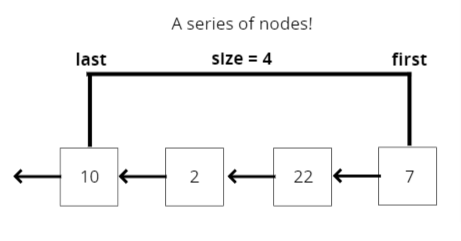
**Stacks & Queues**

Both are data collections. They’re an abstract Data Structure.

**STACKS**



**WHAT IS A STACK?**

A LIFO (Last In First Out) data structure!

The last element added to the stack will be the first element removed from the stack.

It’s basically a set of rules that says, okay, we need to be able to store data in such a way that the first things added in, is the last things removed, the last things added in is the first things removed.

It’s just a concept.

**The Call Stack!:**

WHERE STACKS ARE USED FOR,

* Managing function invocations
* Managing function invocations
* Undo / Redo
* Routing (the history object) is treated like a stack!

There’s more than one way of doing it, and the easiest way is to use an array.

**Array Implementation:**

let stack = [];

stack.push();  
stack.pop();

or

stack.unshift();  
stack.shift();

Both Approaches are valid, since both follow LIFO principle. But the first approach is more efficient as per Big-O Notation.

**Singly Linked Lists Implementation:**

Since, pop() takes O(n) in Singly Linked List. So we’re going to use unshift() (for pushing Node from the Beginning ) & shift() (for popping Node for the Beginning). Although we’re free to use Doubly Linked List and it doesn’t matter what method we use for adding & removing Element from Start or End. But it should follow LIFO principle.

**PUSHING**

Add a value to the top of the stack!

**PUSHING PSEUDOCODE**

* The function should accept a value
* Create a new node with that value
* If there are no nodes in the stack, set the first and last property to be the newly created node
* If there is at least one node, create a variable that stores the current first property on the stack
* Reset the first property to be the newly created node
* Set the next property on the node to be the previously created variable
* Increment the size of the stack by 1

**POP**

Remove a value from the top of the stack!

**POP PSEUDOCODE**

* If there are no nodes in the stack, return null
* Create a temporary variable to store the first property on the stack
* If there is only 1 node, set the first and last property to be null
* If there is more than one node, set the first property to be the next property on the current first
* Decrement the size by 1
* Return the value of the node removed

**Stack Pushing & Poping Code:**

class Node {

constructor(val) {

this.value = val;

this.next = null;

}

}

**// Should Follow Last In Last Out Principle**

class Stack {

constructor() {

this.first = null;

this.last = null;

this.size = 0;

}

**//Should work like unshift** push(val){

let newNode = new Node(val);

if(!this.first){

this.first = newNode;

this.last = newNode;

}else{

let temp = this.first;

this.first = newNode;

newNode.next = temp;

}

return ++list.size;

}

**//Should work like shift**

pop(){

if(!this.first) return null;

let removedNode = this.first;

if(this.first === this.last){

this.first = null;

this.last = null;

}else{

this.first = removedNode.next;

removedNode.next = null;

}

this.size--;

return removedNode.value;

}

}

let stack = new Stack();

stack.push(1);  
stack.push(2);  
stack.push(3);  
stack.push(4);  
stack.push(5);

**BIG O of STACKS**

Insertion - O(1)

Removal - O(1)

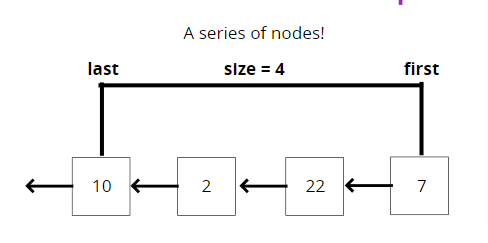
Searching - O(N)

Access - O(N)

**RECAP**

* Stacks are a LIFO data structure where the last value in is always the first one out.
* Stacks are used to handle function invocations (the call stack), for operations like undo/redo, and for routing (remember pages you have visited and go back/forward) and much more!
* They are not a built in data structure in JavaScript, but are relatively simple to implement
* Insert and remove are both O(1)

**QUEUES**



**WHAT IS A QUEUE?**

A **FIFO**(First In First Out) data structure!. Queues exist everywhere!

**How do we use them in programming?**

* Background tasks
* Uploading resources
* Printing / Task processing

**Array Implementation:**

let stack = [];

stack.push();  
stack.shift();

or

stack.unshift();  
stack.pop();

Both Approaches are valid, since both follow LIFO principle. But both approach is more efficient as per Big-O Notation.

**Singly Linked List Implementation:**

**Enqueue**

Adding to the beginning of the Queue!

**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

**Dequeue**

Removing from the beginning of the Queue!

**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

**Queue Enqueue & Dequeue Code:**

class Node {

constructor(val) {

this.value = val;

this.next = null;

}

}

**// Should Follow First In First Out Principle**

class Queue {

constructor() {

this.first = null;

this.last = null;

this.size = 0;

}

**//Should work like push**

enqueue(val){

let newNode = new Node(val);

if(!this.first){

this.first = newNode;

this.last = newNode;

}else{

this.last.next = newNode;

this.last = newNode;

}

this.size++;

return this;

}

**//Should work like shift**

dequeue(){

if(!this.first) return null;

let dequeueNode = this.first;

if(this.first === this.last){

this.first = null;

this.last = null;

}else{

this.first = dequeueNode.next;

dequeueNode.next = null;

}

this.size--;

return dequeueNode.value;

}

}

let queue = new Queue();

queue.enqueue(1);  
queue.enqueue(2);  
queue.enqueue(3);  
queue.enqueue(4);  
queue.enqueue(5);

**BIG O of QUEUES**

Insertion - O(1)

Removal - O(1)

Searching - O(N)

Access - O(N)

**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)