Unit 6 Seminar

Title: Exploring Linters to Support Testing in Python

The following questions will be discussed during this week's seminar. These questions are provided in the Codio workspace – Testing with Python – where the activities should be completed.

Question 1

1.1 Run styleLint.py in Codio.

```
codio@elitewonder-memphisswitch:~/workspace$ python3 styleLint.py
  File "styleLint.py", line 4
    """ Return factorial of n """
                              ^
IndentationError: expected an indented block
```

1.2 What happens when the code is run? Can you modify this code for a more favourable outcome? What amendments have you made to the code?

When I run the code, the output returns an indentation error: "IndentationError: 'expected an indented block'". Below are the amendments I have made to fix the error:

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

Question 2

pip install Pylint Run *pylint* on pylintTest.py

Output after running the code:

```
codio@elitewonder-memphisswitch:~/workspace$ pylint pylintTest.py
************* Module pylintTest
pylintTest.py:10:3: E0001: invalid syntax (<unknown>, line 10) (syntax-error)
```

2.2 Review each of the code errors returned. Can you correct each of the errors identified by pylint?

Here is my code after correcting the errors:

```python
'''import module'''

import string

shift = 3
choice = input("would you like to encode or decode?")
word = input("Please enter text")
letters = string.ascii_letters + string.punctuation + string.digits
encoded = ''

if choice == "encode":
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
    else:
        x = letters.index(letter) + shift
        encoded = encoded + letters[x]
    if choice == "decode":
        for letter in word:
            if letter == ' ':
                encoded = encoded + ' '
        else:
            x = letters.index(letter) - shift
            encoded = encoded + letters[x]

print(encoded)
```

Output after correcting the indentation error:

```
codio@elitewonder-memphisswitch:~/workspace$ pylint pylint_test.py
************* Module pylint_test
pylint_test.py:7:0: C0103: Constant name "shift" doesn't conform to UPPER_CASE naming style
(invalid-name)
pylint_test.py:10:0: C0103: Constant name "letters" doesn't conform to UPPER_CASE naming style
(invalid-name)
pylint_test.py:11:0: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style
(invalid-name)
pylint_test.py:17:4: W0120: Else clause on loop without a break statement, remove the else and de-
indent all the code inside it (useless-else-on-loop)
pylint_test.py:16:12: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming
style (invalid-name)
pylint_test.py:18:8: C0103: Constant name "x" doesn't conform to UPPER_CASE naming style
(invalid-name)
pylint_test.py:19:8: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style
(invalid-name)
```

```
pylint_test.py:24:8: W0120: Else clause on loop without a break statement, remove the else and de-
indent all the code inside it (useless-else-on-loop)
pylint_test.py:23:16: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming
style (invalid-name)
pylint_test.py:25:12: C0103: Constant name "x" doesn't conform to UPPER_CASE naming style
(invalid-name)
pylint_test.py:26:12: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming
style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 4.21/10 (previous run: 5.26/10, -1.05)
```

2.3 Before correcting the code errors, save the pylintTest.py file with a new name (it will be needed again in the next question).

I have taken corrective action on the invalid-name messages and ensured compliance with the PEP 8 standard. Pylint uses a regular expression to verify naming conventions, and I applied the rule "should match [a-z_][a-z0-9_]{2,30}$" to convert all variables to lowercase. By enabling the --const-rgx='[a-z\_][a-z0-9\_]{2,30}$' option, I was able to eliminate errors based on the ideas of pylint.pycqa.org. (N.D.), and here is the new output:

```
codio@elitewonder-memphisswitch:~/workspace$ pylint pylint_test.py --const-rgx='[a-z\_][a-z0-
9\_]{2,30}$'
No config file found, using default configuration
************* Module pylint_test
W: 18, 4: Else clause on loop without a break statement (useless-else-on-loop)
W: 25, 8: Else clause on loop without a break statement (useless-else-on-loop)

------------------------------------------------------------------
Your code has been rated at 8.89/10 (previous run: 6.11/10, +2.78)
```

I have changed my code as shown below:

```
'''import module'''

import string

shift = 3
choice = input("Would you like to encode or decode? ")  # Use input() in Python 3
word = input("Please enter text ")  # Use input() in Python 3
letters = string.ascii_letters + string.punctuation + string.digits
encoded = ''

if choice == "encode":
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
        else:
            x = (letters.index(letter) + shift) % len(letters)
            encoded = encoded + letters[x]
elif choice == "decode":  # Corrected indentation and added elif
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
        else:
            x = (letters.index(letter) - shift) % len(letters)
            encoded = encoded + letters[x]
```

and the output is as follows:

```
codio@elitewonder-memphisswitch:~/workspace$ pylint pylint_test.py --const-rgx='[a-z\_][a-z0-9\_]{2,30}$'No config file found, using default configuration

-----------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.89/10, +1.11)
```

Question 3

pip install flake8 Run *flake8* on pylintTest.py

Output:

```
codio@elitewonder-memphisswitch:~/workspace$ flake8 pylintTest.py
pylintTest.py:3:14: W291 trailing whitespace
pylintTest.py:4:10: W291 trailing whitespace
pylintTest.py:5:58: W291 trailing whitespace
pylintTest.py:6:40: W291 trailing whitespace
pylintTest.py:7:68: W291 trailing whitespace
pylintTest.py:8:13: W291 trailing whitespace
pylintTest.py:9:23: W291 trailing whitespace
pylintTest.py:10:1: E112 expected an indented block
pylintTest.py:10:3: E999 IndentationError: expected an indented block
pylintTest.py:11:1: E112 expected an indented block
pylintTest.py:11:17: E701 multiple statements on one line (colon)
pylintTest.py:11:42: W291 trailing whitespace
pylintTest.py:12:6: W291 trailing whitespace
pylintTest.py:13:1: E112 expected an indented block
pylintTest.py:13:34: W291 trailing whitespace
pylintTest.py:14:8: E225 missing whitespace around operator
pylintTest.py:14:29: W291 trailing whitespace
pylintTest.py:16:1: E112 expected an indented block
pylintTest.py:17:2: E111 indentation is not a multiple of 4
pylintTest.py:17:19: W291 trailing whitespace
pylintTest.py:18:1: E112 expected an indented block
pylintTest.py:18:24: W291 trailing whitespace
pylintTest.py:20:2: E111 indentation is not a multiple of 4
pylintTest.py:20:35: W291 trailing whitespace
pylintTest.py:21:31: W291 trailing whitespace
pylintTest.py:22:14: W291 trailing whitespace
```

3.2 Review the errors returned. In what way does this error message differ from the error message returned by pylint?

The output from Flake8 and Pylint differs. Flake8 detects whitespace and blank line issues, while Pylint flags naming convention violations. Both linters detect indentation errors.

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

Output:

```
codio@elitewonder-memphisswitch:~/workspace$ flake8 metricTest.py
metricTest.py:1:1: E265 block comment should start with '# '
metricTest.py:1:48: W291 trailing whitespace
metricTest.py:12:8: E999 SyntaxError: invalid syntax
metricTest.py:15:1: E112 expected an indented block
metricTest.py:19:1: E128 continuation line under-indented for visual indent
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: E112 expected an indented block
metricTest.py:26:8: E225 missing whitespace around operator
metricTest.py:27:1: E112 expected an indented block
metricTest.py:29:3: E261 at least two spaces before inline comment
metricTest.py:30:8: E225 missing whitespace around operator
metricTest.py:30:17: E225 missing whitespace around operator
metricTest.py:31:1: E112 expected an indented block
metricTest.py:33:3: E261 at least two spaces before inline comment
metricTest.py:33:80: E501 line too long (83 > 79 characters)
metricTest.py:34:2: E201 whitespace after '['
metricTest.py:34:5: E202 whitespace before ']'
metricTest.py:35:8: E225 missing whitespace around operator
metricTest.py:36:1: E112 expected an indented block
metricTest.py:36:8: E225 missing whitespace around operator
metricTest.py:37:1: E112 expected an indented block
metricTest.py:37:3: E261 at least two spaces before inline comment
metricTest.py:38:22: E231 missing whitespace after ','
metricTest.py:39:10: E225 missing whitespace around operator
metricTest.py:40:1: E112 expected an indented block
metricTest.py:40:3: E261 at least two spaces before inline comment
metricTest.py:41:35: E231 missing whitespace after ','
metricTest.py:41:45: E231 missing whitespace after ','
metricTest.py:42:1: E128 continuation line under-indented for visual indent
metricTest.py:43:1: E128 continuation line under-indented for visual indent
metricTest.py:44:10: E225 missing whitespace around operator
metricTest.py:45:1: E112 expected an indented block
metricTest.py:45:3: E261 at least two spaces before inline comment
metricTest.py:47:1: E128 continuation line under-indented for visual indent
metricTest.py:51:1: E112 expected an indented block
metricTest.py:53:24: E231 missing whitespace after ','
metricTest.py:54:1: E112 expected an indented block
metricTest.py:58:1: E112 expected an indented block
metricTest.py:61:1: E112 expected an indented block
metricTest.py:62:13: E225 missing whitespace around operator
metricTest.py:63:1: E112 expected an indented block
metricTest.py:64:10: E225 missing whitespace around operator
metricTest.py:65:1: E112 expected an indented block
metricTest.py:65:12: E225 missing whitespace around operator
metricTest.py:68:1: E112 expected an indented block
metricTest.py:71:1: E112 expected an indented block
metricTest.py:73:17: E231 missing whitespace after ','
metricTest.py:74:1: E112 expected an indented block
metricTest.py:74:8: E225 missing whitespace around operator
metricTest.py:75:1: E112 expected an indented block
metricTest.py:76:8: E231 missing whitespace after ':'
metricTest.py:77:2: E201 whitespace after '['
metricTest.py:77:5: E202 whitespace before ']'
metricTest.py:81:1: E112 expected an indented block
metricTest.py:82:13: E225 missing whitespace around operator
metricTest.py:83:1: E112 expected an indented block
metricTest.py:85:1: E112 expected an indented block
metricTest.py:85:19: W292 no newline at end of file
```

My amendments to correct the errors:

```
"""
 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 30 < d < 45:  # Amended the logical condition
        return ('car', 'metro')

    # If d>45 there are a combination of optionsWriting
    # Modifiable and Readable Code[68]
    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
```

Output:

```
codio@elitewonder-memphisswitch:~/workspace$ flake8 metric_test.py
codio@elitewonder-memphisswitch:~/workspace$
```

Question 4

pip install mccabe

4.1 Run mccabe on sums.py. What is the result?

```
codio@elitewonder-memphisswitch:~/workspace$ python -m mccabe sums.py
("3:0: 'test_sum'", 1)
('If 6', 2)
```

4.2 Run mccabe on sums2.py. What is the result?

```
codio@elitewonder-memphisswitch:~/workspace$ python -m mccabe sums2.py
('If 9', 2)
("6:0: 'test_sum_tuple'", 1)
("3:0: 'test_sum'", 1)
```

4.3 What are the contributors to the cyclomatic complexity in each piece of code?

Thomas McCabe created a metric called cyclomatic complexity to assess the reliability and stability of a programme. It counts how many linearly independent routes there are through a program module. Less sophisticated cyclomatic programmes are safer to adapt and more accessible to comprehend (IBM, 2018). When using the *mccabe* tool, each code calculates cyclomatic complexity by analysing the code's control flow.

The cyclomatic complexity of a codebase is established based on the quantity of control flow statements present within it. If a codebase lacks any control flow statements, its complexity is deemed to be 1. However, if there is a single if condition, the complexity rises to 2. When it comes to structured programming, the directed graph within the control flow represents the edge linking two basic blocks of the program (GeeksforGeeks, 2018).

Hence, for each piece of code, the contributors to the cyclomatic complexity are: the *sums.py* has a cyclomatic complexity of 1 due to the lack of decision points beyond the function definition. On the other hand, *sums2.py* has a cyclomatic complexity of 3 due to branching caused by two functions and an if statement.

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 from the test tools/linters.

Unfortunately, I couldn’t test my colleague's code since they were unavailable and didn’t provide me with their code.

References:

pylint.pycqa.org. (N.D.). *Tutorial — Pylint 2.5.4 documentation*. [online] Available at: https://pylint.pycqa.org/en/2.5/tutorial.html.

IBM (2018). *Cyclomatic complexity*. [online] Available at: https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity.

GeeksforGeeks (2018). *Cyclomatic Complexity - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/cyclomatic-complexity/.