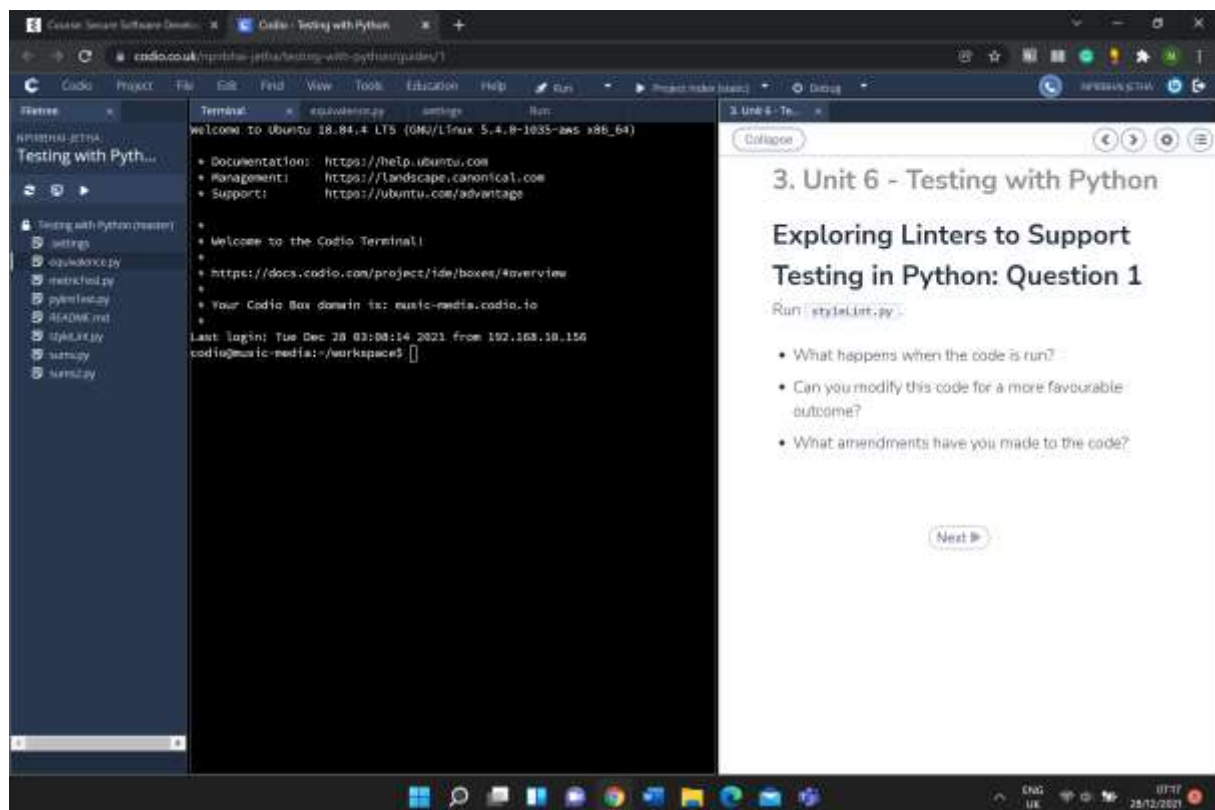
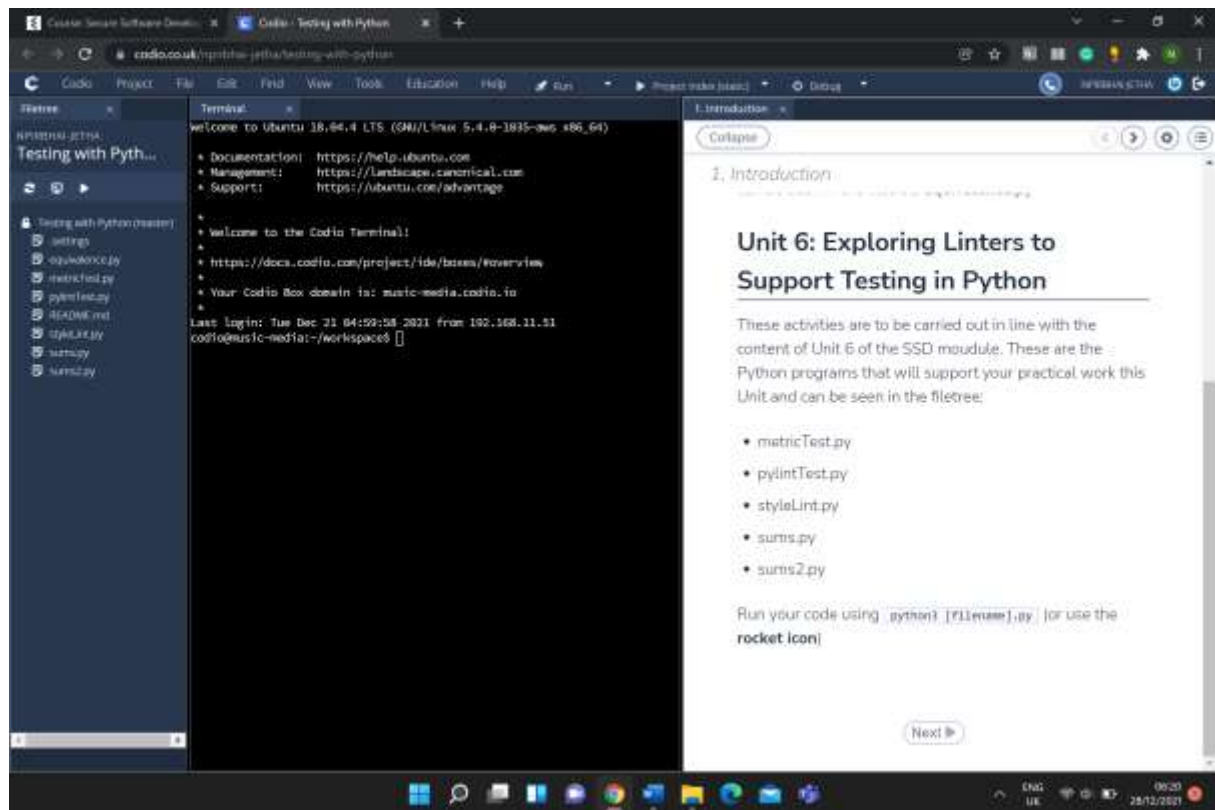
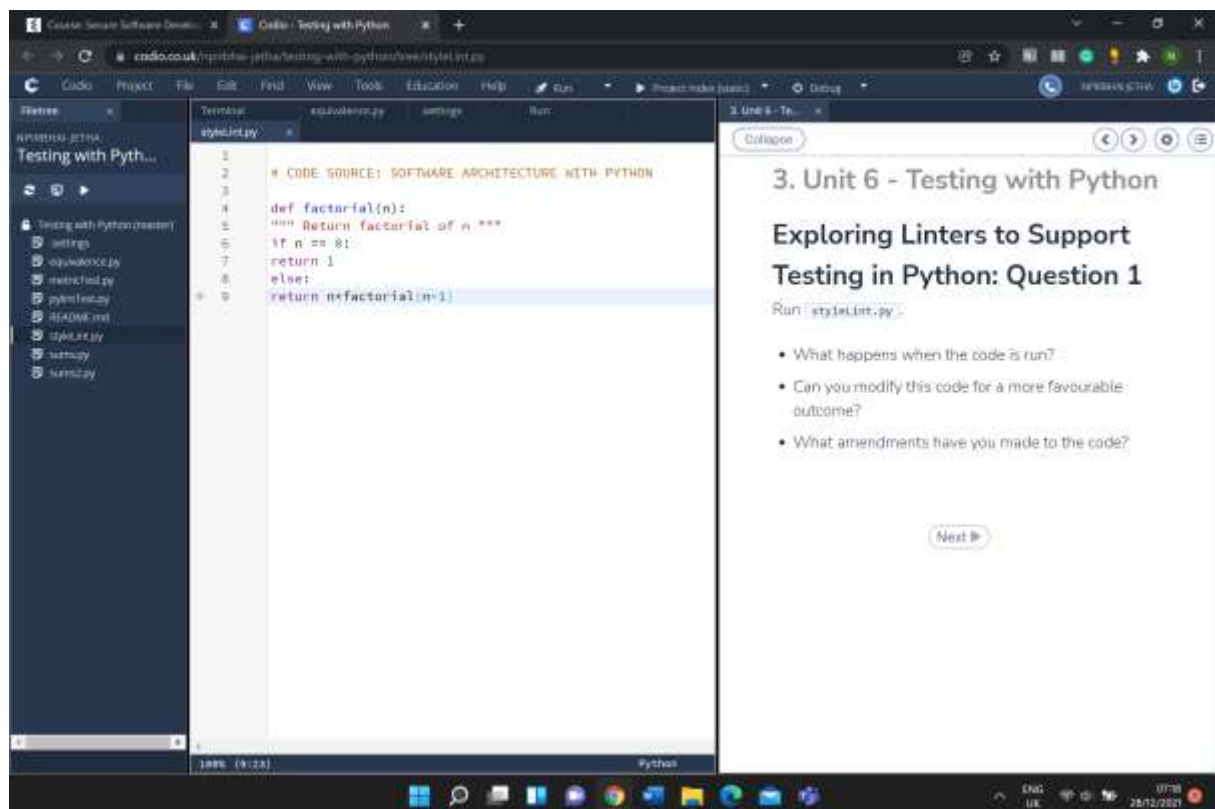


UNIT 6: EXPLORING LINTERS TO SUPPORT TESTING IN PYTHON



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



The screenshot shows the Codio IDE interface. On the left, there's a file explorer with a folder named 'Testing with Python' containing several files. The main editor displays a Python file named 'stylelint.py' with the following code:

```
1
2
3
4
5
6
7
8
9
# CODE SOURCE: SOFTWARE ARCHITECTURE WITH PYTHON
def factorial(n):
    """ Return factorial of n """
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
```

On the right, a sidebar shows a lesson titled '3. Unit 6 - Testing with Python' with the subheading 'Exploring Linters to Support Testing in Python: Question 1'. Below the title, it says 'Run: stylelint.py' and lists three questions:

- 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?

A 'Next' button is visible at the bottom of the sidebar.

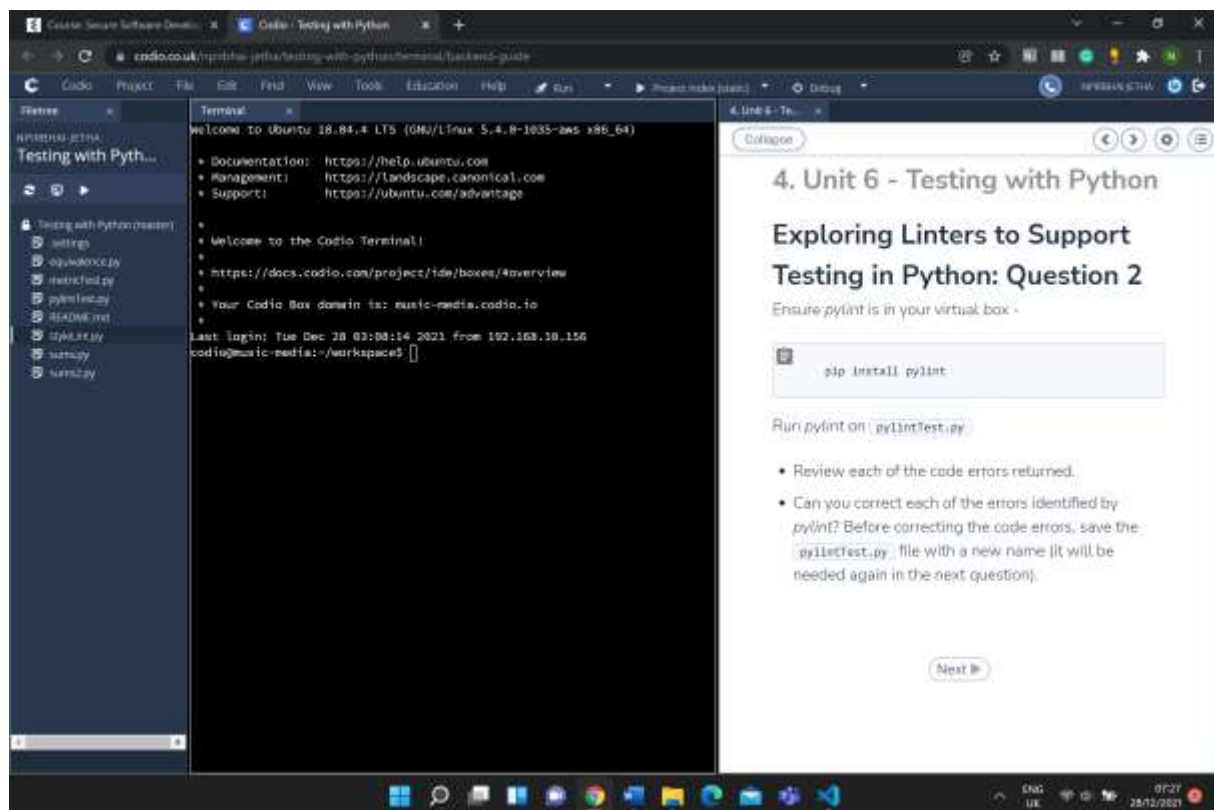
The screenshot shows the same Codio IDE interface, but now the terminal displays an error message:

```
File "stylelint.py", line 5
    """ Return factorial of n """
    ^
IndentationError: expected an indented block
```

A blue box highlights the error message and the corrected code:

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

The right sidebar remains the same, showing the lesson '3. Unit 6 - Testing with Python'.



SOURCE OF CODE: <https://docs.pylint.org/en/1.6.0/tutorial.html>

import string

shift = 3

choice = raw_input("would you like to encode or decode?")

word = (raw_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 == ' ':
```

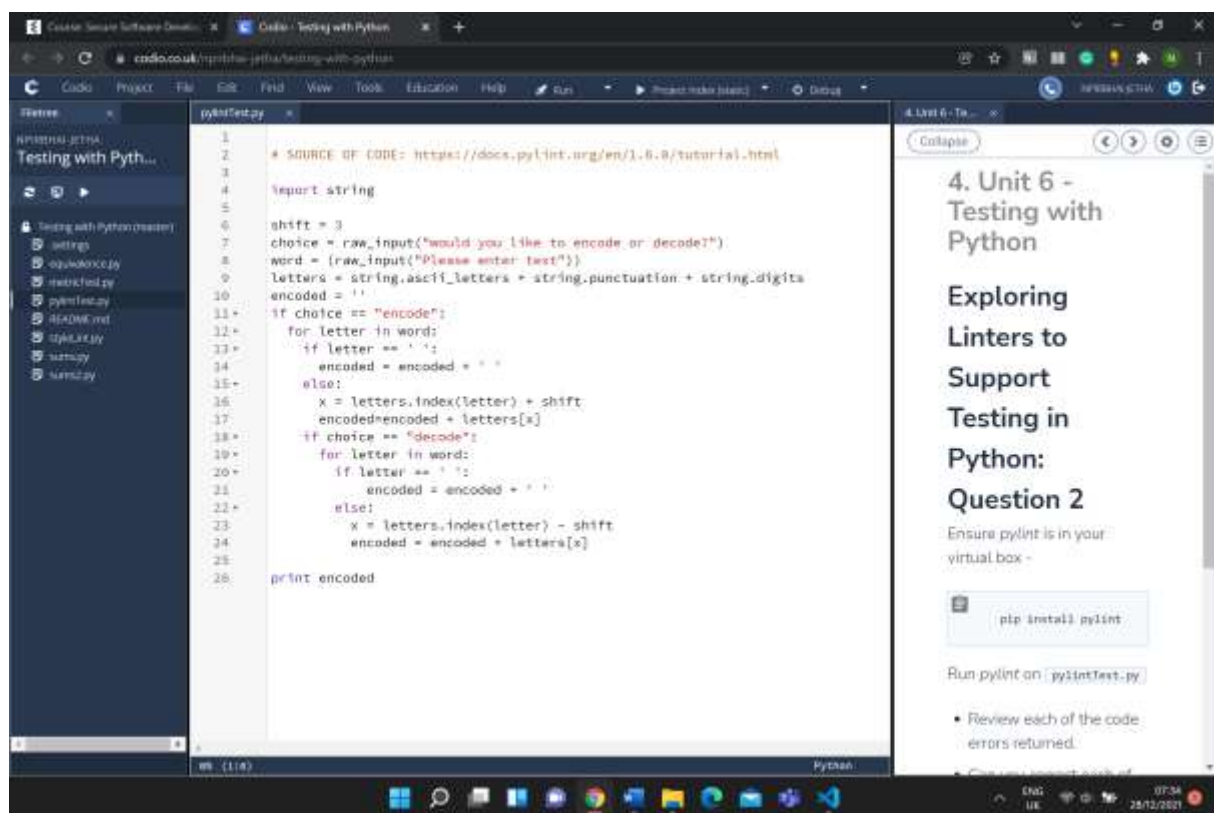
```
            encoded = encoded + ' '
```

```
        else:
```

```
            x = letters.index(letter) - shift
```

```
            encoded = encoded + letters[x]
```

```
print encoded
```



The screenshot shows the Codio IDE interface. On the left, a sidebar lists files in a project named 'Testing with Python...'. The main editor displays the code for `pylint2.py`, which is a Caesar cipher implementation. The code includes comments, variable declarations, and logic for encoding and decoding text based on a user-provided shift and choice. The terminal at the bottom shows the command `python3 pylint2.py` being executed. On the right, a tutorial page titled '4. Unit 6 - Testing with Python' is displayed, featuring the heading 'Exploring Linters to Support Testing in Python: Question 2'. The page instructs the user to ensure `pylint` is installed in their virtual box and provides a code snippet `pip install pylint`. It also lists two tasks: reviewing code errors returned by `pylint` and correcting them by saving the file as `pylintTest.py`.

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
# SOURCE OF CODE: https://docs.pylint.org/en/1.6.9/tutorial.html
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)

```

4. Unit 6 - Testing with Python

Exploring Linters to Support Testing in Python: Question 2

Ensure `pylint` is in your virtual box -

```
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).

This screenshot shows the same Codio IDE interface, but the terminal window is now active. It displays the Ubuntu system boot sequence, including the version (18.04.4 LTS) and kernel (5.4.0-1035-aws x86_64). It also shows the user's login, the current directory (`/workspace`), and the execution of `python3 pylint2.py`. The prompt `would you like to encode or decode?` is visible at the end of the terminal output. The right-hand tutorial page remains the same as in the previous screenshot, providing instructions on installing `pylint` and correcting code errors.

```

welcome to ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

+ Documentation: https://help.ubuntu.com
+ Management:   https://landscape.canonical.com
+ Support:      https://ubuntu.com/advantage

+
+ Welcome to the Codio Terminal!
+
+ https://docs.codio.com/project/ide/boxes/4overview
+
+ Your Codio Box domain is: exsic-media.codio.io
+
Last login: Tue Dec 28 05:29:02 2021 from 192.168.10.156
codingmedia-media:~/workspace$ python3 pylint2.py
would you like to encode or decode?

```

4. Unit 6 - Testing with Python

Exploring Linters to Support Testing in Python: Question 2

Ensure `pylint` is in your virtual box -

```
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).

The screenshot shows a VS Code editor with a Python terminal window and a Jupyter Notebook. The terminal window displays the output of a pip install command for flake8, showing the installation of various dependencies like configparser, flake8, funcparser, and pylint. The Jupyter Notebook on the right shows a slide titled "5. Unit 6 - Testing with Python" and "Exploring Linters to Support Testing in Python: Question 3". The slide includes instructions to ensure flake8 is in the virtual box, a code block to run "pip install flake8", and a list of questions to review errors and correct code.

Terminal Output:

```
python2.py
755e5b7782df118c1f49bdc494dab6e429c93aa77965f33e81287c8c/z
lpp-1.2.0-py2.py3-none-any.whl
Collecting contextlib2; python_version < "3" (from importlib-
metadata; python_version < "3.8")->flake8
  Downloading https://files.pythonhosted.org/packages/85/98/3
78352f7ef6a996c52f0801831622f58f923c1e575427d92180ab311236/c
ontextlib2-0.6.0-py2.py3-none-any.whl
Collecting pathlib2; python_version < "3" (from importlib-met
adata; python_version < "3.8")->flake8
  Downloading https://files.pythonhosted.org/packages/76/67/d
c02c72177ec79f0176e5b9921e9c1745a31ed55aafb3b3ec2b6bba7e/p
athlib2-2.3.0-py2.py3-none-any.whl
Collecting six (from pathlib2; python_version < "3")->importli
b-metadata; python_version < "3.8")->flake8
  Using cached https://files.pythonhosted.org/packages/d9/5a/
w7c11adbe875f2ab6b91bd54cf2dc32d792b5a01386781dbc725c91daf11/
six-1.16.0-py2.py3-none-any.whl
Collecting scandir; python_version < "3.5" (from pathlib2; py
thon_version < "3")->importlib-metadata; python_version < "3.8
")->flake8
  Downloading https://files.pythonhosted.org/packages/d7/f5/9
c854db7d548dbf1bc0b654362bba1912d93c588e950a4f5c5dad4e/s
candir-1.10.0.tar.gz
Building wheels for collected packages: funcparser, scandir
  Running setup.py bdist_wheel for funcparser ... done
  Stored in directory: /home/codio/.cache/pip/wheels/b5/38/32
/77a1898457155686ba6e3c3a8a57132b1a84b1ca7f6517b2
  Running setup.py bdist_wheel for scandir ... done
  Stored in directory: /home/codio/.cache/pip/wheels/91/95/75
/18c98a91230787abbc7c59978ab4b4b48a7ad5b61778335
Successfully built funcparser scandir
Installing collected packages: configparser, flake8, funcpar
sers, contextlib2, zipp, six, scandir, pathlib2, importlib-m
etadata, typing, enum34, mccabe, pycodestyle, flake8
Successfully installed configparser-4.0.2 contextlib2-0.6.0.p
y31 enum34-1.1.10 flake8-3.9.2 funcparser-3.2.3.post2 impor
tlib-metadata-2.1.2 mccabe-0.6.1 pathlib2-2.3.0 pycodestyle-2
.7.0 pyflakes-2.3.1 scandir-1.10.0 six-1.16.0 typing-3.10.0.0
zipp-1.2.0
codio@music-media:~/workspace$
```

Jupyter Notebook Content:

5. Unit 6 - Testing with Python

Exploring Linters to Support Testing in Python: Question 3

Ensure flake8 is in your virtual box -

```
pip install flake8
```

Run flake8 on `pylintTest.py`

- Review the errors returned in what way does this error message differ from the error message returned by pylint?

Run flake8 on `metricTest.py`

- Can you correct each of the errors returned by flake8?
- What amendments have you made to the code?

Next

flake8 pylint2.py

In what way does this error message differ from error message returned by pylint?

More detailed, gives more information about the 'ugly' code lines (such as whitespaces etc.)

- **Pylint(default):** Checks for errors and tries to enforce a coding standard
- **Flake8:** Checks code against style conventions in PEP 8, programming errors and cyclomatic complexity

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
# SOURCE OF CODE: https://docs.pylint.org/en/1.6.0/tutorial.html
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]

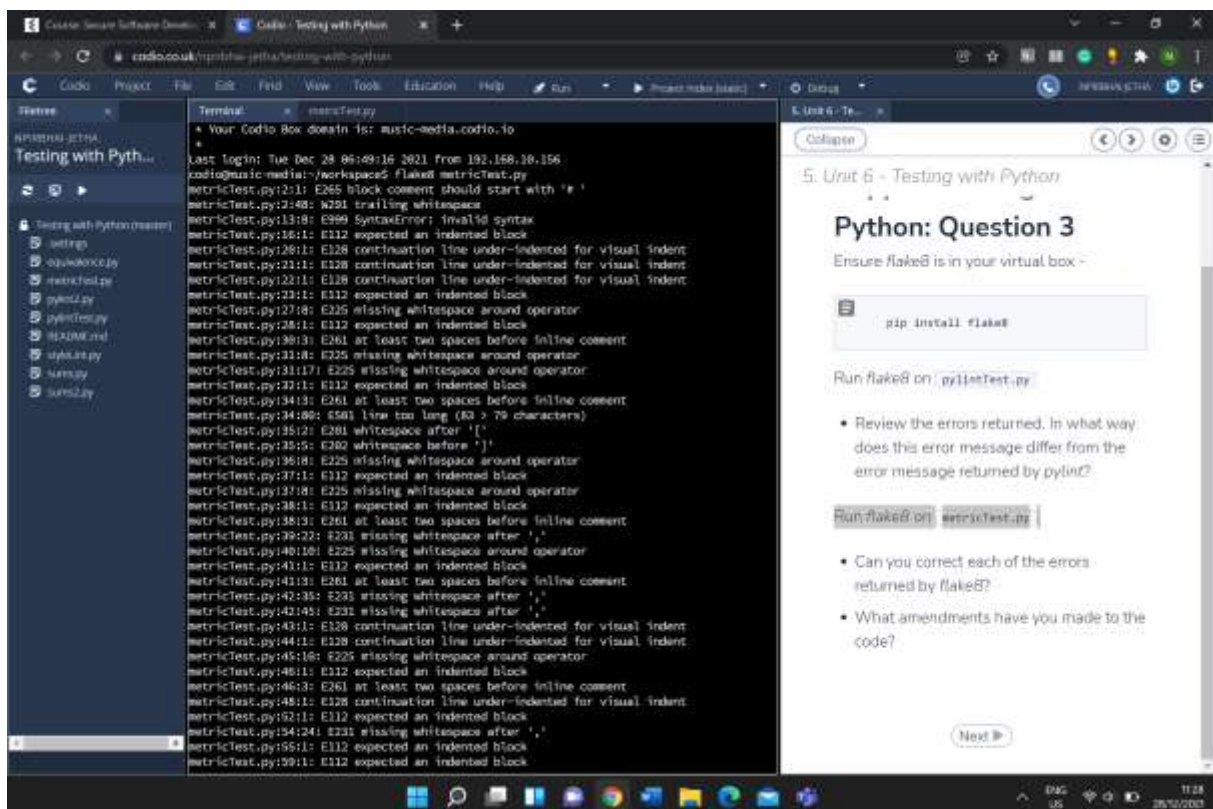
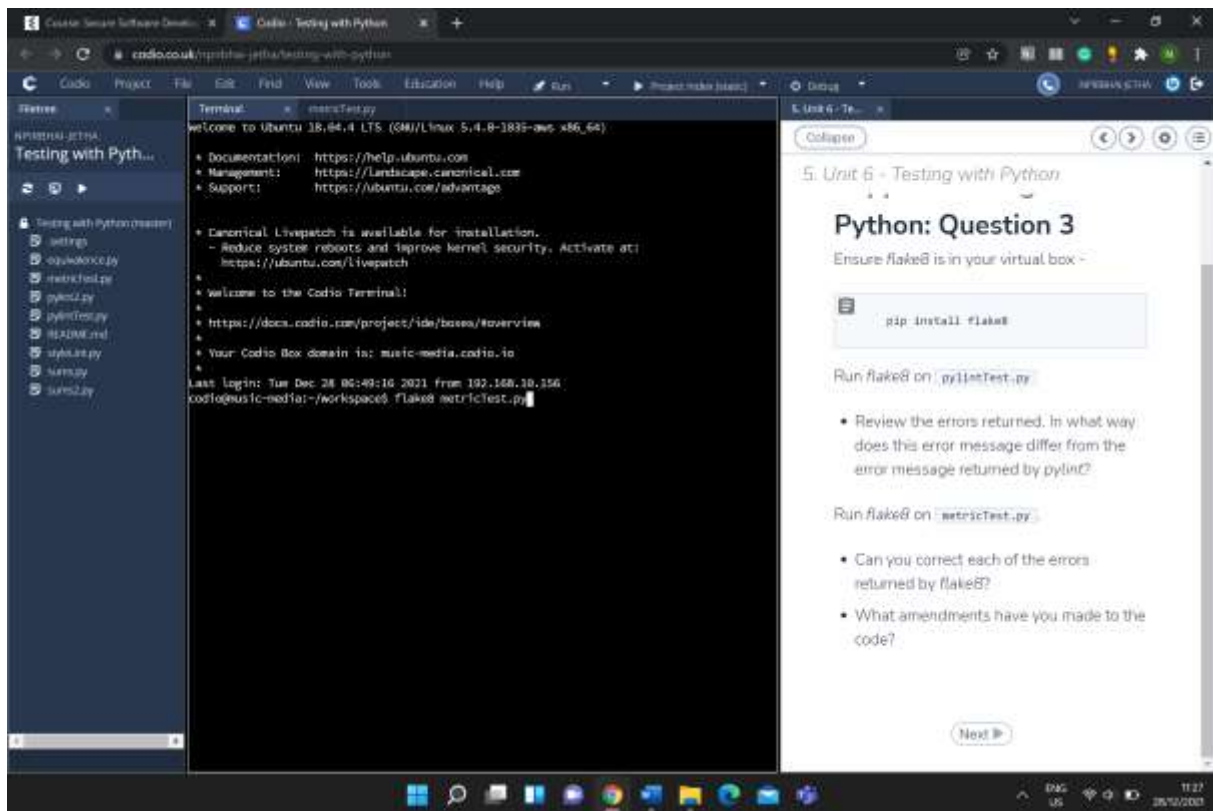
print(encoded)

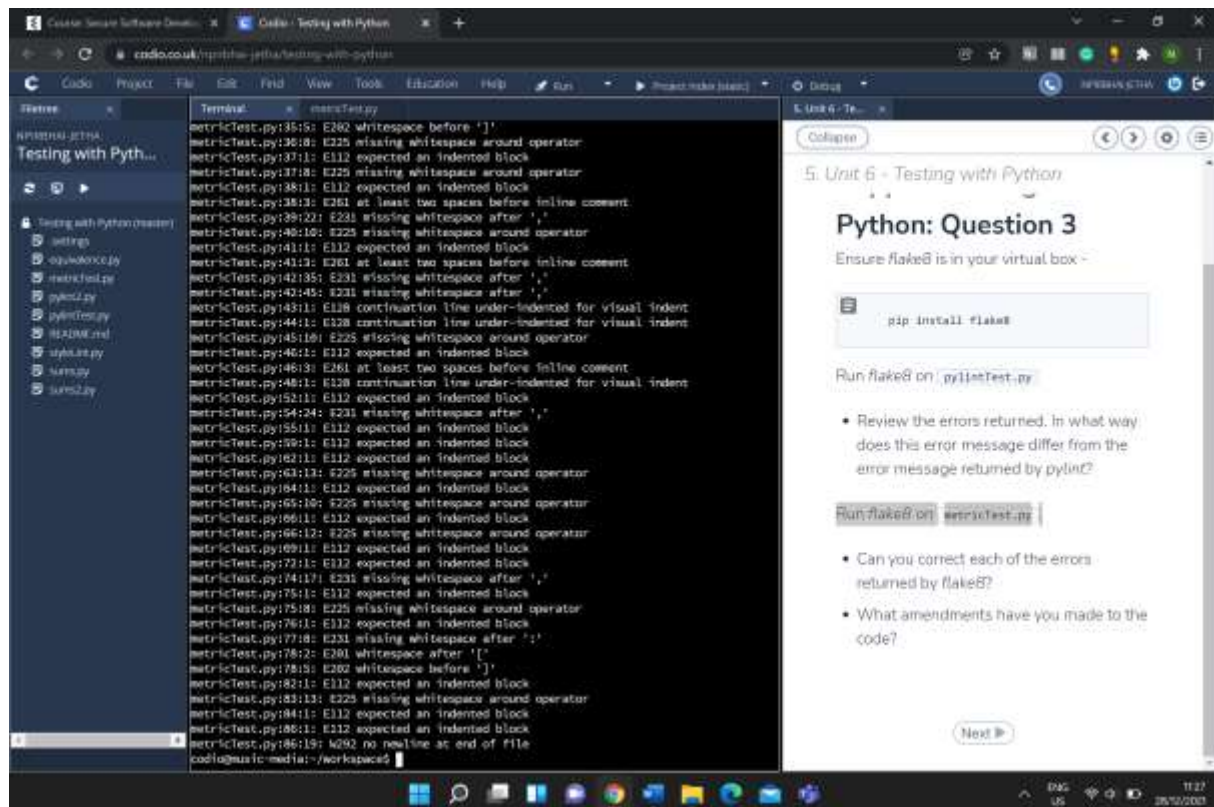
```

CHECK THIS YOUTUBE VIDEO FOR FLAKE8

<https://www.youtube.com/watch?v=TDUf93vqq3g>

Run *flake8* on `metricTest.py`.



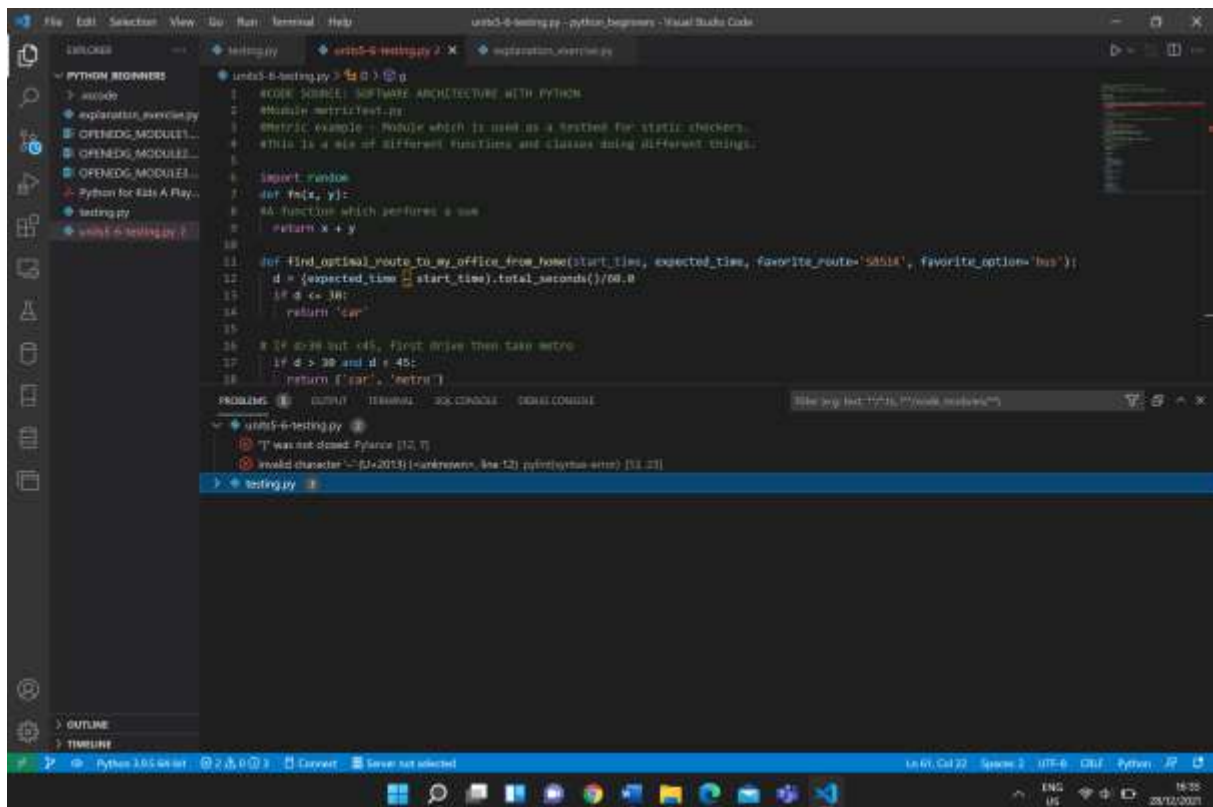


Amendments made:

- Remove all numberings
- Check all indents

For “**Indentation is not a multiple of four (E111)**”, PEP8 recommends that Python code indentation be a multiple of four. [<https://www.python.org/dev/peps/pep-0008/#indentation> or <https://www.flake8rules.com/rules/E111.html>]

- Minus sign checked



```
#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

```


The screenshot shows the Codio IDE interface. On the left, a sidebar lists files under 'Testing with Python...'. The main terminal window displays the Ubuntu 18.04.4 LTS login prompt and system information. The right pane shows a lesson titled '6. Unit 6 - Testing with Python' with a sub-header 'Exploring Linters to Support Testing in Python: Question 4'. The lesson content includes instructions to ensure 'mccabe' is installed and to run it on 'sums.py' and 'sums2.py'.

Terminal Output:

```
welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-1035-aws x86_64)
+ Documentation: https://help.ubuntu.com
+ Management: https://landscape.canonical.com
+ Support: https://ubuntu.com/advantage

+ Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
https://ubuntu.com/livepatch
+
+ Welcome to the Codio Terminal!
+
+ https://docs.codio.com/project/ide/boxes/#overview
+
+ Your Codio Box domain is: music-media.codio.io
+
Last login: Tue Dec 28 11:30:25 2021 from 192.168.18.156
codio@music-media:~/workspace$
```

Lesson Content:

6. Unit 6 - Testing with Python

Exploring Linters to Support Testing in Python: Question 4

Ensure `mccabe` is in your virtual box -

```
pip install mccabe
```

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?

Next ▶

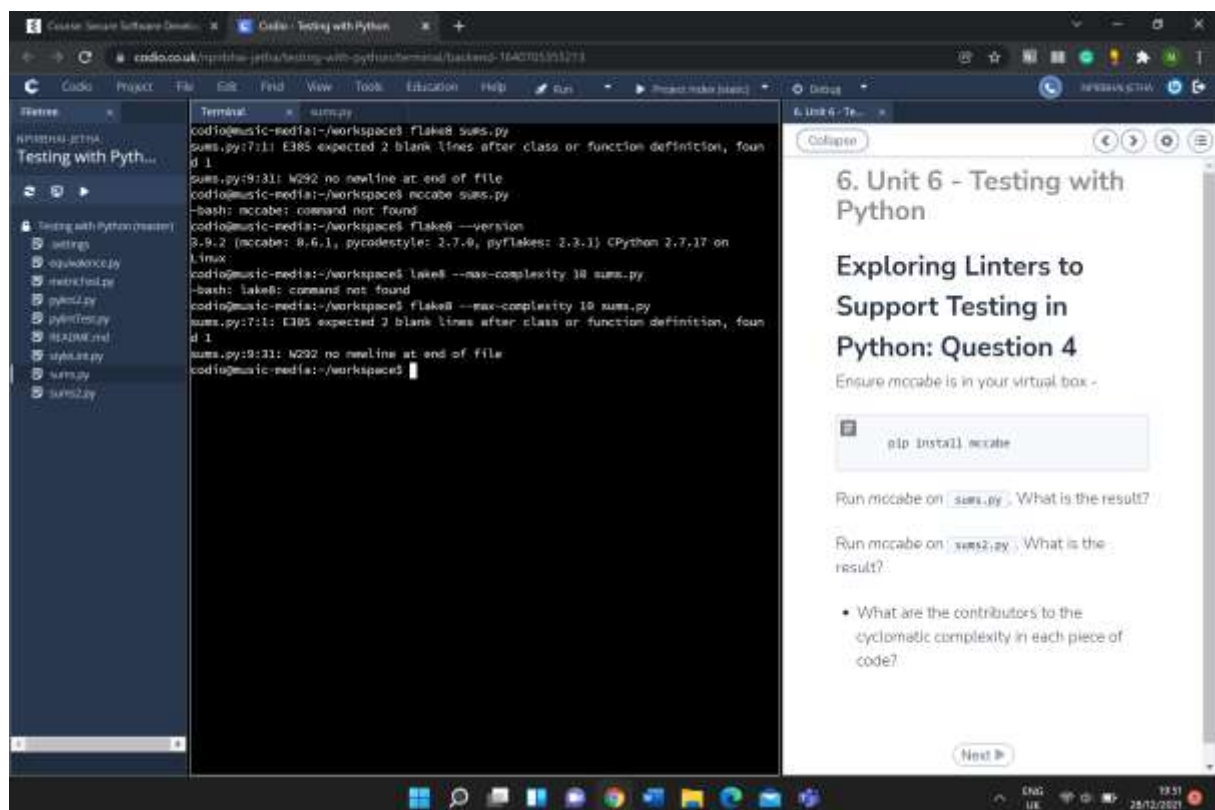
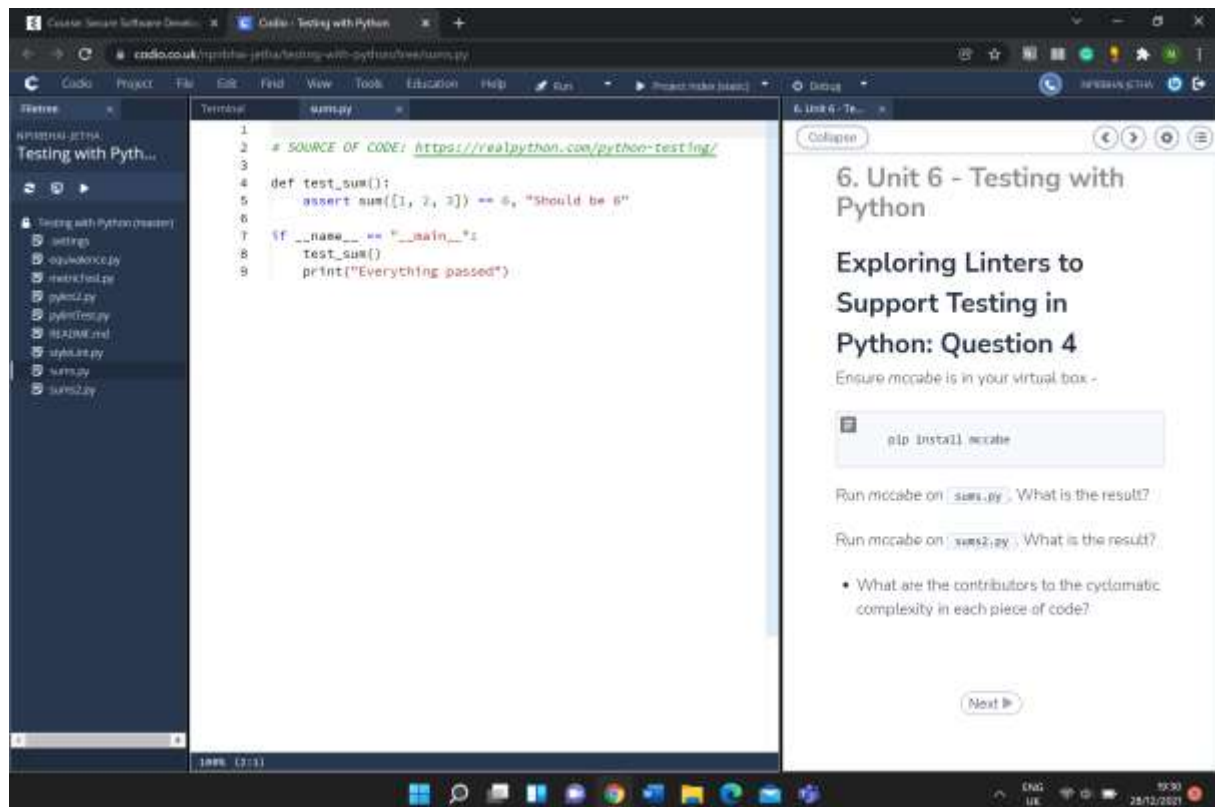
This screenshot shows the same Codio IDE interface as the first, but the terminal window now shows the successful installation of the 'mccabe' package using pip.

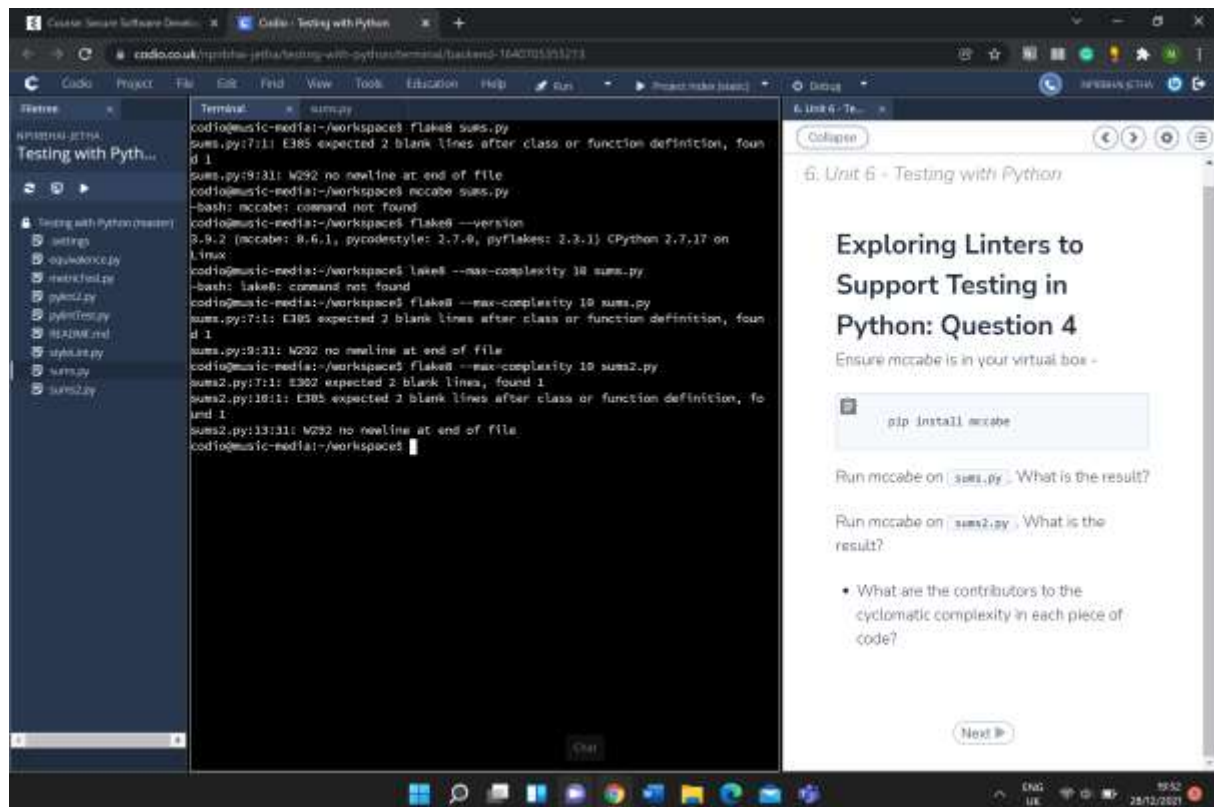
Terminal Output:

```
welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-1035-aws x86_64)
+ Documentation: https://help.ubuntu.com
+ Management: https://landscape.canonical.com
+ Support: https://ubuntu.com/advantage

+ Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
https://ubuntu.com/livepatch
+
+ Welcome to the Codio Terminal!
+
+ https://docs.codio.com/project/ide/boxes/#overview
+
+ Your Codio Box domain is: music-media.codio.io
+
Last login: Tue Dec 28 11:30:25 2021 from 192.168.18.156
codio@music-media:~/workspace$ pip install mccabe
Collecting mccabe
  Using cached https://files.pythonhosted.org/packages/87/69/479dc97638549e21354e93e4e0ef36bd1d237534952483c9681ee6e7b67/mccabe-0.6.1-py2.py3-none-any.whl
Installing collected packages: mccabe
Successfully installed mccabe-0.6.1
codio@music-media:~/workspace$
```

The lesson content on the right remains the same, showing the next steps in the unit.





7. Unit 6 - Testing with Python

Exploring Linters to Support Testing in Python: Question 5 (e-Portfolio Entry)

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

- Specification-based techniques
- Structure-based techniques
- Experience-based techniques

Discuss the scenario(s) in which each technique would be important to be used

• Specification-based techniques	• Structure-based techniques	• Experience-based techniques
<p>There are four specification-based or black-box technique:</p> <ul style="list-style-type: none"> • Equivalence partitioning. • Boundary value analysis. • Decision tables. • State transition testing. <ul style="list-style-type: none"> • The testers have no knowledge of how the system or component is structured inside the box. In black-box testing the tester is concentrating on what the software does, not how it does it. • The definition mentions both functional and non-functional testing. Functional testing is concerned with what the system does its features or 	<p>White-box testing e.g path testing</p> <p>Structure-based techniques serve two purposes: test coverage measurement and structural test case design</p> <ul style="list-style-type: none"> • They are often used first to assess the amount of testing performed by tests derived from specification-based techniques, i.e. to assess coverage. • They are then used to design additional tests with the aim of increasing the test coverage. • Structure-based test design techniques are a good way of generating additional test cases that are 	<p>In experience-based techniques, people's knowledge, skills and background are of prime importance to the test conditions and test cases. The experience of both technical and business people is required, as they bring different perspectives to the test analysis and design process. Because of the previous experience with similar systems, they may have an idea as what could go wrong, which is very useful for testing.</p> <ul style="list-style-type: none"> • Experience-based techniques go together with specification-based and structure-based techniques, and are also used when there is

<p>functions. Non-functional testing is concerned with examining how well the system does. Non-functional testing like performance, usability, portability, maintainability, etc.</p> <ul style="list-style-type: none"> • Specification-based techniques are appropriate at all levels of testing (component testing through to acceptance testing) where a specification exists. For example, when performing system or acceptance testing, the requirements specification or functional specification may form the 	<p>different from existing tests.</p> <ul style="list-style-type: none"> • They can help ensure more breadth of testing, in the sense that test cases that achieve 100% coverage in any measure will be exercising all parts of the software from the point of view of the items being covered. <p>Source: http://tryqa.com/what-is-structure-based-technique-in-software-testing/</p>	<p>no specification, or if the specification is inadequate or out of date.</p> <ul style="list-style-type: none"> • This may be the only type of technique used for low-risk systems, but this approach may be particularly useful under extreme time pressure – in fact this is one of the factors leading to exploratory testing. <p>Source: http://tryqa.com/what-is-experience-based-testing-technique/</p>
--	--	---

<p>basis of the tests.</p> <p>Source: http://tryqa.com/what-is-black-box-specification-based-also-known-as-behavioral-testing-techniques/</p>		
--	--	--