

---

## Unit 5: An Introduction to Testing

---

### Unit 5 Activity: Equivalence Testing in Python

#### Task:

Run `equivalence.py` in your chosen Jupyter Notebook workspace - Testing with Python - 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.

#### Answer:

The script executed successfully and produced the following output:

```
C: > Users > pc > Documents > 00000_Essex > 03_Secure Software Development > 05_Unit > Activity-EvEquivalence Testing in Python >  equivalence-partitioning.py
1 # CODE SOURCE: https://stackoverflow.com/questions/38924421/is-there-a-standard-way-to-partition-an
2
3 def equivalence_partition(iterable, relation):
4     """Partitions a set of objects into equivalence classes
5
6     Args:
7         iterable: collection of objects to be partitioned
8         relation: equivalence relation. I.e. relation(o1,o2) evaluates to True
9                 if and only if o1 and o2 are equivalent
10
11     Returns: classes, partitions
12             classes: A sequence of sets. Each one is an equivalence class
13 """
14
15 # Create a list of sets, each containing one object from the iterable
16 # and all objects that are equivalent to it according to the relation
17 classes = []
18 for obj in iterable:
19     # Create a new set containing the current object
20     new_set = {obj}
21     # Iterate over all objects in the iterable
22     for other_obj in iterable:
23         # If the current object and the other object are equivalent
24         # according to the relation, add the other object to the set
25         if relation(obj, other_obj):
26             new_set.add(other_obj)
27     # Add the new set to the list of sets
28     classes.append(new_set)
29
30 # Return the list of sets
31 return classes
32
33 # Test the equivalence_partition function
34 # Create a list of integers
35 integers = [1, -3, 2, -2, 3, -1, 0, 4]
36 # Create a relation function that returns True if two integers have the same absolute value
37 def relation(o1, o2):
38     return abs(o1) == abs(o2)
39
40 # Partition the integers into equivalence classes
41 classes = equivalence_partition(integers, relation)
42
43 # Print the classes
44 for i, class_obj in enumerate(classes):
45     print(i, : {0, 4}
46     1 : {1, -3}
47     2 : {2, -2}
48     3 : {3, -1}
49     4 : {0, 4}
50
51 PS C:\Users\pc>

```

Equivalence classes:

```
{1, -3}
{2, -2}
{3, -1}
{0, 4}
```

Mapping of each integer to its equivalence class:

```
-3 : {1, -3}
-2 : {2, -2}
-1 : {3, -1}
0  : {0, 4}
1  : {1, -3}
2  : {2, -2}
3  : {3, -1}
4  : {0, 4}
```

This matches the partitioning based on the equivalence relation  $(x-y) \bmod 4 = 0$ . Each set contains numbers that are equivalent modulo 4.