1. Equivalence Testing in Python

Run equivalence.py which is an implementation of equivalence partitioning. This test partitions integers [-3,5] into equivalence classes based on *lambda x*, y: (x-y)%4 = 0. In the output, you should be able to see how a set of objects to be partitioned are considered, and a function evaluates if the two objects are equivalent before printing the result. test_equivalence_partition() produces the following output: set([1, -3]) set([2, -2]) set([3, -1]) set([0, 4]) 0 : set([0, 4]) 1 : set([1, -3]) 2 : set([2, -2]) 3 : set([3, -1]) 4 : set([0, 4]) -2 : set([2, -2]) -3 : set([1, -3]) -1 : set([3, -1]). You should carry out further investigations on the code and experiment with it.

equivalence.py

```
Run
                                              ⊕ Project Index (static) ▼
equivalence.py ×
     # CODE SOURCE: https://stackoverflow.com/questions/38924421/is-there-a-standard-way-to-pa
 2
 3
     def equivalence_partition(iterable, relation):
         """Partitions a set of objects into equivalence classes
 6
         Args:
             iterable: collection of objects to be partitioned
 8
             relation: equivalence relation. I.e. relation(01,02) evaluates to True
 9
                 if and only if o1 and o2 are equivalent
10
         Returns: classes, partitions
11
             classes: A sequence of sets. Each one is an equivalence class
12
13
             partitions: A dictionary mapping objects to equivalence classes
14
         classes = []
15
         partitions = {}
16
17
         for o in iterable: # for each object
18
             # find the class it is in
             found = False
19
             for c in classes:
20
 21
                  if relation(next(iter(c)), o): # is it equivalent to this class?
 22
                     c.add(o)
 23
                      partitions[o] = c
 24
                      found = True
 25
                     break
 26
             if not found: # it is in a new class
 27
                 classes.append(set([o]))
 28
                 partitions[o] = classes[-1]
 29
         return classes, partitions
 30
 31
 32
     def equivalence_enumeration(iterable, relation):
         """Partitions a set of objects into equivalence classes
 33
 34
 35
         Same as equivalence_partition() but also numbers the classes.
 36
 37
 38
             iterable: collection of objects to be partitioned
 39
             relation: equivalence relation. I.e. relation(01,02) evaluates to True
 40
              if and only if o1 and o2 are equivalent
```

```
41
        Returns: classes, partitions, ids
42
          classes: A sequence of sets. Each one is an equivalence class
43
44
           partitions: A dictionary mapping objects to equivalence classes
45
           ids: A dictionary mapping objects to the indices of their equivalence classes
46
47
        classes, partitions = equivalence_partition(iterable, relation)
        ids = {}
48
49
        for i, c in enumerate(classes):
         for o in c:
50
51
             ids[o] = i
        return classes, partitions, ids
52
54
55
   def check_equivalence_partition(classes, partitions, relation):
        """Checks that a partition is consistent under the relationship"""
56
        for o, c in partitions.items():
57
           for _c in classes:
58
                assert (o in _c) ^ (not _c is c)
60
        for cl in classes:
        for ol in cl:
61
62
                for c2 in classes:
63
                    for o2 in c2:
                  assert (c1 is c2) ^ (not relation(o1, o2))
65
66
67
    def test_equivalence_partition():
       relation = lambda x, y: (x - y) \% 4 == 0
        classes, partitions = equivalence_partition(
69
70
           range(-3, 5),
           relation
71
       check_equivalence_partition(classes, partitions, relation)
73
74
        for c in classes: print(c)
75
    for o, c in partitions.items(): print(o, ':', c)
76
77
78 if __name__ == '__main__':
    test_equivalence_partition()
```

Equivalence Partitioning is a testing technique that can be used when the data is too much to test. So, the data is divided into partitions and just samples of each partition will be tested.

The code in 'equivalence.py':

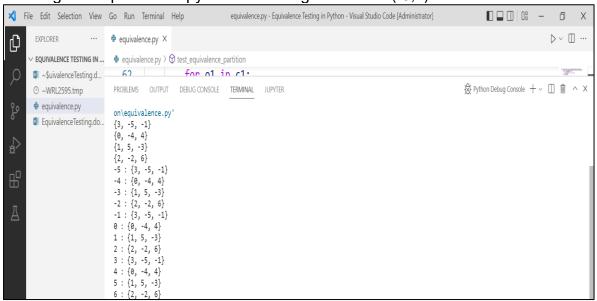
- ☐ First function (equivalence_partition):
 - Makes a class using two objects from iterables. (Two objects have special style which has been mentioned in 'relation')
 - Appends every class to a dictionary called 'partitions'.
 - > To do the above steps, two loops have been used.
 - Returns created classes and partition

This function has been called in two other functions (equivalence_enumeration and test_equivalence_partition).

- ☐ Second function (equivalence_enumeration):
 - Represents the iterables using objects that acts like counters
 - Creates a dictionary called 'ids' and stuff it with the indices of the classes using two loops.
 - Returns created classes, partition, and ids

- ☐ Third function (check_equivalence_partition):
 - Checks for objects in classes and classes in the 'partitions' using assertions.
- ☐ Fourth function (test_equivalence_partition):
 - Mentions the relationships between two objects using the formula (lambda x, y: (x y) % 4 == 0)
 - Displays the range of numbers for partitioning
 - Prints every class
 - Prints every object in every class and every class in the 'partitions' following this pattern: 'object: class'

Running the 'equivalence.py' with new ranges such as (-5,7):



2. Testing with Python

Exploring Linters to Support Testing in Python: Question 1
Run styleLint.py

- What happens when the code is run? There is an Indentation error.
- Can you modify this code for a more favourable outcome? Yes
- What amendments have you made to the code? I indented every line.

styleLint.py

```
≺ File Edit Selection View Go Run Terminal Help
                                                   • styleLint.py - Equivalence Testing in Python - Visual Studio Code [Administrator]

✓ EQUIVALENCE TESTING IN ...  

♦ styleLint.py > 分 factorial

    ~$uivalenceTesting.d...
                         1
     EquivalenceTesting.do...
                          2 # CODE SOURCE: SOFTWARE ARCHITECTURE WITH PYTHON
     metricTest.py
     pylintTest.py
                          4 def factorial(n):
     styleLint.py
                                 """ Return factorial of n """
                          5
                           6
                                  if n == 0:
                           7
                                    return 1
                                  else:
                           8
                              return n*factorial(n-1)
                           9
                          10
```

3. Testing with Python

Exploring Linters to Support Testing in Python: Question 2

```
pip install pylint
```

Run pylint on pylintTest.py

Review each of the code errors returned.

The errors are as follows:

- Module name "styleLint" doesn't conform to snake_case naming
- Missing module docstring pylint (missing-module-docstring)
- Constant name "shift" doesn't conform to UPPER_CASE naming style pylint (invalid-name)
- Bad indentation. Found 2 spaces, expected 4 pylint (bad-indentation)
- Trailing whitespace pylint (trailing-whitespace)
- print(*values: object, sep: str | None = ..., end: str | None = ..., file: SupportsWrite[str] | None = ..., flush: Literal[False] = ...) -> None
- Can you correct each of the errors identified by pylint?
 - For snake-case naming convention, we should change the name of the file into 'pylint test.py'.
 - For module docstring, we should add """ """ before import module.

- For upper-case naming style for constants, we should change the names into 'SHIFT', 'LETTERS', 'ENCODED' and 'X'.
- For bad indentation, we should change the indentation to four spaces
- For trailing whitespace, we should delete whitespaces.
- For the last statement in the code, we should change it into 'print(ENCODED)'.

pylint_test.py

```
Codio Project
                 File Edit Find View Tools Education Help
                pylint_test.py ×
                  2 """ A module for strings """
ESARMAD
Testing with Pyt...
                  3 # SOURCE OF CODE: https://docs.pylint.org/en/1.6.0/tutorial.html
                  4 import string
φ 🖂 🕨
                 5 SHIFT = 3
                 6 choice = input("would you like to encode or decode?")
7 word = (input("Please enter text: "))
Testing with Python (mas
 equivalence.py
                 8 LETTERS = string.ascii_letters + string.punctuation + string.digits
                  9 ENCODED = ''
 metricTest.py
 pylint_test.py
                 10 if choice == "encode":
                       for letter in word:
 pylintTest.py
                 11
 README.md
                          if letter == ' ':
                 12
 styleLint.py
                                  ENCODED = ENCODED + ' '
                 13
 sums2.py
                 14
                              else:
                 15
                                  X = LETTERS.index(letter) + SHIFT
                 16
                                  ENCODED = ENCODED + LETTERS[X]
                 17 if choice == "decode":
                        for letter in word:
                 18
                 19
                            if letter == ' ':
                 20
                                  ENCODED = ENCODED + ' '
                 21
                                X = LETTERS.index(letter) - SHIFT
                 22
                  23
                                   ENCODED = ENCODED + LETTERS[X]
                 24 print(ENCODED)
```

4. Testing with Python

Exploring Linters to Support Testing in Python: Question 3

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?

The errors are as follows:

- Whitespace error
- "raw_input" is not defined Pylance (reportUndefinedVariable)
- indentation is not a multiple of 4 flake8(E111)
- missing whitespace around operator flake8(E225)
- no newline at end of file flake8(W292)

The corrections are as follows:

- For whitespace error, we should remove the whitespaces after import module
- For raw-input(), we should change it into 'input()'.
- For indentation error, we should change the indentation into 4 spaces.
- For missing whitespace, we should put whitespace around = operator like this: 'encoded = encoded + letters[x]'
 For newline error, we should change the place of cursor from this line (print(encoded)) to the next line.

pylintTest.py

```
Codio Project File Edit Find View Tools Education Help
                                                      ☑ Run ▼ ⊕ Project Index (static) ▼ 👸 Debug ▼
Testing with Pyt...
                 2 # SOURCE OF CODE: https://docs.pylint.org/en/1.6.0/tutorial.html
                 3 import string
Φ 🗵 🕨
                 4 shift = 3
                  5 choice = input("would you like to encode or decode?")
Testing with Python (mas
                  6 word = (input("Please enter text: "))
                 7 letters = string.ascii_letters + string.punctuation + string.digits
 equivalence.py
 metricTest.py
                 8 encoded = ''
 pylint_test.py
                 9 if choice == "encode":
 pylintTest.py
                10 for letter in word:
 README.md
                         if letter == ' ':
                 11
 styleLint.py
                 12
                               encoded = encoded + ' '
 sums.py
 sums2.py
                 13
                             else:
                        x = letters.index(letter) + shift
encoded = encoded + letters[x]
                 14
15
                 16 if choice == "decode":
                 17
                     for letter in word:
                           if letter == ' ':
                 18
                 19
                               encoded = encoded + ' '
                 20
                             else:
                               x = letters.index(letter) - shift
                 21
                 22
                               encoded = encoded + letters[x]
                 23
                 24 print(encoded)
```

Run flake8 on metricTest.py

Can you correct each of the errors returned by flake8?

The errors are as follows:

- block comment should start with '#' flake8(E265)
- trailing whitespace flake8(W291)
- expected 2 blank lines, found 1 flake8(E302)
- missing whitespace after ',' flake8(E231)
- line too long (121 > 79 characters) flake8(E501)
- indentation is not a multiple of 4flake8(E111)
- IndentationError: unindent does not match any outer indentation levelflake8(E999)

What amendments have you made to the code?

- For block comment with '#' error, we should put a space after '#'.
- For trailing whitespace error, we should remove spaces at the end of the line.
- For 2 blank lines, we should put two blank lines before functions,...
- For missing whitespace after ',', we should put ',' after the name of variable such as 'start_time, expected_time'.
- For too long line error, we should just put not more than 79 characters in a line.
- For indentation error, we should change the indentation into 4 spaces.
- For indentationError, we should change 'if d>45'.' into 'if d > 45'.

metricTest.py

```
■ Run
                                                                     ⊕ Project Index (static) ▼
Filetree
                 Terminal
                            metricTest.py × Run
                  2 # CODE SOURCE: SOFTWARE ARCHITECTURE WITH PYTHON
ESARMAD
                   3
Testing with Pyt...
                      Module metricTest.py
φ 🖂 🕨
                       Metric example - Module which is used as a testbed for static checkers.
                  6
                       This is a mix of different functions and classes doing different things.
Testing with Python (mas
 .settings
                   7
 equivalence.pv
                  8
 metricTest.pv
                  9 import random
 pylint_test.py
                 10
 pylintTest.py
                 11
                 12 def fn(x, y):
 styleLint.py
                         """ A function which performs a sum """
                 13
 sums.py
                 14
                           return x + y

■ sums2.py

                  15
                  16
                  17
                      def find_optimal_route_to_my_office_from_home(start_time, expected_time,
                                                                        favorite_route='SBS1K',
                  18
                  19
                                                                        favorite_option='bus'):
                  20
                  21
                           d = (expected time, start time).total seconds() / 60.0
                  22
                           if d <= 30:
                  23
                             return 'car'
                           # If d > 30 but <45, first drive then take metro
```

```
if d > 30 and d < 45:
                  25
                              return ('car', 'metro')
Testing with Pyt...
                  26
                  27
 φ 🖸 🕨
                            """If d>45 there are a combination of optionsWriting Modifiable
                  28
                            and Readable Code"""
                  29
settings .
                  30
                            if d > 45:
 equivalence.py
                  31
 metricTest.py
                                if d < 60:
                  32
 pylint_test.py
                  33
                                    # First volvo, then connecting bus
 pylintTest.py
                  34
                                    return ('bus: 335E', 'bus: connector')

■ README.md

                                elif d > 80:
                  35
 styleLint.py
                                     # Might as well go by normal bus
                  36
 sums.py
                                     return random.choice(('bus: 330', 'bus: 331', ':'.join((
 sums2.py
                  37
                   38
                                        favorite_option, favorite_route))))
                                elif d > 90:
                   39
                                     # Relax and choose favorite route
                  40
                  41
                                     return ':'.join((favorite_option, favorite_route))
                  42
                   43
                  44
                       class C(object):
                   45
                            """ A class which does almost nothing """
                   46
                  47
                            def __init__(self, x, y):
                                self.x = x
                   48
                   49
                                self.y = y
Testing with Pyt...
                   50
 φ <u>N</u>
                   51
                            def f(self):
                   52
                            pass
Testing with Python (mas
                   53
 .settings
 equivalence.py
                   54
                            def g(self, x, y):
 metricTest.py
                   55
                                if self.x > x:
 pylint_test.py
                   56
                                    return self.x+self.y
 pylintTest.py
                  57
                                elif x > self.x:
 ■ README.md
                   58
                                  return x + self.y
 styleLint.py
                   59
 sums.py
                   60
 sums2.py
                       class D(C):
                   61
                            """ D class """
                   62
                   63
                            def __init__(self, x):
                   64
                              self.x = x
                   65
                            def f(self, x, y):
                   66
                   67
                                if x > y:
                   68
                                    return x-y
                   69
                                else:
                   70
                                  return x+y
                   71
                   72
                            def g(self, y):
Testing with Pyt...
                   73
                                if self.x > y:
 Φ 🖸 🕨
                   74
                                    return self.x + y
                  75
                                else:
Testing with Python (mas
                   76
                                    return y - self.x
 .settings
 equivalence.py
                   77
 metricTest.py
```

5. Testing with Python

Python: Question 4

pip install mccabe

sums.py

```
Edit Find View Tools Education Help
                                                       1
Testing with Python
                     2
                         # SOURCE OF CODE: https://realpython.com/python-testing/
                     3
Ф 🖂 🕨
                        def test_sum():
                           assert sum([1, 2, 3]) == 6, "Should be 6"
                     5
Testing with Python (master)
                     6
 settings .
                        if __name__ == "__main__":
                     7
 equivalence.pv
 metricTest.py
                     8
                             test_sum()
 pylint_test.py
                             print("Everything passed")
 pylintTest.py
 README.md
 styleLint.py
 sums.py
 sums2.py
```

sums2.py

```
Edit Find View Tools Education Help 🖸 Run 🔻 🌐 Project Index (static) 💌 💆 Debug 🔻
🤃 Codio Project File
ESARMAD
Testing with Python
                            # SOURCE OF CODE: https://realpython.com/python-testing/
                        3
φ 🖂 🕨
                            def test_sum():
                        4
                                assert sum([1, 2, 3]) == 6, "Should be 6"
Testing with Python (master)
                        6
 .settings
                           def test_sum_tuple():
                        7
 equivalence.py
                               assert sum((1, 2, 2)) == 6, "Should be 6"
 metricTest.py
                        8
 pylint_test.py
 pylintTest.py
                            if __name__ == "__main__":
                       10
 ■ README.md
                       11
                                test_sum()
 styleLint.py
                       12
                                test_sum_tuple()
 sums.py sums2.py
                                print("Everything passed")
```

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

```
$ python -m mccabe --min 1 sums.py
3:0: 'test_sum' 1
If 6 2
```

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

```
$ python -m mccabe --min 1 sums2.py
4:0: 'test_sum' 1
7:0: 'test_sum_tuple' 1
If 10 2
```

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

In 'sums.py', the cyclomatic complexity numbers for 'test_sum' function is 3 which means that the function is easy to understand and test. The reason behind it is that the function is small, and it has just used one assertion and one function invocation.

In 'sums2.py', the cyclomatic complexity numbers for 'test_sum' and 'test_sum_tuple' are less than 10. It means that the functions are considered as a medium complexity because of using 2 assertions and 2 function invocations. So, there are more lines of code.