Queue Using Array

**Implement a queue using an array**

In this notebook, we'll look at one way to implement a queue by using an array. First, check out the walkthrough for an overview of the concepts, and then we'll take a look at the code.

Walkthrough

OK, so those are the characteristics of a queue, but how would we implement those characteristics using an array?

Walkthrough

What happens when we run out of space in the array? This is one of the trickier things we'll need to handle with our code.

Walkthrough

**Functionality**

Once implemented, our queue will need to have the following functionality:

1. enqueue - adds data to the back of the queue
2. dequeue - removes data from the front of the queue
3. front - returns the element at the front of the queue
4. size - returns the number of elements present in the queue
5. is\_empty - returns True if there are no elements in the queue, and False otherwise
6. \_handle\_full\_capacity - increases the capacity of the array, for cases in which the queue would otherwise overflow

Also, if the queue is empty, dequeue and front operations should return None.

**1. Create the queue class and its \_\_init\_\_ method**

First, have a look at the walkthrough:

Walkthrough

In [1]:



**class** Queue:

**def** \_\_init\_\_(self):

self.arr**=**[0,0,0,0,0,0,0,0,0,0]

self.next\_index**=**0

self.front\_index**=-**1

self.queue\_size**=**0

Now give it a try for yourself. In the cell below:

* Define a class named Queue and add the \_\_init\_\_ method
* Initialize the arr attribute with an array containing 10 elements, like this: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
* Initialize the next\_index attribute
* Initialize the front\_index attribute
* Initialize the queue\_size attribute

In [ ]:



​

Show Solution

Let's check that the array is being initialized correctly. We can create a Queue object and access the arr attribute, and we should see our ten-element array:

In [2]:



q **=** Queue()

print(q.arr)

print("Pass" **if** q.arr **==** [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] **else** "Fail")

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Pass

**2. Add the enqueue method**

Walkthrough

In [ ]:



​

In the cell below, add the code for the enqueue method.

The method should:

* Take a value as input and assign this value to the next free slot in the array
* Increment queue\_size
* Increment next\_index (this is where you'll need to use the modulo operator %)
* If the front index is -1 (because the queue was empty), it should set the front index to 0

In [ ]:



**class** Queue:

​

**def** \_\_init\_\_(self, initial\_size**=**10):

self.arr **=** [0 **for** \_ **in** range(initial\_size)]

self.next\_index **=** 0

self.front\_index **=** **-**1

self.queue\_size **=** 0

​

**def** enqueue(self,data):

self.arr[self.next\_index]**=**data

self.queue\_size**+=**1

self.next\_index **=** (self.next\_index **+** 1) **%** len(self.arr)

**if** self.front\_index **==** **-**1:

self.front\_index **=** 0

Show Solution

**3. Add the size, is\_empty, and front methods**

Just like with stacks, we need methods to keep track of the size of the queue and whether it is empty. We can also add a front method that returns the value of the front element.

* Add a size method that returns the current size of the queue
* Add an is\_empty method that returns True if the queue is empty and False otherwise
* Add a front method that returns the value for the front element (whatever item is located at the front\_index position). If the queue is empty, the front method should return None.

In [ ]:



**class** Queue:

​

**def** \_\_init\_\_(self, initial\_size**=**10):

self.arr **=** [0 **for** \_ **in** range(initial\_size)]

self.next\_index **=** 0

self.front\_index **=** **-**1

self.queue\_size **=** 0

​

**def** enqueue(self, value):

*# enqueue new element*

self.arr[self.next\_index] **=** value

self.queue\_size **+=** 1

self.next\_index **=** (self.next\_index **+** 1) **%** len(self.arr)

**if** self.front\_index **==** **-**1:

self.front\_index **=** 0

**def** size(self):

**return** self.queue\_size

**def** is\_empty(self):

**return** self.queue\_size**==**0

​

**def** front(self):

**if** self.is\_empty():

**return** **None**

**return** self.arr[self.front\_index]

Show Solution

**4. Add the dequeue method**

Walkthrough

In [ ]:



​

In the cell below, see if you can add the deqeueue method.

Here's what it should do:

* If the queue is empty, reset the front\_index and next\_index and then simply return None. Otherwise...
* Get the value from the front of the queue and store this in a local variable (to return later)
* Shift the head over so that it refers to the next index
* Update the queue\_size attribute
* Return the value that was dequeued

In [ ]:



**class** Queue:

​

**def** \_\_init\_\_(self, initial\_size**=**10):

self.arr **=** [0 **for** \_ **in** range(initial\_size)]

self.next\_index **=** 0

self.front\_index **=** **-**1

self.queue\_size **=** 0

​

**def** enqueue(self, value):

*# enqueue new element*

self.arr[self.next\_index] **=** value

self.queue\_size **+=** 1

self.next\_index **=** (self.next\_index **+** 1) **%** len(self.arr)

**if** self.front\_index **==** **-**1:

self.front\_index **=** 0

**def** dequeue(self):

**if** self.is\_empty():

self.front\_index **=** **-**1 *# resetting pointers*

self.next\_index **=** 0

**return** **None**

​

*# dequeue front element*

value **=** self.arr[self.front\_index]

self.front\_index **=** (self.front\_index **+** 1) **%** len(self.arr)

self.queue\_size **-=** 1

**return** value

​

**def** size(self):

**return** self.queue\_size

​

**def** is\_empty(self):

**return** self.size() **==** 0

**def** front(self):

*# check if queue is empty*

**if** self.is\_empty():

**return** **None**

**return** self.arr[self.front\_index]

Show Solution

**5. Add the \_handle\_queue\_capacity\_full method**

Walkthrough

In [ ]:



​

First, define the \_handle\_queue\_capacity\_full method:

* Define an old\_arr variable and assign the the current (full) array so that we have a copy of it
* Create a new (larger) array and assign it to arr.
* Iterate over the values in the old array and copy them to the new array. Remember that you'll need two for loops for this.

Then, in the enqueue method:

* Add a conditional to check if the queue is full; if it is, call \_handle\_queue\_capacity\_full

In [3]:



**class** Queue:

​

**def** \_\_init\_\_(self, initial\_size**=**10):

self.arr **=** [0 **for** \_ **in** range(initial\_size)]

self.next\_index **=** 0

self.front\_index **=** **-**1

self.queue\_size **=** 0

​

**def** enqueue(self, value):

*# TODO: Check if the queue is full; if it is, call the \_handle\_queue\_capacity\_full method*

​

*# enqueue new element*

self.arr[self.next\_index] **=** value

self.queue\_size **+=** 1

self.next\_index **=** (self.next\_index **+** 1) **%** len(self.arr)

**if** self.front\_index **==** **-**1:

self.front\_index **=** 0

​

**def** dequeue(self):

*# check if queue is empty*

**if** self.is\_empty():

self.front\_index **=** **-**1 *# resetting pointers*

self.next\_index **=** 0

**return** **None**

​

*# dequeue front element*

value **=** self.arr[self.front\_index]

self.front\_index **=** (self.front\_index **+** 1) **%** len(self.arr)

self.queue\_size **-=** 1

**return** value

​

**def** size(self):

**return** self.queue\_size

​

**def** is\_empty(self):

**return** self.size() **==** 0

**def** front(self):

*# check if queue is empty*

**if** self.is\_empty():

**return** **None**

**return** self.arr[self.front\_index]

​

**def** \_handle\_queue\_capacity\_full(self):

old\_arr **=** self.arr

self.arr **=** [0 **for** \_ **in** range(2 **\*** len(old\_arr))]

​

index **=** 0

​

*# copy all elements from front of queue (front-index) until end*

**for** i **in** range(self.front\_index, len(old\_arr)):

self.arr[index] **=** old\_arr[i]

index **+=** 1

​

*# case: when front-index is ahead of next index*

**for** i **in** range(0, self.front\_index):

self.arr[index] **=** old\_arr[i]

index **+=** 1

​

*# reset pointers*

self.front\_index **=** 0

self.next\_index **=** index

Show Solution

**Test your queue**

In [4]:



*# Setup*

q **=** Queue()

q.enqueue(1)

q.enqueue(2)

q.enqueue(3)

​

*# Test size*

print ("Pass" **if** (q.size() **==** 3) **else** "Fail")

​

*# Test dequeue*

print ("Pass" **if** (q.dequeue() **==** 1) **else** "Fail")

​

*# Test enqueue*

q.enqueue(4)

print ("Pass" **if** (q.dequeue() **==** 2) **else** "Fail")

print ("Pass" **if** (q.dequeue() **==** 3) **else** "Fail")

print ("Pass" **if** (q.dequeue() **==** 4) **else** "Fail")

q.enqueue(5)

print ("Pass" **if** (q.size() **==** 1) **else** "Fail")

Pass

Pass

Pass

Pass

Pass

Pass