Unit 6

Seminar 3

Using Linters to Support Python Testing

Question 1: Run `styleLint.py` in Codio.

- What happens when the code is run?

```
codio@circle-marble:~/workspace$ python styleLint.py
  File "styleLint.py", line 5
    """ Return factorial of n """
                                ^
IndentationError: expected an indented block
```

The original source code is not correctly indented and so running the code results in an an IndentationError.

- Can you modify this code for a more favourable outcome? What amendments have you made to the code?

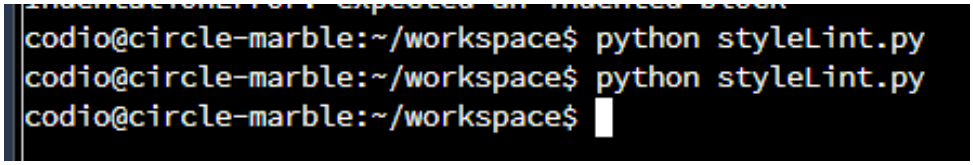
Yes, it is possible to correct this code by amending the source code and ensuring that correct indentation is used. For example, the original code looks like so:

```
# CODE SOURCE: SOFTWARE ARCHITEC

def factorial(n):
""" Return factorial of n """
if n == 0:
return 1
else:
return n*factorial(n-1)
```

And is correctly amended like so:

```
# CODE SOURCE: SOFTWARE ARCHITECTURE WITH

def factorial(n):
  """ Return factorial of n """
  if n == 0:
    return 1
  else:
    return n*factorial(n-1)
```

Now, executing the Python source file compiles successfully:



Question 2:

- `pip install pylint`. Run `pylint` on `pylintTest.py`. Review each of the code errors returned. Can you correct each of the errors identified by pylint? *Before correcting the code errors, save the `pylintTest.py` file with a new name (it will be needed again in the next question).*

```
codio@circle-marble:~/workspace$ pylint pylintTest.py
No config file found, using default configuration
************* Module pylintTest
C:  5, 0: Trailing whitespace (trailing-whitespace)
W: 12, 0: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
W: 13, 0: Bad indentation. Found 4 spaces, expected 8 (bad-indentation)
W: 14, 0: Bad indentation. Found 6 spaces, expected 12 (bad-indentation)
W: 15, 0: Bad indentation. Found 4 spaces, expected 8 (bad-indentation)
W: 16, 0: Bad indentation. Found 6 spaces, expected 12 (bad-indentation)
W: 17, 0: Bad indentation. Found 6 spaces, expected 12 (bad-indentation)
C: 17, 0: Exactly one space required around assignment
      encoded=encoded + letters[x]
              ^ (bad-whitespace)
W: 18, 0: Bad indentation. Found 4 spaces, expected 8 (bad-indentation)
W: 19, 0: Bad indentation. Found 6 spaces, expected 12 (bad-indentation)
W: 20, 0: Bad indentation. Found 8 spaces, expected 16 (bad-indentation)
W: 21, 0: Bad indentation. Found 12 spaces, expected 20 (bad-indentation)
W: 22, 0: Bad indentation. Found 8 spaces, expected 16 (bad-indentation)
W: 23, 0: Bad indentation. Found 10 spaces, expected 20 (bad-indentation)
W: 24, 0: Bad indentation. Found 10 spaces, expected 20 (bad-indentation)
C: 26, 0: Final newline missing (missing-final-newline)
C:  1, 0: Module name "pylintTest" doesn't conform to snake_case naming style (invalid-name)
C:  1, 0: Missing module docstring (missing-docstring)
C:  6, 0: Constant name "shift" doesn't conform to UPPER_CASE naming style (invalid-name)
C:  7, 0: Constant name "choice" doesn't conform to UPPER_CASE naming style (invalid-name)
C:  8, 0: Constant name "word" doesn't conform to UPPER_CASE naming style (invalid-name)
C:  9, 0: Constant name "letters" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 10, 0: Constant name "encoded" doesn't conform to UPPER_CASE naming style (invalid-name)
W: 19, 6: Redefining name 'letter' from outer scope (line 12) (redefined-outer-name)

--------------------------------------
Your code has been rated at -2.63/10
```

A number of errors refer to “bad indentation”. So, the first step go through the Python source file line by line and ensure indentation is consistent. Also, paying attention to each linter message referring to case conventions and also if-then-else logic.

After fiddling around in Codio and not making any obvious headway (always issues with “indentation”), I decided to copy-paste the code into Visual Studio Code and perform “Format Document”. I then copied the code back into Codio and ran pylint on the corrected version and eventually arrived at “clean” code

```
codio@circle-marble:~/workspace$ pylint pylint_test2.py
No config file found, using default configuration

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.95/10, +1.05)

codio@circle-marble:~/workspace$
```

Question 3:

- `pip install flake8`. Run `flake8` on `pylintTest.py`. Review the errors returned.

```
codio@circle-marble:~/workspace$ flake8 pylintTest.py
pylintTest.py:5:1: W293 blank line contains whitespace
pylintTest.py:12:3: E111 indentation is not a multiple of 4
pylintTest.py:14:7: E111 indentation is not a multiple of 4
pylintTest.py:16:7: E111 indentation is not a multiple of 4
pylintTest.py:17:7: E111 indentation is not a multiple of 4
pylintTest.py:17:14: E225 missing whitespace around operator
pylintTest.py:19:7: E111 indentation is not a multiple of 4
pylintTest.py:23:11: E111 indentation is not a multiple of 4
pylintTest.py:24:11: E111 indentation is not a multiple of 4
pylintTest.py:26:14: W292 no newline at end of file
codio@circle-marble:~/workspace$
```

My first impression on review of the errors is “What is going on?” There was a bit of cognitive lapse trying to decipher the “E111” and “W293” error messages because the *what* of the error messages seems to be lost by the presentation of an error code. However, I expected pretty much the same concerns based on the previous run, namely indentation issues.

- In what way does this error message differ form the error message returned by pylint?

  Flake8 definitely has *fewer* errors than pylint, however my preference is that pylint presents the errors in a better manner. That is, they make clear, *what* the problem is without focusing on some abstract code which is irrelevant to a developer checking their code. With flake8, what would be better presentation is for the errors to be *grouped* by a common cause.

- Run `flake8` on `metricTest.py`. Can you correct each of the errors returned by `flake8`? What amendments have you made to the code?

```
codio@circle-marble:~/workspace$ flake8 metricTest.py
metricTest.py:2:1: E265 block comment should start with '# '
metricTest.py:2:48: W291 trailing whitespace
metricTest.py:13:8: E999 SyntaxError: invalid syntax
metricTest.py:16:1: E112 expected an indented block
metricTest.py:20:1: E128 continuation line under-indented for visual indent
metricTest.py:21:1: E128 continuation line under-indented for visual indent
metricTest.py:22:1: E128 continuation line under-indented for visual indent
metricTest.py:23:1: E112 expected an indented block
metricTest.py:27:8: E225 missing whitespace around operator
metricTest.py:28:1: E112 expected an indented block
metricTest.py:30:3: E261 at least two spaces before inline comment
metricTest.py:31:8: E225 missing whitespace around operator
metricTest.py:31:17: E225 missing whitespace around operator
metricTest.py:32:1: E112 expected an indented block
metricTest.py:34:3: E261 at least two spaces before inline comment
metricTest.py:34:80: E501 line too long (83 > 79 characters)
metricTest.py:35:2: E201 whitespace after '['
metricTest.py:35:5: E202 whitespace before ']'
metricTest.py:36:8: E225 missing whitespace around operator
metricTest.py:37:1: E112 expected an indented block
metricTest.py:37:8: E225 missing whitespace around operator
metricTest.py:38:1: E112 expected an indented block
metricTest.py:38:3: E261 at least two spaces before inline comment
metricTest.py:39:22: E231 missing whitespace after ','
metricTest.py:40:10: E225 missing whitespace around operator
metricTest.py:41:1: E112 expected an indented block
metricTest.py:41:3: E261 at least two spaces before inline comment
metricTest.py:42:35: E231 missing whitespace after ','
metricTest.py:42:45: E231 missing whitespace after ','
metricTest.py:43:1: E128 continuation line under-indented for visual indent
```

Yes it is possible to correct the errors. I would perform these steps:

1. Remove the redundant line numbers from each source code line
2. I will then ensure all whitespace is remove from the start of each line
3. I then move top-to-bottom to ensure all if-then-else blocks are correctly indented
4. Then ensure all split lines of code are merged onto a single line

```
# Might as well go by normal bus
return random.choice(('bus:330','bus:331',':'.join((favorite_option, favorite_route))))
elif d>90:
```

5. Continuously perform tabbed indentation for each line as I move from top to bottom.
6. I then run the formatted code through flake8 once more and observe the returned errors:

```
codio@circle-marble:~/workspace$ flake8 metricTest2.py
metricTest2.py:2:1: E265 block comment should start with '# '
metricTest2.py:2:48: W291 trailing whitespace
metricTest2.py:9:1: W293 blank line contains whitespace
metricTest2.py:12:1: E302 expected 2 blank lines, found 1
metricTest2.py:13:2: E111 indentation is not a multiple of 4
metricTest2.py:14:2: E111 indentation is not a multiple of 4
metricTest2.py:16:1: E302 expected 2 blank lines, found 1
metricTest2.py:16:72: E231 missing whitespace after ','
metricTest2.py:16:80: E501 line too long (118 > 79 characters)
metricTest2.py:16:95: E231 missing whitespace after ','
metricTest2.py:17:24: E999 SyntaxError: invalid syntax
metricTest2.py:19:9: E225 missing whitespace around operator
metricTest2.py:23:9: E225 missing whitespace around operator
metricTest2.py:23:18: E225 missing whitespace around operator
metricTest2.py:26:80: E501 line too long (84 > 79 characters)
metricTest2.py:27:9: E225 missing whitespace around operator
metricTest2.py:28:13: E225 missing whitespace around operator
metricTest2.py:30:31: E231 missing whitespace after ','
metricTest2.py:31:15: E225 missing whitespace around operator
metricTest2.py:33:44: E231 missing whitespace after ','
metricTest2.py:33:54: E231 missing whitespace after ','
metricTest2.py:33:80: E501 line too long (99 > 79 characters)
metricTest2.py:34:15: E225 missing whitespace around operator
metricTest2.py:42:25: E231 missing whitespace after ','
metricTest2.py:51:18: E225 missing whitespace around operator
metricTest2.py:53:15: E225 missing whitespace around operator
metricTest2.py:54:21: E225 missing whitespace around operator
metricTest2.py:56:1: E302 expected 2 blank lines, found 1
metricTest2.py:62:18: E231 missing whitespace after ','
metricTest2.py:63:13: E225 missing whitespace around operator
metricTest2.py:69:18: E225 missing whitespace around operator
metricTest2.py:72:28: W292 no newline at end of file
codio@circle-marble:~/workspace$
```

7. I address each error in turn and eventually, arrive at the following formatted Python code:

```python
# CODE SOURCE: SOFTWARE ARCHITECTURE WITH PYTHON

"""
Module metricTest.py
Metric example - Module which is used as a testbed for static checkers.
This is a mix of different functions and classes doing different things.
"""

import random


def fn(x, y):
    """ A function which performs a sum """
    return x + y


def find_optimal_route_to_my_office_from_home(start_time, expected_time,
                                              favorite_route='SBS1K',
                                              favorite_option='bus'):
    d = (expected_time-start_time).total_seconds() / 60.0

    if d <= 30:
        return 'car'
```

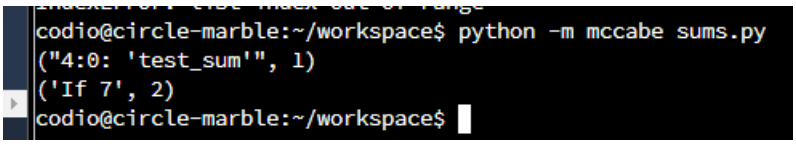
```
    # If d>30 but <45, first drive then take metro
    if d > 30 and d < 45:
        return ('car', 'metro')

    # If d>45 there are a combination of optionsWriting
    # Modifiable and Readable Code
    if d > 45:
        if d < 60:
            # First volvo,then connecting bus
            return ('bus:335E', 'bus:connector')
        elif d > 80:
            # Might as well go by normal bus
            return random.choice(('bus:330', 'bus:331', ':'
                                  .join((favorite_option, favorite_route))))
        elif d > 90:
            # Relax and choose favorite route
            return ':'.join((favorite_option, favorite_route))


class C(object):
    """ A class which does almost nothing """

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def f(self):
        pass

    def g(self, x, y):

        if self.x > x:
            return self.x + self.y
        elif x > self.x:
            return x + self.y


class D(C):
    """ D class """

    def __init__(self, x):
        self.x = x

    def f(self, x, y):
        if x > y:
            return x - y
        else:
            return x + y

    def g(self, y):
        if self.x > y:
            return self.x + y
        else:
            return y-self.x
```

Observation: despite the linter complaining about Python code requiring a space between operators, I am unsure why this code `(expected_time-start_time)` does not require a space between the two variables.

Question 4:

7

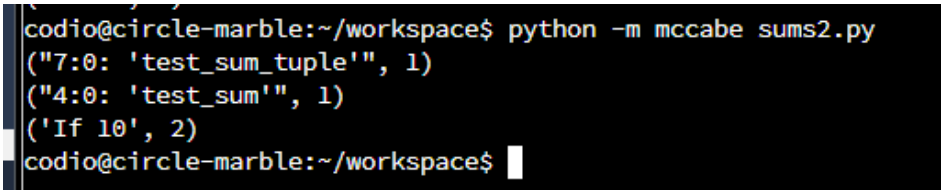
- `pip install mccabe`. Run mccabe on `sums.py`. What is the result?



I did not understand the meaning of the output and briefly researched what the output is meant to represent (Hamilton, 2021). I understand the last value is detailing the complexity of the element found in parameter 1, according to the McCabe-Thiele algorithm.

This tells me that the function “test_sum” has a complexity of 1, while the “if” statement (on line “7”) has a complexity of 2—it calls a function (“test_sum”) and then calls the “print” function.

- Run `mccabe` on `sum2.py`. What is the result? What are the contributors to the cyclomatic complexity in each piece of code?



The complexity levels are specified as

- “test_sum_tuple” has a complexity level of “1”. It makes a single function call.
- “test_sum” on line “4” has a complexity of “1”. It makes a single function call.
- The “if” statement on line “10” has a complexity level of 2, because according to the formula

$$V(G) = Edge\_Count – Node\_Count + 2$$

Thus, *Edge_Count (control flow) = 2 MINUS Node_Count (tasks) = 2* PLUS 2 = 2.

Activity

Select one or more of the tools installed above and use it/them to test the code your team has created as part of the summative assessment. You should demonstrate your tests (and share your results) during the Seminar. Discuss the need to change the code based on the output form the test tools/linters.

This was not possible to complete at the point of the seminar as the team was actively engaged in the technical design report and not focused on coding the solution.

References

Hamilton, T. (2021) Mccabe's Cyclomatic Complexity: Calculate with Flow Graph (Example) Available from https://www.guru99.com/cyclomatic-complexity.html [Accessed 22 Sep. 2021]

Bibliography

Stadlbauer, R. (2018) What exactly is Mccabe Cyclomatic Complextiy? Available from https://dzone.com/articles/what-exactly-is-mccabe-cyclomatic-complexity [Accessed 22 Sep. 2021]