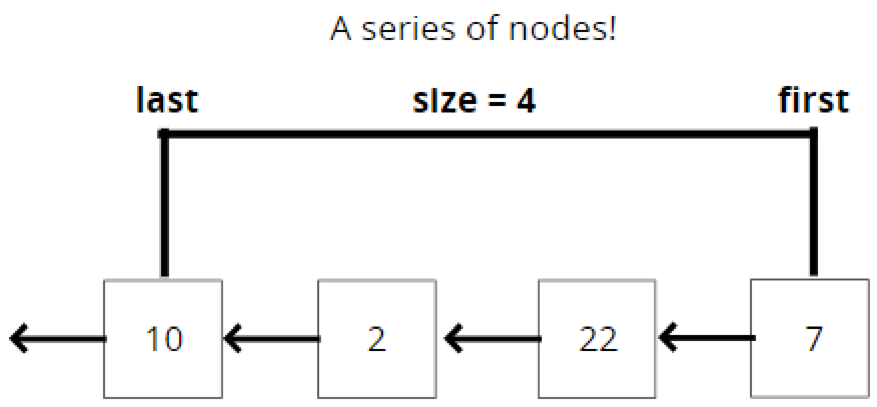
Queues

Just like stacks, queue is a data structure that you add data in, and you remove data out.

Just the order is different.

FIFO data structure

The first element added to the queue will be the first element removed from the queue.



Think of a line of people, a queue in a bank. The first person in line is the first person out.

How do we use them in programming?

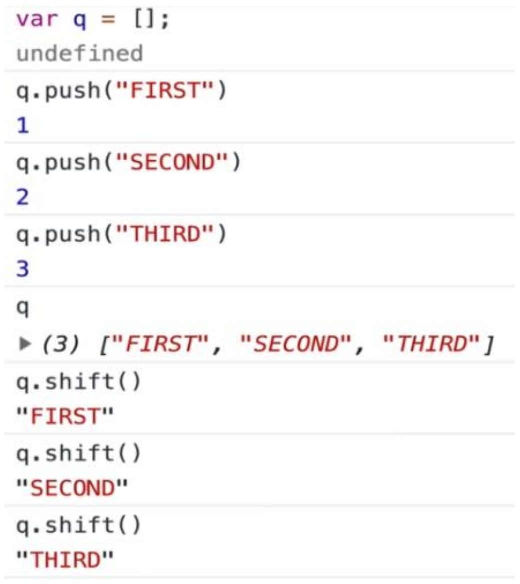
- ➔ Online game where players are waiting to enter in. So, there's some queue structure keeping track of who's been waiting the longest. He'll be the first one to enter into the game.
- ➔ Background tasks
- ➔ Uploading resources (if they're of same size)
- ➔ Printing/Task processing
- ➔ Wherever you need to keep track and maintain an order.

Building a queue with an array

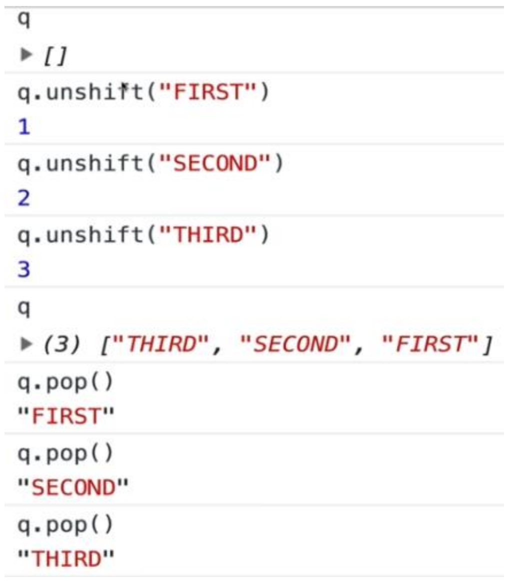
Two ways:

1. Add to the beginning and remove from the end. (unsihft & pop)
2. Add to the end and remove from the beginning. (push & shift)

Both are inefficient as both either involves shift or unshift, which required reindexing.



```
var q = [];
undefined
q.push("FIRST")
1
q.push("SECOND")
2
q.push("THIRD")
3
q
▶ (3) ["FIRST", "SECOND", "THIRD"]
q.shift()
"FIRST"
q.shift()
"SECOND"
q.shift()
"THIRD"
```



```
q
▶ []
q.unshift("FIRST")
1
q.unshift("SECOND")
2
q.unshift("THIRD")
3
q
▶ (3) ["THIRD", "SECOND", "FIRST"]
q.pop()
"FIRST"
q.pop()
"SECOND"
q.pop()
"THIRD"
```

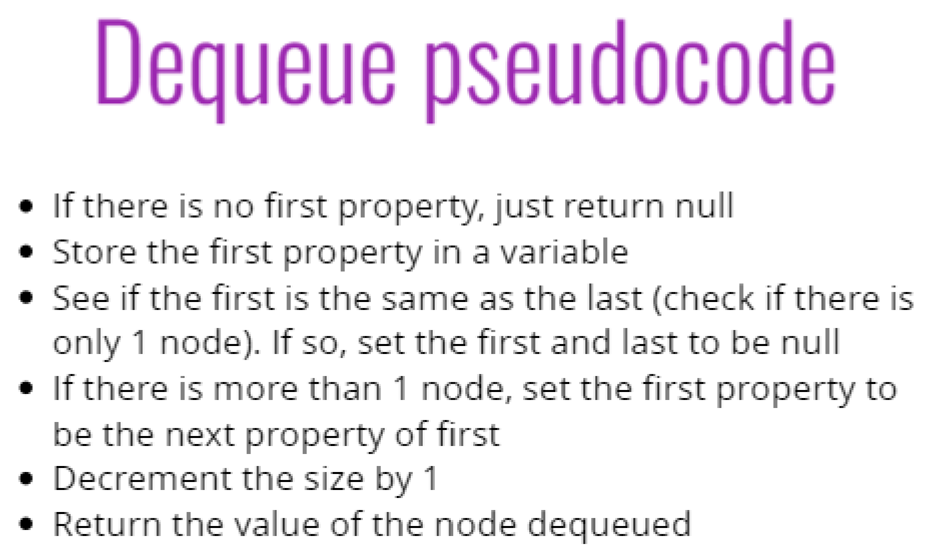
```
class Node {
  constructor(value) {
    this.val = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.first = null;
    this.last = null;
    this.size = 0;
  }
}
```

Enqueue Pseudocode

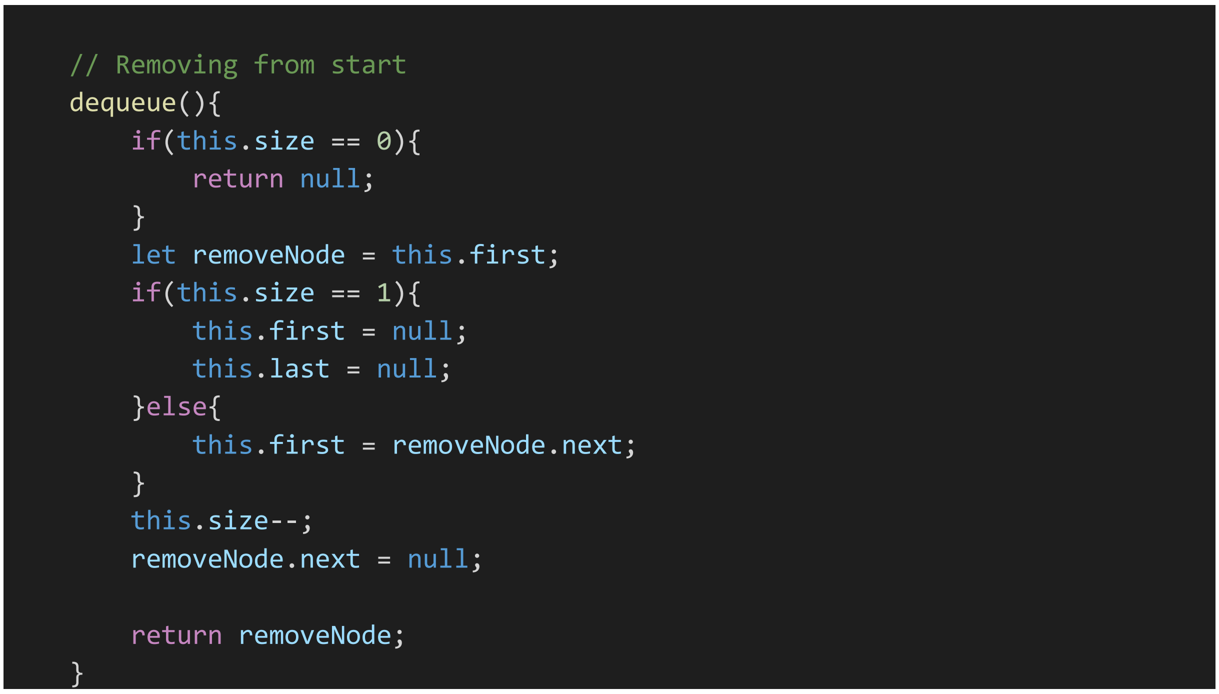
- This function accepts some value
- Create a new node using that value passed to the function
- If there are no nodes in the queue, set this node to be the first and last property of the queue
- Otherwise, set the next property on the current last to be that node, and then set the last property of the queue to be that node
- Increment the size of the queue by 1

```
// Adding at end
enqueue(val){
    let newNode = new Node(val);
    if(this.size == 0){
        this.first = newNode;
        this.last = newNode;
    }else{
        let prevNode = this.last;
        this.last = newNode;
        prevNode.next = newNode;
    }
    return ++this.size
}
```



Dequeue pseudocode

- If there is no first property, just return null
- Store the first property in a variable
- See if the first is the same as the last (check if there is only 1 node). If so, set the first and last to be null
- If there is more than 1 node, set the first property to be the next property of first
- Decrement the size by 1
- Return the value of the node dequeued



```
// Removing from start
dequeue(){
    if(this.size == 0){
        return null;
    }
    let removeNode = this.first;
    if(this.size == 1){
        this.first = null;
        this.last = null;
    }else{
        this.first = removeNode.next;
    }
    this.size--;
    removeNode.next = null;

    return removeNode;
}
```

Time Complexity:

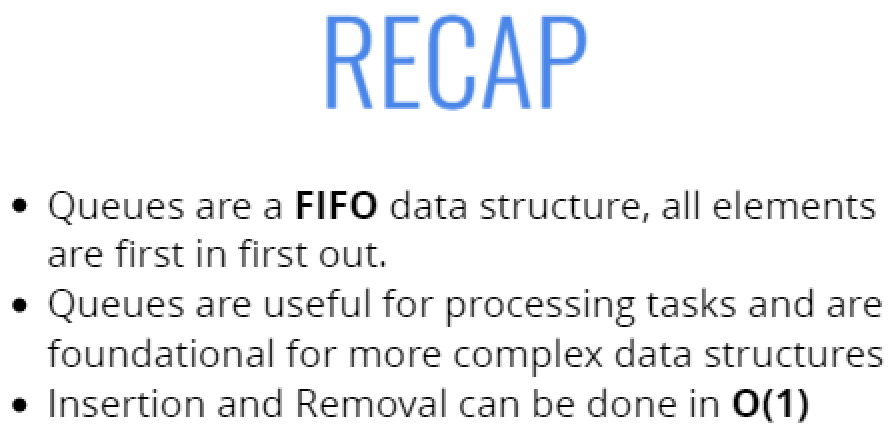
Insertion	O(1)
Removal	O(1)
Accessing	O(n)
Searching	O(n)

What actually matters are the first two (insertion and removal)

Cuz the way we wrote our queue class was to make sure that enqueue and enqueue were both constant time.

Note that in array, it was not constant both scenarios.

Searching and Accessing something isn't something that you'd do with a queue otherwise you may not wanna use queue data structure if you need to prioritize searching.



RECAP

- Queues are a **FIFO** data structure, all elements are first in first out.
- Queues are useful for processing tasks and are foundational for more complex data structures
- Insertion and Removal can be done in **O(1)**