Blackbox testing

https://www.udacity.com/course/software-testing--cs258

Problem

- Break 5 incorrect implementations of the fixed-size queue using assert statements (i.e. catch buggy queue implementations)
- The queue class is a fixed-size queue of integers. There are 4 methods that we're going to need to concern ourselves with and these method calls are all we're going to have access to during testing (we don't actually know the code that we're running against, we don't have access to it).

```
# the Queue class provides a fized-size FIFO queue of integers

# the constructor takes a single parameter: an integer >0 that
# is the maximum number of elements the queue can hold

# empty() returns True iff the queue currently
# holds no elements, and False otherwise.

# full() returns True iff the queue cannot hold
# any more elements, and False otherwise.

# enqueue(i) attempts to put the integer i into the queue; it returns
# True if successful and False if the queue is full

# dequeue() removes an integer from the queue and returns it,
# or else returns None if the queue is empty

# Example:
# q = Queue(1)
# is_empty = q.empty()
# succeeded = q.enqueue(10)
# is_full = q.full()
# value = q.dequeue()
#
# 1. Should create a Queue q that can only hold 1 element
# 2. Should then check whether q is empty, which should return True
# 3. Should attempt to put 10 into the queue, and return True
# 4. Should check whether q is now full, which should return True
# 5. Should attempt to dequeue and put the result into value, which
#    should be 10
#
# Your test function should run assertion checks and throw an
# AssertionError for each of the 5 incorrect Queues.


def correct_test():
    ###Your code here.
```

Queue1

```
import array


######
###### this queue is buggy: silently holds 2 byte integers only
######

class Queue1:
    def __init__(self,size_max):
        assert size_max > 0
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.size == 0

    def full(self):
        return self.size == self.max

    def enqueue(self, x):
        if self.size == self.max:
            return False
        x = x % (2**16)  # only stores integers up to 2^16
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x
```

Queue2

```
######
###### this queue is buggy: it silently fails to create queues with
###### more than 15 elements
######

class Queue2:
    def __init__(self,size_max):
        assert size_max > 0
        if size_max > 15:  # max_size is set to 15 elements
            size_max = 15
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.size == 0

    def full(self):
        return self.size == self.max

    def enqueue(self, x):
        if self.size == self.max:
            return False
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x
```

Queue3

```
######
###### this queue is buggy: its empty() method is implemented by seeing
###### if an element can be reached
######

class Queue3:
    def __init__(self,size_max):
        assert size_max > 0
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.dequeue() is None  # tricky; trying to dequeue an element
and checking if is None

    def full(self):
        return self.size == self.max

    def enqueue(self, x):
        if self.size == self.max:
            return False
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x
```

Queue4

```
######
###### this queue is buggy: dequeue of empty returns False instead of None
######

class Queue4:
    def __init__(self,size_max):
        assert size_max > 0
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.size == 0

    def full(self):
        return self.size == self.max

    def enqueue(self, x):
        if self.size == self.max:
            return False
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return False  # returns False where it should, according to the
specification return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x
```

Queue5

```
######
###### this queue is buggy: it holds one less item than intended
######

class Queue5:
    def __init__(self,size_max):
        assert size_max > 0
        size_max -= 1  # holds one less element than you tell it to hold
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.size == 0

    def full(self):
        return self.size == self.max

    def enqueue(self, x):
        if self.size == self.max:
            return False
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x
```

Tests

```
def correct_test():
    # Queue1 silently holds only 2 byte unsigned integers, than wraps around
    q = Queue1(2)
    succeeded = q.enqueue(100000)  # value greater than 2^16
    assert succeeded
    value = q.dequeue()
    assert value == 100000  # test1 failed

    # Queue2 silently fails to hold more than 15 elements
    q = Queue2(30)
    # try to enqueue more than 15 elements
    for i in range(20):
        succeeded = q.enqueue(i)
        assert succeeded  # test2 failed

    # Queue3 implements empty() by checking if dequeue() succeeds.
    # This changes the state of the queue unintentionally.
    q = Queue3(2)
    succeeded = q.enqueue(10)
    assert succeeded
    assert not q.empty()  # the function checks by trying to dequeue
    value = q.dequeue()
    assert value == 10  # test3 failed

    # Queue4 dequeue() of an empty queue returns False instead of None
    q = Queue4(2)
    value = q.dequeue()
    assert value is None  # test4 failed

    # Queue5 holds one less item than intended
    q = Queue5(2)
    for i in range(2):
        succeeded = q.enqueue(i)
        assert succeeded  # test4 failed


correct_test()
```