Lab 4: Abstract Data Type Solution

3) Running timing experiments

1. Your first task is to open *timequeue.py* and follow the instructions contained within it to complete the timing experiment.

```
 ...
 def _setup_queues(qsize: int, n: int) -> List[Queue]:
     """Return a list of <n> queues, each of the given size."""
     # Experiment preparation: make a list containing <n> queues,
     # each of size <qsize>.
     # You can "cheat" here and set your queue's _items attribute directly
     # to a list of the appropriate size by writing something like
     #
     #     queue._items = list(range(qsize))
     #
     # to save a bit of time in setting up the experiment.

     queue_list = []

     i = 0
     while i < n:
         queue = Queue()
         queue._items = list(range(qsize))

         queue_list.append(queue)
         i += 1

     return queue_list


 def time_queue() -> None:
     """Run timing experiments for Queue.enqueue and Queue.dequeue."""
     # The queue sizes to try.
     # You can change these values if you'd like.
     queue_sizes = [10000, 20000, 40000, 80000, 160000]

     # The number of times to call a single enqueue or dequeue operation.
     trials = 200
```

```python

        # This loop runs the timing experiment. It has three main steps:
        for queue_size in queue_sizes:
            #   1. Initialize the sample queues.
            queues = _setup_queues(queue_size, trials)

            #   2. For each one, calling the function "timeit", takes three
            #      arguments:
            #        - a *string* representation of a piece of code to run
            #        - the number of times to run it (just 1 for us)
            #        - globals is a technical argument that you DON'T need to
            #          care about
            time = 0
            for queue in queues:
                time += timeit('queue.enqueue(1)', number=1, globals=locals())

            #   3. Report the total time taken to do an enqueue on each queue.
            print(f'enqueue: Queue size {queue_size:>7}, time {time}')

        # TODO: using the above loop as an analogy, write a second timing
        # experiment here that runs dequeue on the given queues, and reports the
        # time taken. Note that you can reuse most of the same code.
        for queue_size in queue_sizes:
            queues = _setup_queues(queue_size, trials)

            time = 0
            for queue in queues:
                time += timeit('queue.dequeue()', number=1, globals=locals())

            #   3. Report the total time taken to do an enqueue on each queue.
            print(f'dequeue: Queue size {queue_size:>7}, time {time}')
    ...

```

Listing 1: task_3_q1_solution.py

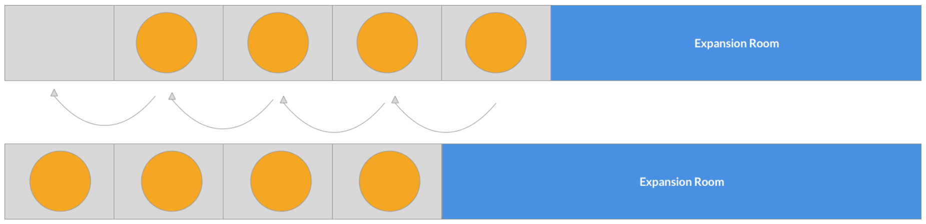
2. After you've run your experiment, you should notice that your two queue operations *enqueue* and *dequeue* behave quite differently.

   While one seems to take the same amount of time no matter how many items are in the queue, the other takes longer and longer as the number of items are in the queue.

   Compare your notes with other groups. Which end of a Python list seems to be the "slow" end? Do you have a guess as to why this might be the case? (If you don't: don't worry! You'll learn about this in later weeks.)

- The front end of a Python list is the slow end.

  This is because when *LIST.pop(0)* is used, all of the later elements need to be shifted to the left by 1 position.


Expansion Room
Expansion Room