Stacks

It’s a data structure that you add data in, and you remove data out.

LIFO data structure

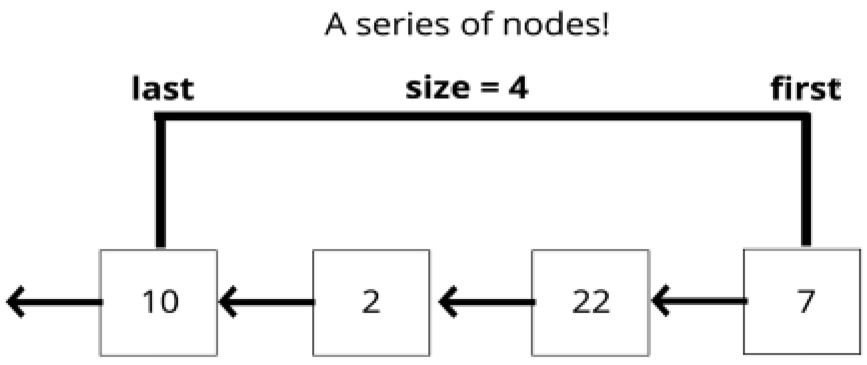
The last element added to the stack will be the first element removed from the stack.

Think about a stack of plates. As you pile it up, the last thing is what gets removed first.

There ars more than one way of implementing a stack.

Stack is just a concept.

You could use a linked list:



Remember call stack, it is based of stack data structure.

Where stacks are used:

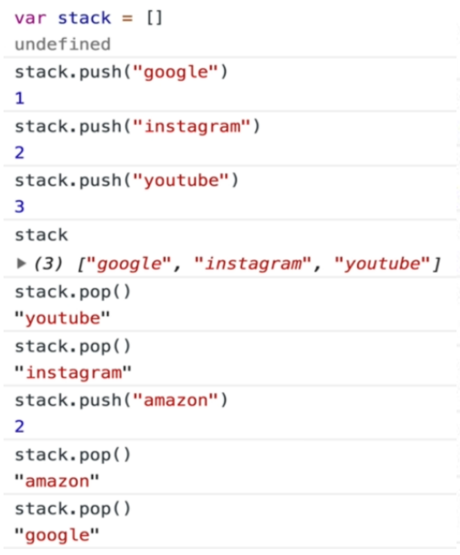
- ➔ Managing function invocations
- ➔ Undo/Redo
- ➔ Routing(the history object in browser)

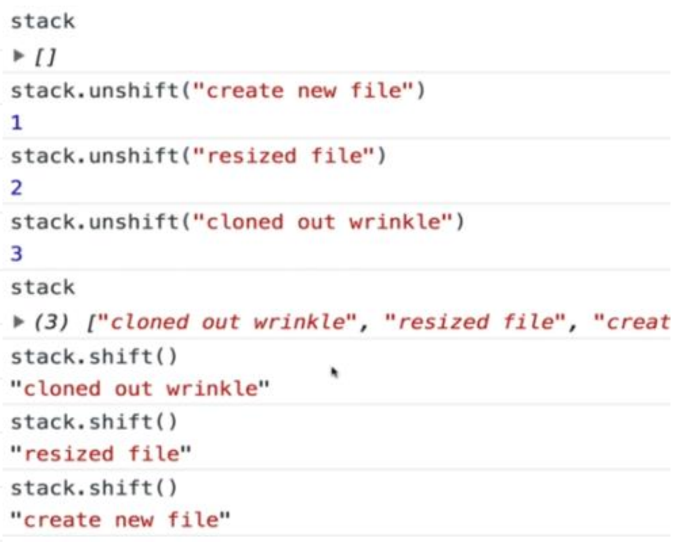
Building a stack with an array

Two ways:

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

So using push and pop in tandem or using shift and unshift together on an array create a stack. 2st way is better as it’s more efficient to add and remove from the end.


```
var stack = []
undefined
stack.push("google")
1
stack.push("instagram")
2
stack.push("youtube")
3
stack
▶ (3) ["google", "instagram", "youtube"]
stack.pop()
"youtube"
stack.pop()
"instagram"
stack.push("amazon")
2
stack.pop()
"amazon"
stack.pop()
"google"
```


```
stack
▶ []
stack.unshift("create new file")
1
stack.unshift("resized file")
2
stack.unshift("cloned out wrinkle")
3
stack
▶ (3) ["cloned out wrinkle", "resized file", "creat
stack.shift()
"cloned out wrinkle"
stack.shift()
"resized file"
stack.shift()
"create new file"
```

Both of these approaches are valid stacks. It does'nt matter what direction we're doing it in as long as we're removing from the same direction from where we added. All that matters is that we have a way of adding data in and removing data such that it satisfies LIFO principle.

But there's a disticntion between these two.

Push-pop	adding to the end and removing from the end
Unshift-shift	adding to the start and removing from the start

In case of efficiency, adding and removing from from the beginning is not good.

So using push-pop is better (in arrays).

If we care about efficiency, then we don't want to use an array for a stack.

All you need is the last in first out capability, we'll use linked list as there's no need to have this indecies as we're not accessing information based off of an index but of when it was inserted.

The order needs to be preserved only and a linked list will do that for us.

We've already build push and pop methods in linked list so why can't we use the same methods here?

Because in stacks, push and pop are supposed to be in constant time.

Ok so as in arrays adding or removing element from start is not efficient. So we added and removed from the end.

In singly linked lists removing from the end is not efficient and as it is our requirement that we have to apply add and remove from same side. So we'll be adding and removing from start of the linked list.

So as we want *Last In First Out*, so in our case:

Push	→	adding node at start
Pop	→	removing node from start

Same as unshift and shift. So here you could rename shift and inshift as pop and push.

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

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

```
  // pushing at start
  push(val) {
    let newNode = new Node(val);
    if (this.size == 0) {
      this.first = newNode;
      this.last = newNode;
    } else {
      let oldNode = this.first;
      this.first = newNode;
      newNode.next = oldNode;
    }
    return ++this.size;
  }
```

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

  return removeNode.val;
}
```

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 stack class was to make sure that pushing and popping were both constant time.

Note that in array, it was constant in one scenario if we add and remove from the end.

If searching and accessing is important then you should use an array.

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