Linters for Python Code

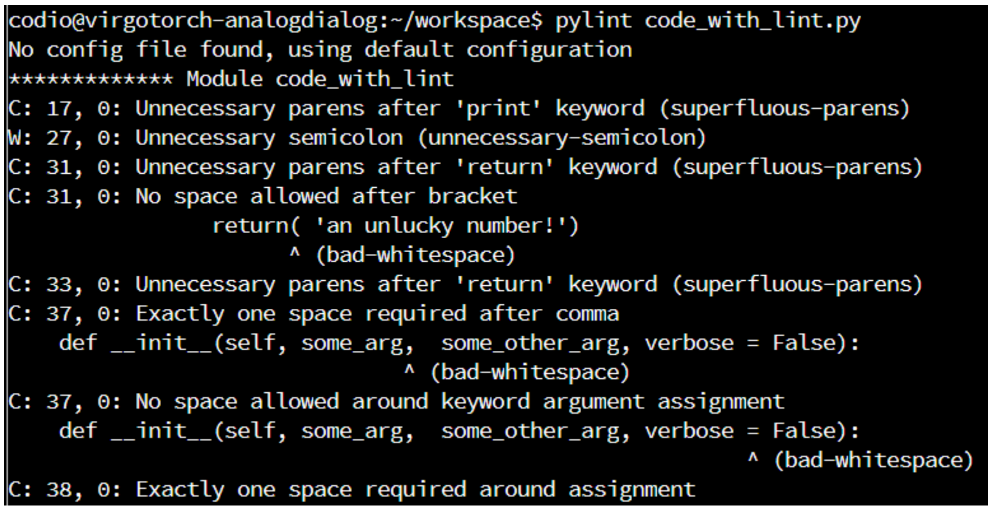
For this activity, you will use Linters to check the quality of your Python code. Remember to save your work to your GitHub Repository.

My work

Using Linters to Achieve Python Code Quality

pylint code_with_lint.py

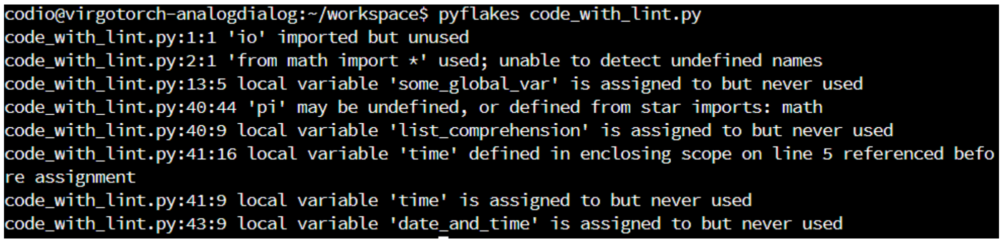
This tool highlights the style and formatting of the code, including the spacing and the letter case.



```
codio@virgotorch-analogdialog:~/workspace$ pylint code_with_lint.py
No config file found, using default configuration
************* Module code_with_lint
C: 17, 0: Unnecessary parens after 'print' keyword (superfluous-parens)
W: 27, 0: Unnecessary semicolon (unnecessary-semicolon)
C: 31, 0: Unnecessary parens after 'return' keyword (superfluous-parens)
C: 31, 0: No space allowed after bracket
                return( 'an unlucky number!')
                      ^ (bad-whitespace)
C: 33, 0: Unnecessary parens after 'return' keyword (superfluous-parens)
C: 37, 0: Exactly one space required after comma
    def __init__(self, some_arg,  some_other_arg, verbose = False):
                               ^ (bad-whitespace)
C: 37, 0: No space allowed around keyword argument assignment
    def __init__(self, some_arg,  some_other_arg, verbose = False):
                                                          ^ (bad-whitespace)
C: 38, 0: Exactly one space required around assignment
```

pyflakes code_with_lint.py

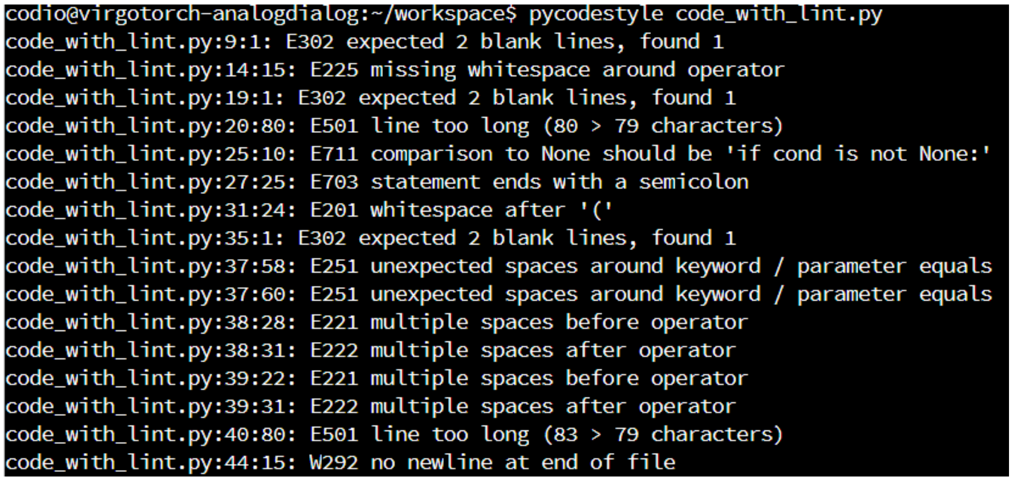
In this approach, the tool focus on the variable and the module to highlight if they are being used.



```
codio@virgotorch-analogdialog:~/workspace$ pyflakes code_with_lint.py
code_with_lint.py:1:1 'io' imported but unused
code_with_lint.py:2:1 'from math import *' used; unable to detect undefined names
code_with_lint.py:13:5 local variable 'some_global_var' is assigned to but never used
code_with_lint.py:40:44 'pi' may be undefined, or defined from star imports: math
code_with_lint.py:40:9 local variable 'list_comprehension' is assigned to but never used
code_with_lint.py:41:16 local variable 'time' defined in enclosing scope on line 5 referenced befo
re assignment
code_with_lint.py:41:9 local variable 'time' is assigned to but never used
code_with_lint.py:43:9 local variable 'date_and_time' is assigned to but never used
```

pycodestyle code_with_lint.py

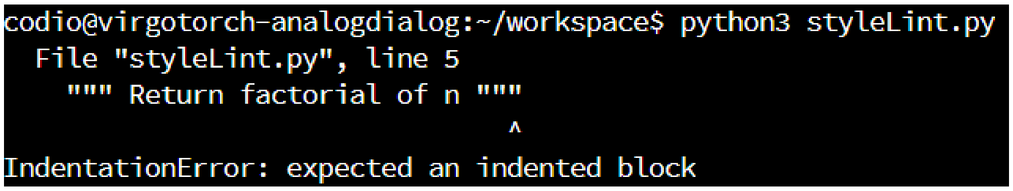
The tool is useful to find the formatting issue such as missing space or extra space.



```
codio@virgotorch-analogdialog:~/workspace$ pycodestyle code_with_lint.py
code_with_lint.py:9:1: E302 expected 2 blank lines, found 1
code_with_lint.py:14:15: E225 missing whitespace around operator
code_with_lint.py:19:1: E302 expected 2 blank lines, found 1
code_with_lint.py:20:80: E501 line too long (80 > 79 characters)
code_with_lint.py:25:10: E711 comparison to None should be 'if cond is not None:'
code_with_lint.py:27:25: E703 statement ends with a semicolon
code_with_lint.py:31:24: E201 whitespace after '('
code_with_lint.py:35:1: E302 expected 2 blank lines, found 1
code_with_lint.py:37:58: E251 unexpected spaces around keyword / parameter equals
code_with_lint.py:37:60: E251 unexpected spaces around keyword / parameter equals
code_with_lint.py:38:28: E221 multiple spaces before operator
code_with_lint.py:38:31: E222 multiple spaces after operator
code_with_lint.py:39:22: E221 multiple spaces before operator
code_with_lint.py:39:31: E222 multiple spaces after operator
code_with_lint.py:40:80: E501 line too long (83 > 79 characters)
code_with_lint.py:44:15: W292 no newline at end of file
```

Testing in Python: Question 1

What happens when the code is run?



```
codio@virgotorch-analogdialog:~/workspace$ python3 styleLint.py
  File "styleLint.py", line 5
    """ Return factorial of n """
                                ^
IndentationError: expected an indented block
```

Can you modify this code for a more favorable outcome?

Yes

What amendments have you made to the code?

Fix the indented issue.

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

Testing in Python: Question 2

Review each of the code errors returned.

Can you correct each of the errors identified by pylint?

Yes.

```
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 LETTERS in WORD:
        if LETTERS == ' ':
            ENCODED = ENCODED + ' '
        else:
            x = LETTERS.index(LETTERS) + SHIFT
            ENCODED = ENCODED + LETTERS[x]
if CHOICE == "decode":
    for LETTERS in WORD:
        if LETTERS == ' ':
            ENCODED = ENCODED + ' '
        else:
            x = LETTERS.index(LETTERS) - SHIFT
            ENCODED = ENCODED + LETTERS[x]
print ENCODED
```

Testing in Python: Question 3

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

The error message does not show if the variable does not follow the naming style

The error message shows the indentation issue, but not in detail (in pylint, it shows the expected indentation number)

Can you correct each of the errors returned by flake8?

Yes.

What amendments have you made to the code?

```
# 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.
"""
from random 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'):
```

```
        dd = (expected_time - start_time).total_seconds()
        d = dd/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[ 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
```

Testing in Python: Question 4

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

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

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

I could not install the mccabe as the python version on Codio is outdated and I am not able to upgrade it

Testing in Python: Question 5

<u>Run mccabe on sums.py. What is the result?</u>

<u>From Section 5 of the Firdaus et al (2014) reading, select a test technique from the following categories:</u>

I would select the Experience-based techniques. The Experience-based techniques focus on the resource based on the past experience.

From the company I am working in, we have an online library section where we can store all the coding file of our best practice. And all the best practices of the coding are updated from all the historical cases where the specialist came together and agreed on the solution.