

Create a Circular Queue

In this challenge you will be creating a Circular Queue. A circular queue is a queue that writes to the end of a collection then begins overwriting itself at the beginning of the collection. This type of data structure is useful in certain situations. For example, a circular queue can be used for streaming media. Once the queue is full, new media data will overwrite old data.

A good way to illustrate this concept is with an array of length 5:

```
[null, null, null, null, null]
```

```
^Read @ 0
```

```
^Write @ 0
```

Here the read and write are both at position 0. Now the queue gets 3 new records a, b, and c. Our queue now looks like:

```
[a, b, c, null, null]
```

```
^Read @ 0
```

```
^Write @ 3
```

As the read head reads, it can remove values or keep them:

```
[null, null, null, null, null]
```

```
^Read @ 3
```

```
^Write @ 3
```

Now we write the values d, e, and f to the queue. Once the write reaches the end of the array it loops back to the beginning:

```
[f, null, null, d, e]
```

```
^Read @ 3
```

```
^Write @ 1
```






This approach requires a constant amount of memory but allows files of a much larger size to be processed.

In this challenge we will implement a circular queue. The circular queue should provide `enqueue` and `dequeue` methods which allow you to read from and write to the queue. The class itself should also accept an integer argument which you can use to specify the size of the queue when created. We've written the starting version of this class for you in the code editor.

When you enqueue items to the queue, the write pointer should advance forward and loop back to the beginning once it reaches the end of the queue. The `enqueue` method should return the item you enqueued if it is successful; otherwise it will return `null`.

Likewise, the read pointer should advance forward as you dequeue items. When you dequeue an item, that item should be returned. If you cannot dequeue an item, you should return `null`.

The write pointer should not be allowed to move past the read pointer (our class won't let you overwrite data you haven't read yet) and the read pointer should not be able to advance past data you have written.

	The <code>enqueue</code> method should add items to the circular queue.
	You should not enqueue items past the read pointer.
	The <code>dequeue</code> method should dequeue items from the queue.
	After an item is dequeued, its position in the queue should be reset to <code>null</code> .
	Trying to dequeue past the write pointer should return <code>null</code> and does not advance the write pointer.