1. Assignment Description

This assignment applies the testing techniques of static code testing and analysis. The objective of this assignment is to apply the techniques from the lecture to static testing of the classify Triangles program.

---

2. Author- Ryan Davis

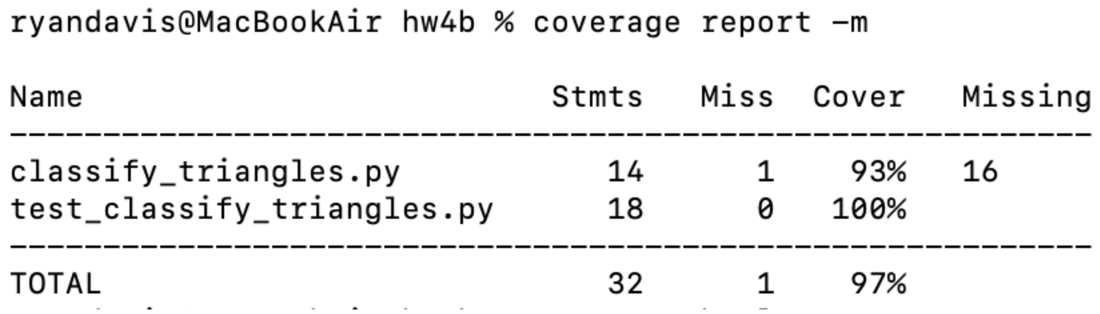
---

3. Summary

Results:

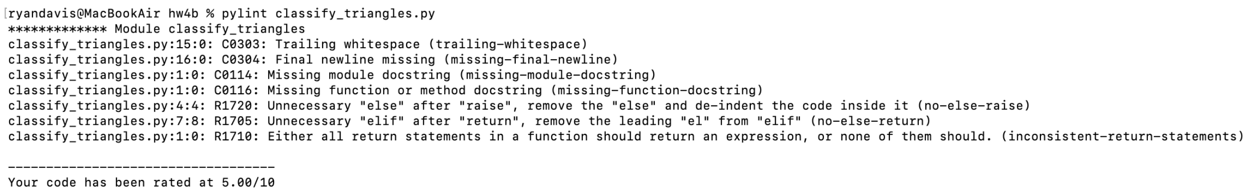
GitHub Repository containing triangle classification: https://github.com/ryry91021/ssw567/tree/main/hw/hw4/hw4b

- Initial Code Analysis:
  - Coverage (Coverage.py):



```
ryandavis@MacBookAir hw4b % coverage report -m

Name                           Stmts   Miss  Cover   Missing
------------------------------------------------------------
classify_triangles.py             14      1    93%   16
test_classify_triangles.py        18      0   100%
------------------------------------------------------------
TOTAL                             32      1    97%
```

  - Static Code Analysis (Pylint):



```
ryandavis@MacBookAir hw4b % pylint classify_triangles.py
************* Module classify_triangles
classify_triangles.py:15:0: C0303: Trailing whitespace (trailing-whitespace)
classify_triangles.py:16:0: C0304: Final newline missing (missing-final-newline)
classify_triangles.py:1:0: C0114: Missing module docstring (missing-module-docstring)
classify_triangles.py:1:0: C0116: Missing function or method docstring (missing-function-docstring)
classify_triangles.py:4:4: R1720: Unnecessary "else" after "raise", remove the "else" and de-indent the code inside it (no-else-raise)
classify_triangles.py:7:8: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)
classify_triangles.py:1:0: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)

-----------------------------------
Your code has been rated at 5.00/10
```

- Original Code:

Original testing Code:

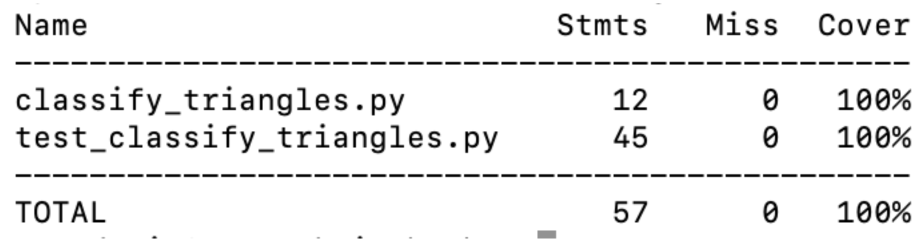
```
import unittest

from classify_triangles import classify_triangle

class testTriangle(unittest.TestCase):

    def test_isEquilateral(self):

        self.assertEqual(classify_triangle(500, 500, 500), "equilateral")
    def test_isIsosceles(self):
        self.assertEqual(classify_triangle(6, 6, 8), "isosceles")
    def test_isScalene(self):
        self.assertEqual(classify_triangle(15, 34, 32), "scalene")
    def test_isRight(self):
        self.assertEqual(classify_triangle(3, 4, 5), "right")
    def test_negativeLengthError(self):
        with self.assertRaises(ValueError):
            classify_triangle(-1, 2, 5)
    def test_improperTriangle(self):
        with self.assertRaises(ValueError):
            classify_triangle(1000, 1, 1)
if __name__ == '__main__':
    unittest.main()
```

Original classify triangle:

```
def classify_triangle(a,b,c):
    if any(side <= 0 for side in [a, b, c]):
        raise ValueError("Lengths must be positive")
    if not (a + b > c and a + c > b and b + c > a):
        raise ValueError("This is not a working triangle")
    else:
        if a==b==c:
            return "equilateral"
        elif a==b!=c or a!=b==c or a==c!=b:
            return "isosceles"
        elif ((a**2)+(b**2)==(c**2)) or ((b**2)+(c**2)==(a**2)) or ((a**2)+(c**2)==(c**2)):
            return "right"
```

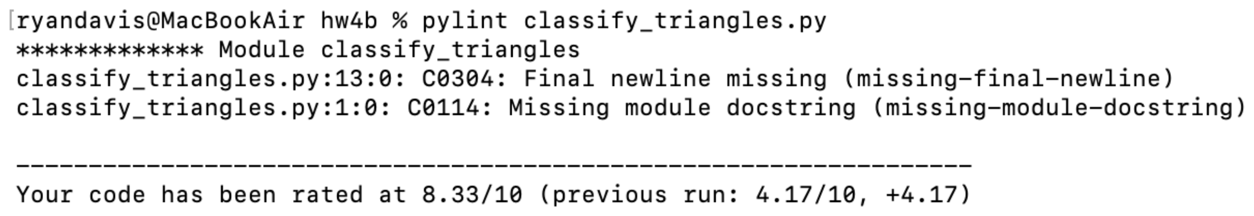
```
elif a!=b!=c:
    return "scalene"


return
```

- Changes applied:
    - Classify_triangle.py:
        - Removed trailing whitespace.
        - Added a final newline at the end of the file.
        - Added a function docstring.
        - Removed unnecessary else after raise.
        - Replaced elif with if where return already exits the function.
        - Ensured all return statements have a return value
    - Test_classify_triangles.py:
        - Covered different placements of the two equal sides in isosceles triangles.
        - Added more unique cases for scalene triangles.
        - Included more known Pythagorean triples for right triangles.
        - Checked behavior when invalid side lengths, including negative and zero values, are given.
        - Ensured that improper triangles violating the triangle inequality are correctly rejected.
        - Checked if floating-point numbers work correctly.
- Post-change code analysis:
    - Coverage:



```
Name                             Stmts   Miss  Cover
----------------------------------------------------
classify_triangles.py               12      0   100%
test_classify_triangles.py          45      0   100%
----------------------------------------------------
TOTAL                               57      0   100%
```

- Static Code Analysis:



```
[ryandavis@MacBookAir hw4b % pylint classify_triangles.py
************* Module classify_triangles
classify_triangles.py:13:0: C0304: Final newline missing (missing-final-newline)
classify_triangles.py:1:0: C0114: Missing module docstring (missing-module-docstring)

------------------------------------------------------------------
Your code has been rated at 8.33/10 (previous run: 4.17/10, +4.17)
```

Reflection:

- What I learned:
  - How to use Coverage in order to determine that test cases are properly testing the function
  - How to use static code analysis in order to improve functions and code.
- What went well:
  - Coverage and static code analysis with Pylint successfully addressed issues in the code.
- Challenges faced:
  - Determining which file to run coverage and pylint on was confusing, however I was able to learn that coverage is ran on the test cases, and pylint can be ran on both to find issues with each program.
  - I was able to attain an improvement in both coverage and improved the code's rating in static code analysis.

---

5. Honor Pledge

"I pledge my honor that I have abided by the Stevens Honor System."

Signed: *Ryan Davis*
