.00/10

C:\Users\student\Desktop> python -m pylint gfg.py
***** Module gfg
gfg.py:19:0: C0304: Final newline missing (missing-final-newline)
gfg.py:6:8: C0103: Argument name "a" doesn't conform to snake case naming style (invalid-name)
gfg.py:6:18: C0103: Argument name "b" doesn't conform to snake case naming style (invalid-name)
gfg.py:6:8: W0621: Redefining name 'a' from outer scope (line 17) (redefined-outer-name)
gfg.py:6:18: W0621: Redefining name 'b' from outer scope (line 18) (redefined-outer-name)
gfg.py:17:4: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
gfg.py:18:4: 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)
```

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.

Example 2:

repo link:

https://github.com/TheAlgorithms/Python/blob/master/maths/arc_length.py

The screenshot shows a Visual Studio Code editor with a file named `gfg.py` open. The code in the file is as follows:

```
1 from math import pi
2
3
4 def arc_length(angle: int, radius: int) -> float:
5     """
6     >>> arc_length(45, 5)
7     3.9269908169872414
8     >>> arc_length(120, 15)
9     31.415926535897928
10    """
11    return 2 * pi * radius * (angle / 360)
12
13
14 if __name__ == "__main__":
15     print(arc_length(90, 10))
```

The terminal window shows the output of the code:

```
C:\Windows\System32\cmd.exe
gfg.py:6:8: W0621: Redefining name 'a' from outer scope (line 17) (redefined-outer-name)
gfg.py:6:18: W0621: Redefining name 'b' from outer scope (line 18) (redefined-outer-name)
gfg.py:17:4: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
gfg.py:18:4: 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)

C:\Users\student\Desktop>py -m pylint gfg.py
***** Module gfg
gfg.py:15:0: C0304: Final newline missing (missing-final-newline)
gfg.py:11:0: C0114: Missing module docstring (missing-module-docstring)

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

C:\Users\student\Desktop>
```

Analysis:

- All errors are true

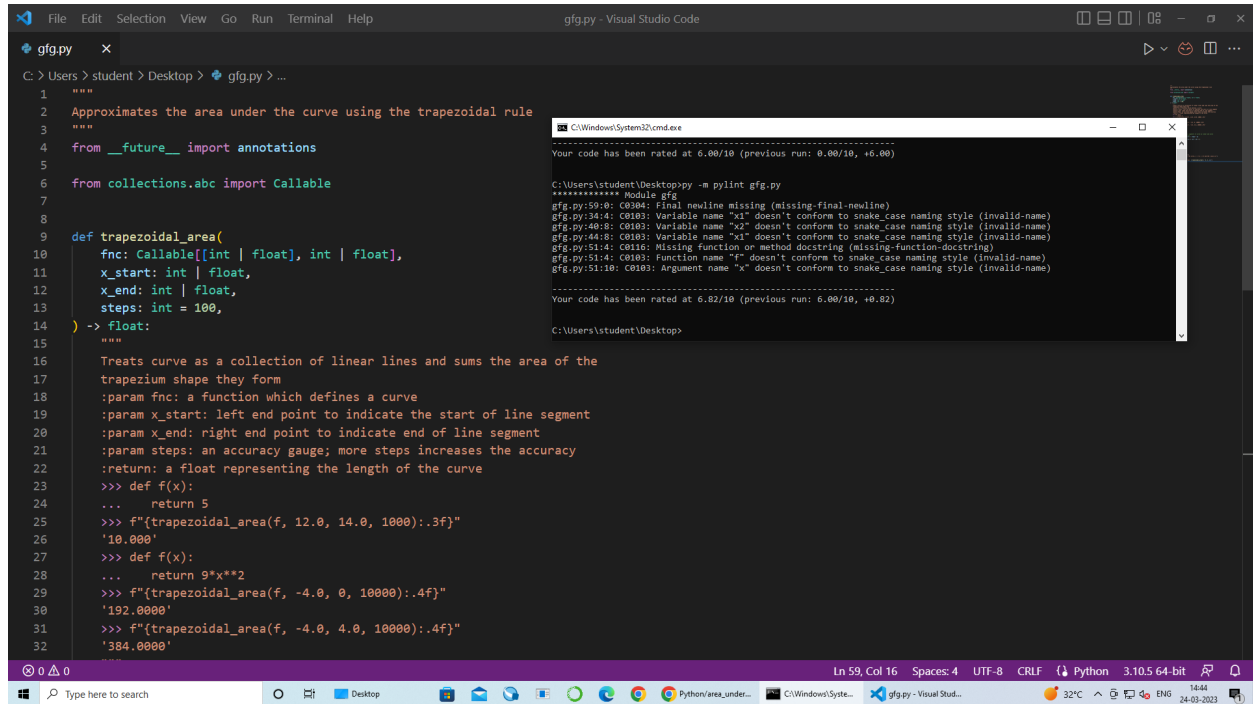
Error types:

- **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.

Example 3:

repo link:

https://github.com/TheAlgorithms/Python/blob/master/maths/area_under_curve.py



```
1 """
2 Approximates the area under the curve using the trapezoidal rule
3 """
4 from __future__ import annotations
5
6 from collections.abc import Callable
7
8
9 def trapezoidal_area(
10     fnc: Callable[[int | float], int | float],
11     x_start: int | float,
12     x_end: int | float,
13     steps: int = 100,
14 ) -> float:
15     """
16     Treats curve as a collection of linear lines and sums the area of the
17     trapezium shape they form
18     :param fnc: a function which defines a curve
19     :param x_start: left end point to indicate the start of line segment
20     :param x_end: right end point to indicate end of line segment
21     :param steps: an accuracy gauge; more steps increases the accuracy
22     :return: a float representing the length of the curve
23     """
24     >>> def f(x):
25     ...     return 5
26     >>> f"{trapezoidal_area(f, 12.0, 14.0, 1000):.3f}"
27     '10.000'
28     >>> def f(x):
29     ...     return 9*x**2
30     >>> f"{trapezoidal_area(f, -4.0, 0, 10000):.4f}"
31     '192.0000'
32     >>> f"{trapezoidal_area(f, -4.0, 4.0, 10000):.4f}"
33     '384.0000'
```

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
```

Analysis:

- All errors are true

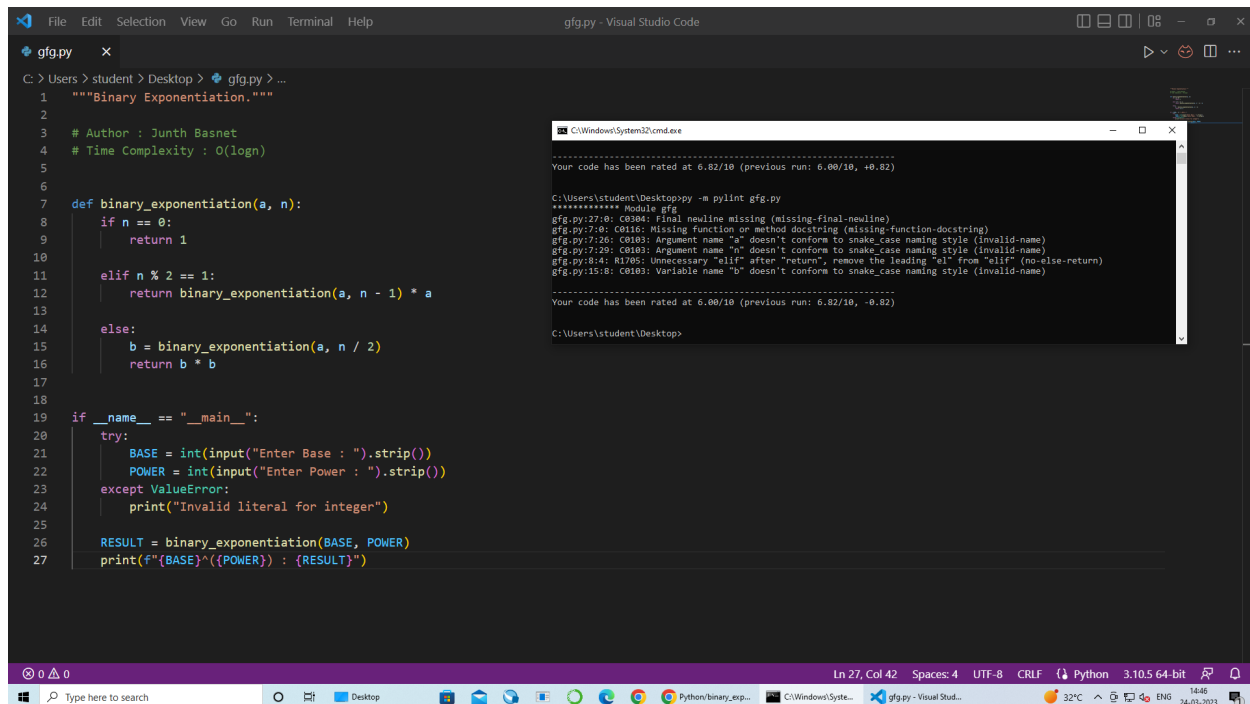
Error types:

- **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.

Example 4:

repo link:

https://github.com/TheAlgorithms/Python/blob/master/maths/binary_exponentiation.py



The screenshot shows a Visual Studio Code editor with a file named `gfg.py` open. The code is a Python function for binary exponentiation. The terminal window shows the output of running the code, which is a list of Pylint errors. The errors are:

- `gfg.py:12:0: C0103: Final newline missing (missing-final-newline)`
- `gfg.py:7:8: C0116: Missing function or method docstring (missing-function-docstring)`
- `gfg.py:7:26: C0103: Argument name "a" doesn't conform to snake_case naming style (invalid-name)`
- `gfg.py:7:29: C0103: Argument name "n" doesn't conform to snake_case naming style (invalid-name)`
- `gfg.py:8:4: R1705: Unnecessary "elif" after "return", remove the leading "elif" (no-else-return)`
- `gfg.py:15:8: C0103: Variable name "b" doesn't conform to snake_case naming style (invalid-name)`

```
1 """Binary Exponentiation."""
2
3 # Author : Junth Basnet
4 # Time Complexity : O(logn)
5
6
7 def binary_exponentiation(a, n):
8     if n == 0:
9         return 1
10
11     elif n % 2 == 1:
12         return binary_exponentiation(a, n - 1) * a
13
14     else:
15         b = binary_exponentiation(a, n / 2)
16         return b * b
17
18
19 if __name__ == "__main__":
20     try:
21         BASE = int(input("Enter Base : ").strip())
22         POWER = int(input("Enter Power : ").strip())
23     except ValueError:
24         print("Invalid literal for integer")
25
26     RESULT = binary_exponentiation(BASE, POWER)
27     print(f"{BASE}^{POWER} : {RESULT}")
```

Analysis:

- All errors are true

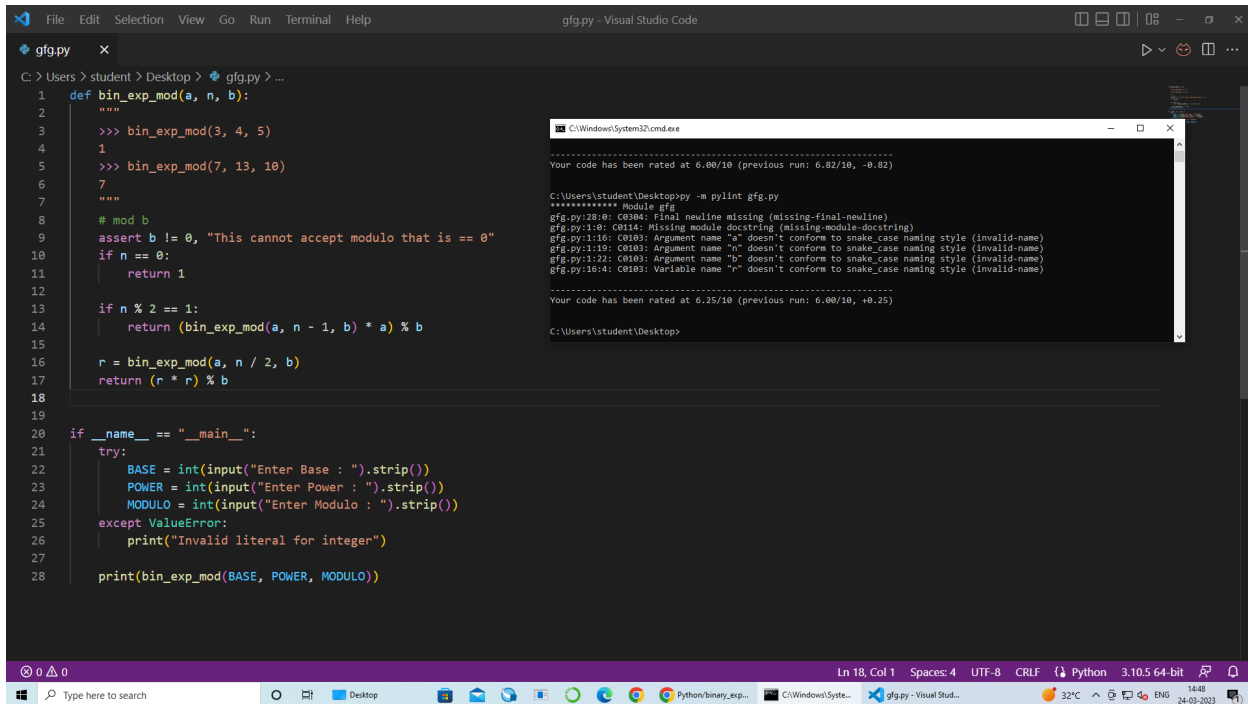
Error types:

- **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.

Example 5:

repo link:

https://github.com/TheAlgorithms/Python/blob/master/maths/binary_exp_mod.py



The screenshot shows a Visual Studio Code editor with a file named `gfg.py` open. The code in the editor is a Python function `bin_exp_mod(a, n, b)` that calculates the binary exponentiation of `a` to the power of `n` modulo `b`. The code includes a docstring, an assertion, and a recursive implementation. Below the function, there is a main block that takes user input for base, power, and modulo, and prints the result. A terminal window is open, showing the output of the command `pylint gfg.py`. The terminal output displays several pylint errors: `gfg.py:18:0: C0304: final newline missing (missing-final-newline)`, `gfg.py:18:0: C0116: Missing module docstring (missing-module-docstring)`, `gfg.py:11:16: C0103: Argument name 'a' doesn't conform to snake_case naming style (invalid-name)`, `gfg.py:11:19: C0103: Argument name 'n' doesn't conform to snake_case naming style (invalid-name)`, `gfg.py:11:22: C0103: Argument name 'b' doesn't conform to snake_case naming style (invalid-name)`, and `gfg.py:16:4: C0103: Variable name 'r' doesn't conform to snake_case naming style (invalid-name)`. The terminal also shows the code's quality rating: `Your code has been rated at 6.00/10 (previous run: 6.82/10, -0.82)` and `6.25/10 (previous run: 6.00/10, +0.25)`.

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.