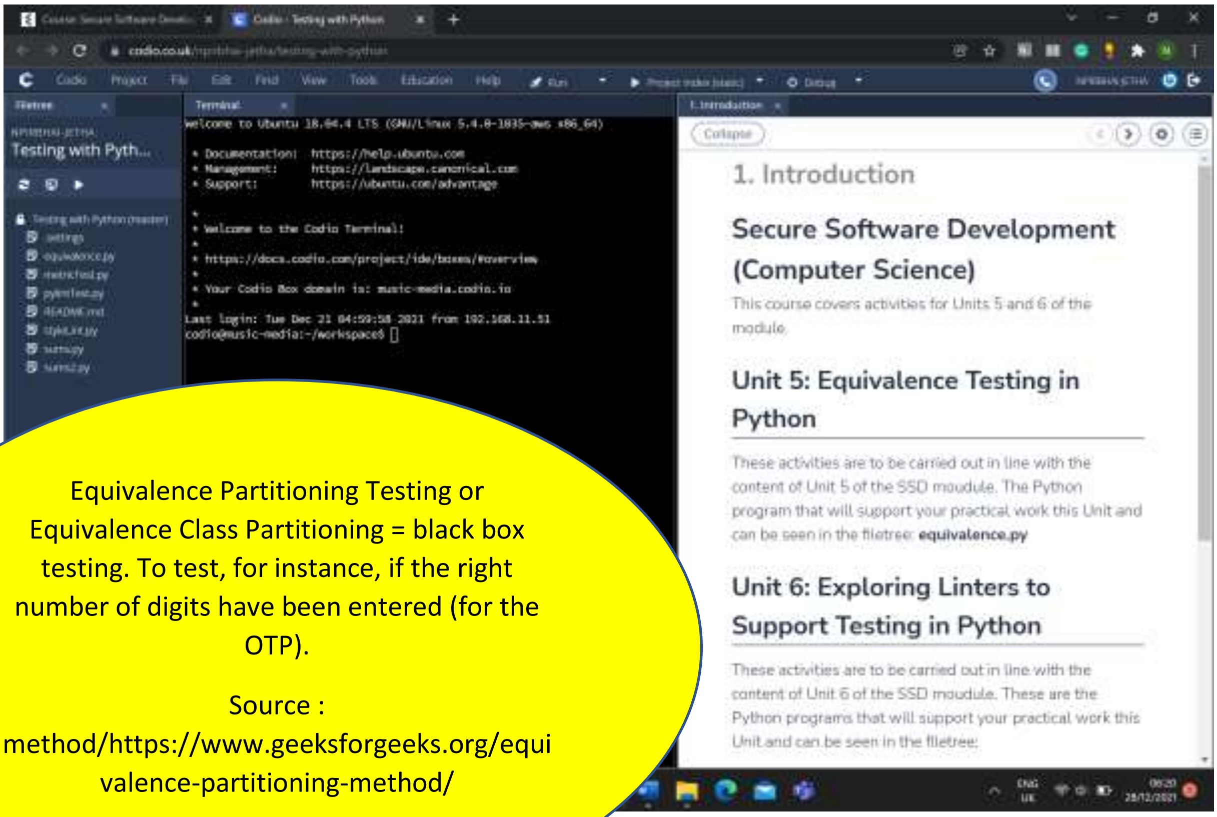
# Unit 5: Equivalence Testing in Python



```
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/#overview
 *
 * Your Codio Box domain is: music-media.codio.io
 *
Last login: Tue Dec 21 04:59:58 2021 from 192.168.11.51
codio@music-media:~/workspace$ 
```

1. Introduction

## Secure Software Development (Computer Science)

This course covers activities for Units 5 and 6 of the module.

## Unit 5: Equivalence Testing in Python

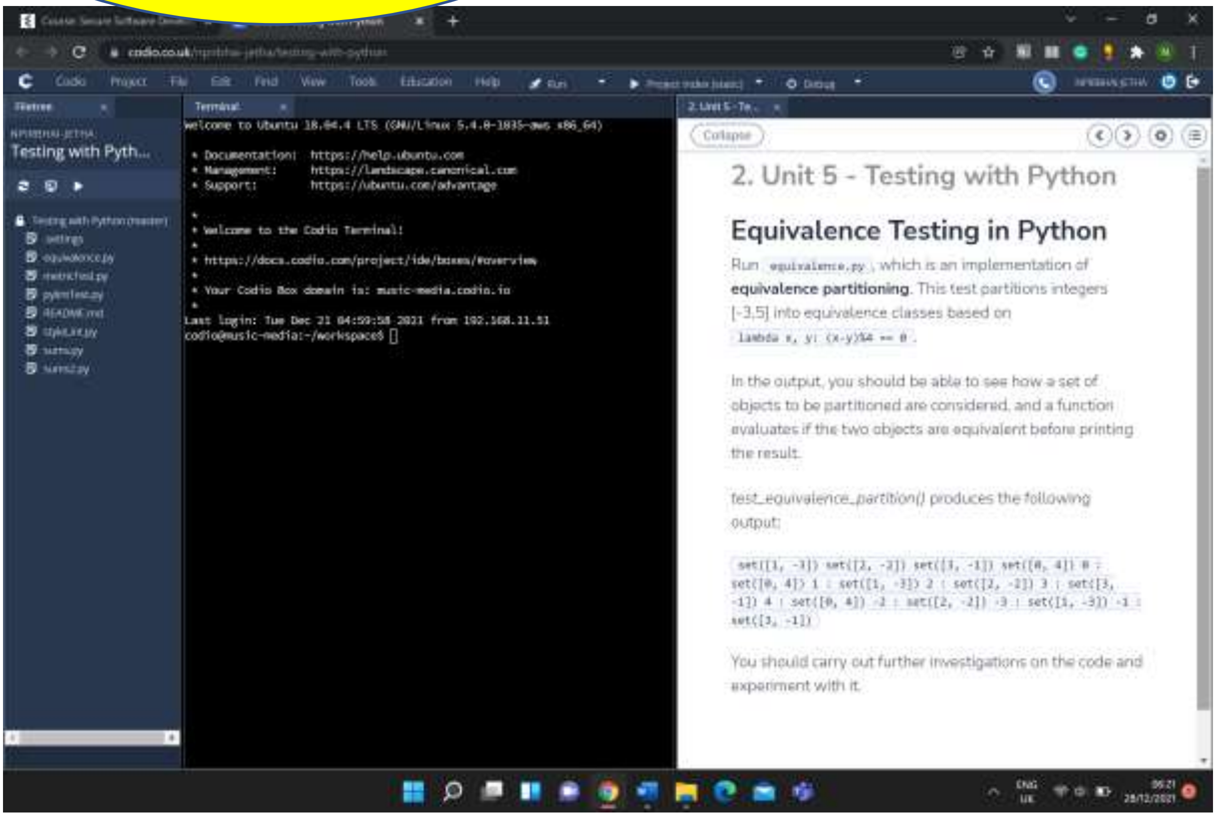
These activities are to be carried out in line with the content of Unit 5 of the SSD moudule. The Python program that will support your practical work this Unit and can be seen in the filetree: **equivalence.py**

## Unit 6: Exploring Linters to Support Testing in Python

These activities are to be carried out in line with the content of Unit 6 of the SSD moudule. These are the Python programs that will support your practical work this Unit and can be seen in the filetree:

Equivalence Partitioning Testing or Equivalence Class Partitioning = black box testing. To test, for instance, if the right number of digits have been entered (for the OTP).

Source : method/https://www.geeksforgeeks.org/equivalence-partitioning-method/



```
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/#overview
 *
 * Your Codio Box domain is: music-media.codio.io
 *
Last login: Tue Dec 21 04:59:58 2021 from 192.168.11.51
codio@music-media:~/workspace$ 
```

2. Unit 5 - Testing with Python

## 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.

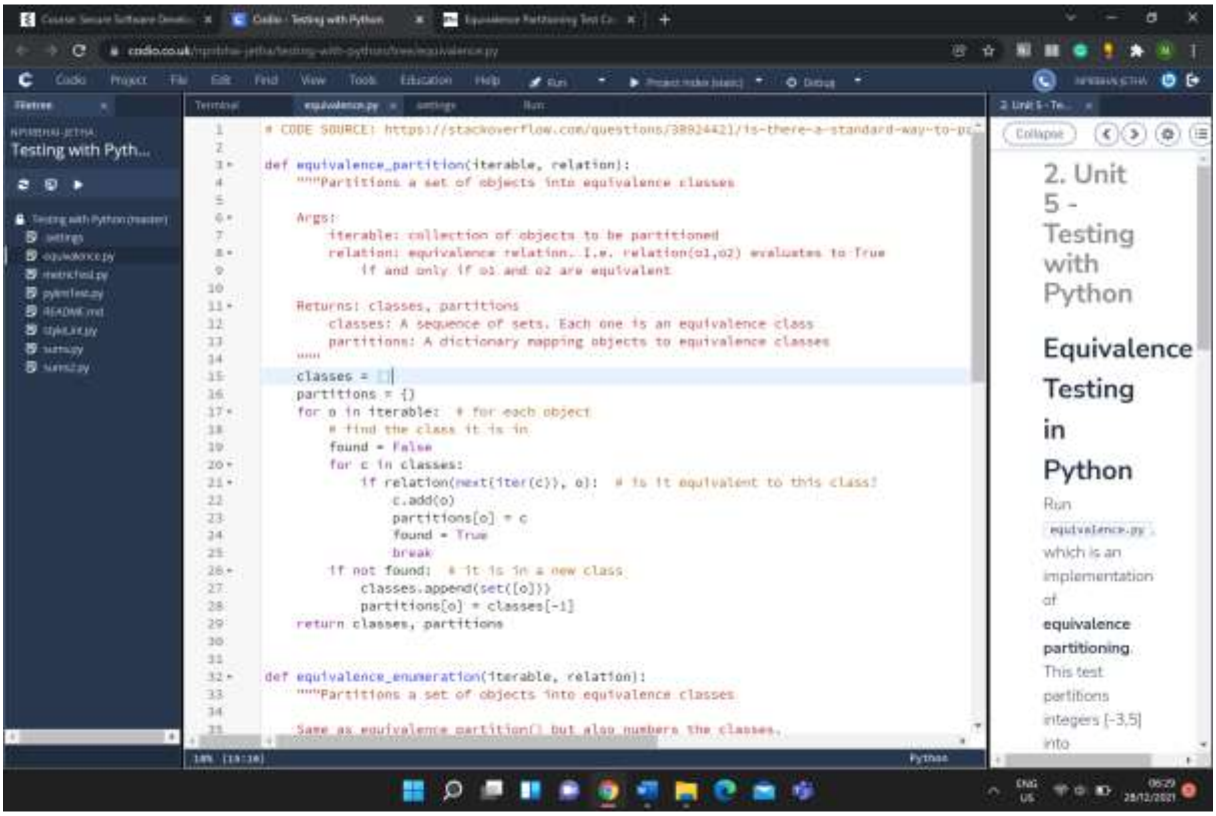
*test_equivalence_partition()* produces the following output:

```
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.

**# CODE SOURCE for equivalence testing** : https://stackoverflow.com/questions/38924421/is-there-a-standard-way-to-partition-an-interable-into-equivalence-classes-given/38924631#38924631

```
def equivalence_partition(iterable, relation):
    """Partitions a set of objects into equivalence classes



    Args:
        iterable: collection of objects to be partitioned
        relation: equivalence relation. I.e. relation(o1,o2) evaluates to True
            if and only if o1 and o2 are equivalent

    Returns: classes, partitions
        classes: A sequence of sets. Each one is an equivalence class
        partitions: A dictionary mapping objects to equivalence classes
    """
    classes = []
    partitions = {}
    for o in iterable:  # for each object
        # find the class it is in
        found = False
        for c in classes:
            if relation(next(iter(c)), o):  # is it equivalent to this class?
                c.add(o)
                partitions[o] = c
                found = True
                break
        if not found:  # it is in a new class
            classes.append(set([o]))
            partitions[o] = classes[-1]
    return classes, partitions



def equivalence_enumeration(iterable, relation):
    """Partitions a set of objects into equivalence classes

    Same as equivalence_partition() but also numbers the classes.

    Args:
        iterable: collection of objects to be partitioned
        relation: equivalence relation. I.e. relation(o1,o2) evaluates to True
            if and only if o1 and o2 are equivalent

    Returns: classes, partitions, ids
        classes: A sequence of sets. Each one is an equivalence class
        partitions: A dictionary mapping objects to equivalence classes
        ids: A dictionary mapping objects to the indices of their equivalence classes
    """
    classes, partitions = equivalence_partition(iterable, relation)
    ids = {}
    for i, c in enumerate(classes):
        for o in c:
            ids[o] = i
    return classes, partitions, ids



def check_equivalence_partition(classes, partitions, relation):
    """Checks that a partition is consistent under the relationship"""
    for o, c in partitions.items():
        for _c in classes:
            assert (o in _c) ^ (not _c is c)
    for c1 in classes:
        for o1 in c1:
            for c2 in classes:
                for o2 in c2:
                    assert (c1 is c2) ^ (not relation(o1, o2))


def test_equivalence_partition():
    relation = lambda x, y: (x - y) % 4 == 0
    classes, partitions = equivalence_partition(
        range(-3, 5),
        relation
    )
    check_equivalence_partition(classes, partitions, relation)
    for c in classes: print(c)
    for o, c in partitions.items(): print(o, ':', c)


if __name__ == '__main__':
    test_equivalence_partition()
```

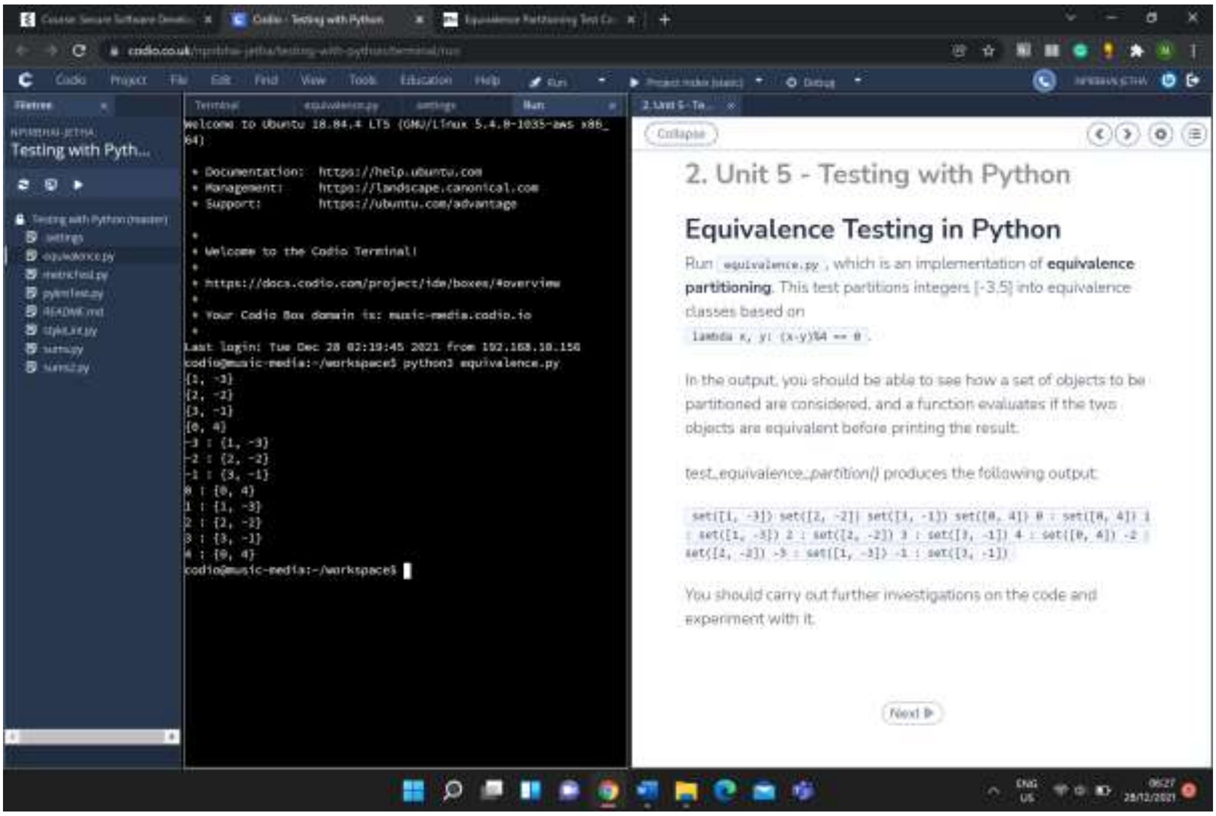


```
# CODE SOURCE: https://stackoverflow.com/questions/38924421/is-there-a-standard-way-to-pa

def equivalence_partition(iterable, relation):
    """Partitions a set of objects into equivalence classes

    Args:
        iterable: collection of objects to be partitioned
        relation: equivalence relation. I.e. relation(o1,o2) evaluates to True
            if and only if o1 and o2 are equivalent

    Returns: classes, partitions
        classes: A sequence of sets. Each one is an equivalence class
        partitions: A dictionary mapping objects to equivalence classes
    """
    classes = []
    partitions = {}
    for o in iterable:  # for each object
        # find the class it is in
        found = False
        for c in classes:
            if relation(next(iter(c)), o):  # is it equivalent to this class?
                c.add(o)
                partitions[o] = c
                found = True
                break
        if not found:  # it is in a new class
            classes.append(set([o]))
            partitions[o] = classes[-1]
    return classes, partitions


def equivalence_enumeration(iterable, relation):
    """Partitions a set of objects into equivalence classes

    Same as equivalence_partition() but also numbers the classes.
```

2. Unit 5 - Testing with Python

Equivalence Testing in Python

Run `equivalence.py` which is an implementation of **equivalence partitioning**. This test partitions integers [-3,5] into



```
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/#overview
 *
 * Your Codio Box domain is: music-media.codio.io
 *
Last login: Tue Dec 28 02:19:45 2021 from 192.168.10.156
codio@music-media:~/workspace$ python3 equivalence.py
{1, -3}
{2, -2}
{3, -1}
{0, 4}
-3 : {1, -3}
-2 : {2, -2}
-1 : {3, -1}
0 : {0, 4}
1 : {1, -3}
2 : {2, -2}
3 : {3, -1}
4 : {0, 4}
codio@music-media:~/workspace$ 
```

## 2. Unit 5 - Testing with Python

## 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.