STACKS

OBJECTIVES

- Define what a stack is
- Understand use cases for a stack
- Implement operations on a stack data structure

WHAT IS A STACK?

A **LIFO** data structure!

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

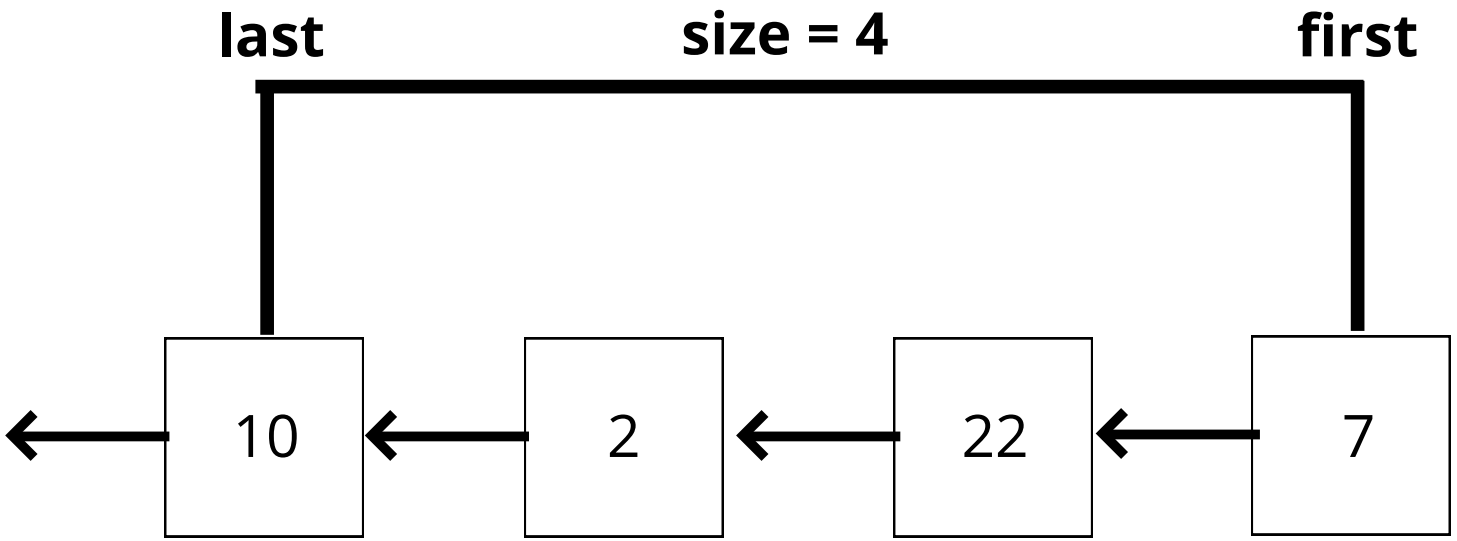
HOW IS IT USED?

Think about a stack of **plates**, or a stack of **markers**, or a stack of....**anything**.

As you pile it up the last thing (or the topmost thing) is what gets removed first.

How we'll visualize a stack

A series of nodes!



Let's see this in action!

WE'VE SEEN THIS BEFORE

The Call Stack!

WHERE STACKS ARE USED

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

THERE IS MORE THAN ONE WAY OF IMPLEMENTING A STACK

ARRAY IMPLEMENTATION

LINKED LIST IMPLEMENTATION

A STACK CLASS

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

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

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

YOUR TURN

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

YOUR TURN

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)**