# Unit testing using Pytest

Name: Vikrant Singh, Student no: 21262315, Module: CA650 Software Process Quality, Programme: MSc. in computing in secure software engineering

- **Why python pytest instead of X-unit for this assignment?**

There is no such a big difference while testing with the x-unit and pytest except python is more flexible and powerful due to the availability of more libraries. The library hypothesis in python available for testing add more flexibility to the testing because it provides functions to throw different input streams to test many possible types of inputs. It's like the theories in x-unit but we don't have to add every test manually we can create a list of values of a different sort and pass these values at once.

- **Code of the queue used for testing [2]**

```
import math


class Queue(object):
    def __init__(self):
        self.queue = None
        if self.queue is None:
            self.queue = []
        else:
            self.queue = list(queue)

    def __str__(self):
        myString = ' '.join(str(i) for i in self.queue)
        return myString

    def enqueue(self, v):
        # print(v)
        if v is None:
            raise ValueError("Variable 'v' is None")
        if isinstance(v, float):
            if math.isnan(v):
                raise ValueError("Variable 'v' is NaN")
            if math.isinf(v):
                raise ValueError("Variable 'v' is Infinity")
        self.queue.insert(0, v)

    def dequeue(self):
        if not self.queue:
            return None
        else:
            return self.queue.pop()
```

```
    def len(self):
        return len(self.queue)

    def isEmpty(self):
        return self.queue == []


q = Queue()
q.enqueue(2)
q.enqueue(3)


q.enqueue(4)
q.enqueue(4)
print(q)
q.dequeue()
print(q)
print(q.len())
```

- **Testing coed in pytest[2]**

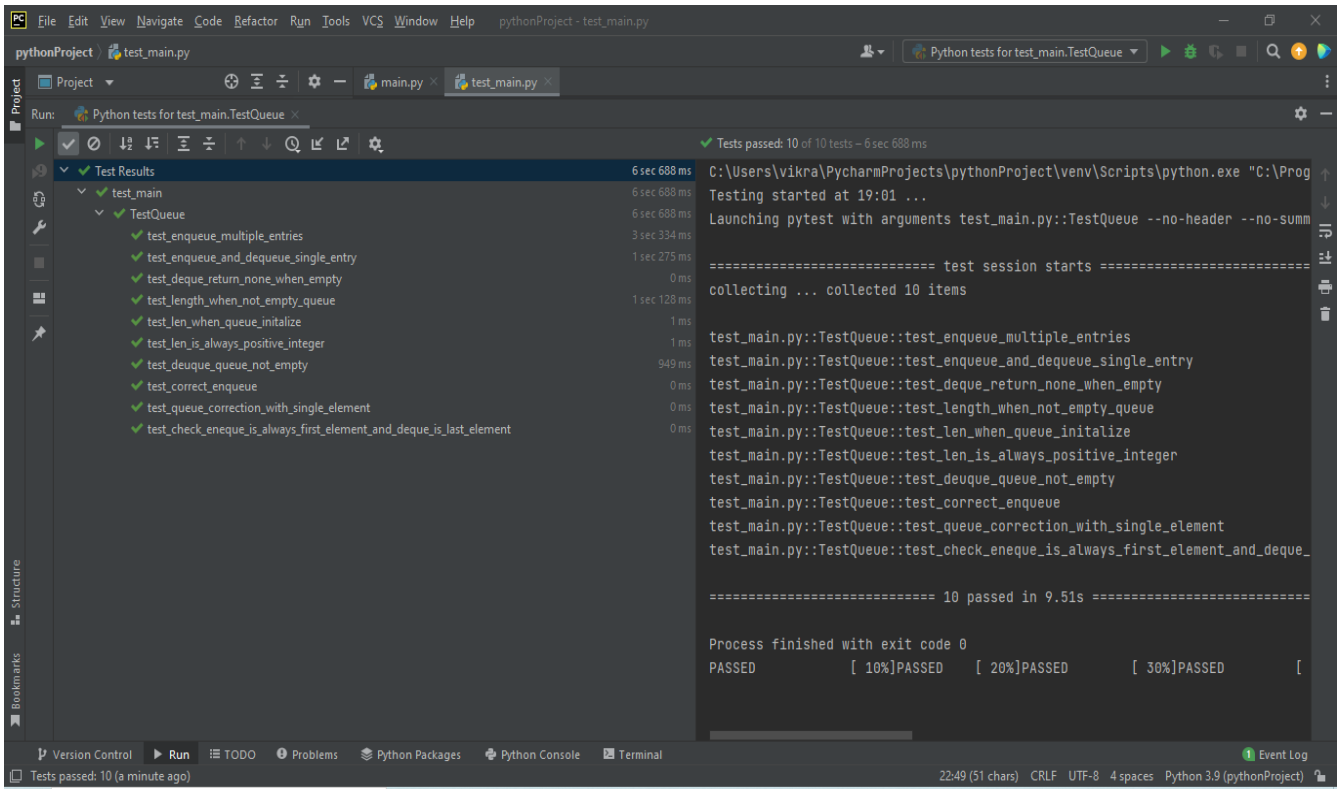
```
import hypothesis.strategies as st
from hypothesis import given, example
import main
import pytest


# vals, special_vals and more_vals are testing values generated using
hypothesis.strategies

class TestQueue:

vals=(st.lists(elements=(st.floats(allow_nan=False,allow_infinity=False) |
st.text() | st.integers() | st.booleans() |
st.tuples(st.integers(),st.booleans()) | st.dictionaries(st.integers(),
st.text(), min_size=1))))

special_vals=(st.lists(elements=(st.floats(allow_nan=False,allow_infinity=F
alse)) | st.text() | st.integers() | st.booleans() |
st.tuples(st.integers(), st.booleans()) | st.dictionaries(st.integers(),
st.text(), min_size=1), min_size=1))


more_vals=(st.lists(elements=(st.floats(allow_nan=False,allow_infinity=Fals
e)) | st.text() | st.integers() | st.booleans() | st.tuples(st.integers(),
st.booleans()) | st.dictionaries(st.integers(), st.text(), min_size=1),
min_size=1, max_size=1))

    def setup_method(self):
        pass

    # testing functions for multiple enqueue/dequeue ..
```

```
    @given(vals)
    def test_enqueue_multiple_entries(self, a):
        queue = main.Queue()
        assert queue.len() == 0
        for x in a:
            # print (x)
            queue.enqueue(x)
        assert queue.len() == len(a)
        for x in a:
            queue.dequeue()
        assert queue.dequeue() is None
        assert queue.isEmpty() is True  # checked here we can also make a
new method to test is empty

    # Testing functions for single enqueue/dequeue
    @given(vals)
    def test_enqueue_and_dequeue_single_entry(self, a):  # a=[1, 2, 5]) if
want to pass values by ourselves
        queue = main.Queue()
        for x in a:
            # print(x)
            expected_enqueue_len = queue.len() + 1
            # print(expected_len)
            queue.enqueue(x)
            assert queue.len() is expected_enqueue_len

            expected_deque_len = queue.len() - 1
            queue.dequeue()
            assert queue.len() is expected_deque_len

    # 3 tests below are also mention in first method but redefine here for
more clear testing criteria
    # testing for 1. queue return None if empty
    # 2. test length when queue is not empty
    # 3. test length of queue is zero when its initialized
    def test_deque_return_none_when_empty(self):
        queue = main.Queue()
        expected_results = queue.dequeue()
        assert expected_results is None

    @given(vals)
    def test_length_when_not_empty_queue(self, a):
        queue = main.Queue()
        for x in a:
            queue.enqueue(x)
        assert queue.len() == len(a)

    def test_len_when_queue_initalize(self):
        queue = main.Queue()
        try:
            assert queue.len() == 0
        except:
            assert False
```

```
    def test_len_is_always_positive_integer(self):
        queue = main.Queue()
        try:
            assert isinstance(queue.len(), int) and (queue.len() >= 0)
        except:
            assert False

    # test deque doesn't return None when queue is non-empty
    @given(special_vals)  # This test fails with vals even the expected and
actual answer are same so I dont know
    # the reason but passes with special_vals and more_vals)
    def test_deuque_queue_not_empty(self, a):
        queue = main.Queue()
        for x in a:
            queue.enqueue(x)
        results_dequeue = queue.dequeue()
        assert results_dequeue is not None

    # Test enqueue inserts multiple element at correct position
    def test_correct_enqueue(self, a=[1, 2, 5]):
        queue = main.Queue()
        try:
            for x in a:
                queue.enqueue(x)
            # print(queue.len())
            # print(len(a))
            if queue.len() is len(a):
                for x in a:
                    result = queue.dequeue()
                    assert result is x
            else:
                assert False
        except:
            False

    # Test enqueue inserts single element at correct position
    def test_queue_correction_with_single_element(self, a=[1, 2, 5]):  #
make test small here so that we can track
        # length and element
        queue = main.Queue()

        try:
            for x in a:
                queue.enqueue(x)
            queue.enqueue(7)
            assert queue.len() is 4
            result = queue.dequeue()
            assert result
            assert queue.len() is 0
        except:
            False
```

```
    # Test to check eneque is always first element and deque is last
element
    def eneque_is_always_first_element_and_deque_is_last_element(self):
        queue = main.Queue()
        try:
            queue.enqueue('hello')
            queue.enqueue(1)
            result = queue.dequeue()
            assert result is 'hello'
            queue.enqueue('@')
            result_again = queue.dequeue()
            # print(result_again)
            assert result_again is 1
        except:
            False
```

**Results:**




**Explanation of failed Test case:**

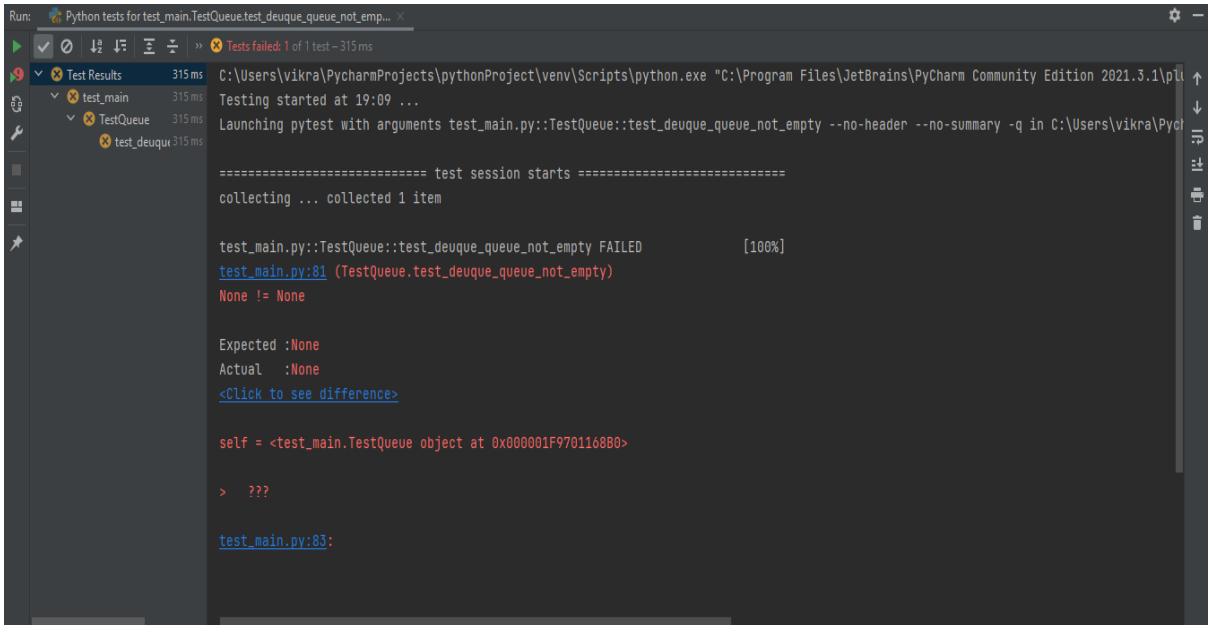
```
# test deque doesn't return None when queue is non-empty
@given(vals)  # This test fails with vals even the expected and actual
answer are same so I dont know
# the reason but passes with special_vals and more_vals)
def test_deuque_queue_not_empty(self, a):
    queue = main.Queue()
```

```
    for x in a:
        queue.enqueue(x)
    results_dequeue = queue.dequeue()
    assert results_dequeue is not None
```

I think it's a failure because the expected output and the actual output are the same and it's not a fault because the code I am using works fine with special_vals and more_vals strategies and even with the less amount of manual integer values and obviously I am not getting any error because the expected and actual output is same.

I am not really sure about the reason for this failure because the only difference in all those strategies is the limits on the st.list which is implemented using min_size and max_size which are the parameter for the list strategy in the hypothesis library and only describe the limit between the number of the lowest and highest element.

I think the reason is that when we set minimum element and maximum element limit in our list strategy there is always at least one element to deque for the deque function, but when we use it without any limits there is an instance in the beginning that queue has no element to deque so that why the test case fails and the return values is None because there is no element in the queue and result after deque all the element is also None, so I think that's why we are getting same expected and actual output but the test case fails.




**Reference**

**[1] https://hypothesis.readthedocs.io/en/latest/quickstart.html**

**[2] https://github.com/ms5589/Queue-implementation-and-Testing**