Stacks

Stack is a Last In, First Out data structure useful is solving many problems in computer science.

Introduction

A stack is a list like data structure which allows three basic operations.

1. Insert into the stack. This is known as ***Push***.
2. Extract the most recently added. This is called ***Pop***.
3. Look at the most recently added element without removing (popping) it. This is called ***Peek***.

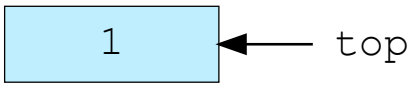
Using the above terminology, we can say that we push elements into the stack and we pop elements from the stack. Stack is known as a ***Last-In First-Out (LIFO)*** data structure.

To support operations in stack, we always keep track of the top most element in the stack. Here's how tracking the ***top*** element in the stack helps us implement the above operations.

1. When we push an element, we update the top to start pointing to the newly added element.
2. When we pop an element, we update the top to point to the next most-recently-added element in the stack.
3. When we peek, we just return the data pointed by the top.

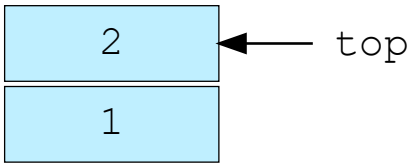
Here's a quick animation where we push and pop elements in the stack.

Push (1)



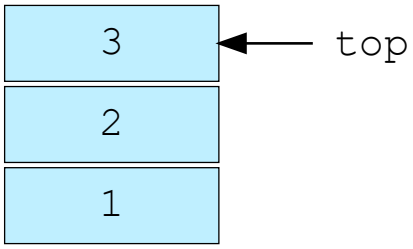
Let's insert the first element in the stack.
Pushing 1 on the stack

Push (2)



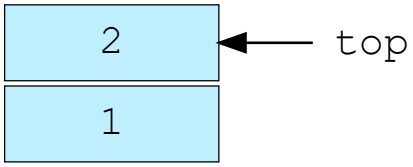
Push 2 on the stack. Look how Top starts pointing to the newly added element.

Push (3)



Element of value 3 is pushed onto the stack.
It now becomes the top most element on the stack

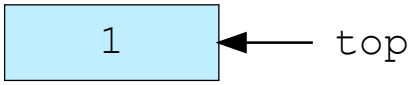
Pop() returns 3



Pop an element from stack. It returns 3.
Top now points to 2.

4 of 5

Pop() returns 2



Pop returns 2. Top starts pointing to 1

5 of 5

Defining the stack

The below block of code shows how a stack would be defined in JavaScript

```
function Stack() {
  this._top = -1;
  this._values = [];
}
```

The Stack function would contain 2 parts.

- The index of the most recent element tracked by *_top*. It is initialized by -1 to indicate that stack contains no elements.

- Array of values inserted into the stack tracked by *_values*. It is initialized by an empty array.

Pushing elements onto the stack

Let's implement the push operation. The below block of code defines how elements can be pushed or added to the stack.

```
Stack.prototype.push = function(data) {
  this._top++;
  this._values[this._top] = data;
};
```

We just increment the top to get the index for new element and then insert the data. Top is automatically pointing to the top-most element in the stack

Popping elements from the stack

Let's look at how can we implement pop operation on our stack. First, we check that stack has any element. If stack is empty, we return null. We then note down the element at the top, then reduce the top and reduce the size of the array. As the last step we return the element that was at the top.

```
Stack.prototype.pop = function() {
  if (this._top < 0) {
    return null;
  }

  var topElement = this._values[this._top];
  this._top--;

  //We could have used array pop method here to remove
  //the last array element but we haven't used it to
  //avoid confusion with Stack's pop method
  this._values.length--;

  return topElement;
};
```

Peeking at the top element

Implementing peek is easy. We just return the element pointed by the top.

Here's the code which is self explanatory.

```
Stack.prototype.peek = function() {
  if (this._top < 0) {
    return null;
  }

  return this._values[this._top];
};
```

Stack in action

Let's run the functions that we just discussed and see how the stack works. We'll push a few elements and then pop them.

```
var stack = new Stack();

for (var i = 100; i <= 300; i+=100) {
  console.log('Pushing on stack: ' + i);
  stack.push(i);
}

console.log('Top of the stack using Peek: ' + stack.peek());
console.log('Pop from stack. Popped element = ' + stack.pop());
console.log('Pop from stack. Popped element = ' + stack.pop());
```

Let's have a quick quiz

Stack - Basics

1 What are the two most common operations on a stack?

2 Both return the top value of stack. Is Peek Operation different from Pop Operation?

Check Answers

Exercise 1

As a first exercise, let's implement a method size that returns the number of elements in the stack.

> Some tests are failing right now. Implement 'size' to fix them.

JavaScript

HTML

CSS (SCSS)

```
Stack.prototype.size = function() {
  ///// TODO. Implement this /////
  return undefined;
}

// This is the function that runs test cases.
runEvaluation();
```

Console

Clear

```
*** There is some bug lurking there. See failed test cases ***

Here are the tests that ran:

Test case FAILED for stack.size(). Result: undefined. Expected: 0
```

```
Test case FAILED for stack.size(). Result: undefined. Expected: 2
```

```
Test case FAILED for stack.size(). Result: undefined. Expected: 1
```

Exercise 2

As a second exercise, let's implement isEmpty method. It returns true when stack is empty and returns false when the stack has one or more elements.

> Some tests are failing right now. Implement 'isEmpty' to fix them.

JavaScript

HTML

CSS (SCSS)

```
Stack.prototype.isEmpty = function() {
  ///// TODO. Implement this /////
  // Return true when empty //
  // Return false when not empty
  return undefined;
}

// This is the function that runs test cases.
runEvaluation();
```

Console

Clear

```
*** There is some bug lurking there. See failed test cases ***
```

```
Here are the tests that ran:
```

```
Test case FAILED for stack.isEmpty(). Result: undefined. Expected: true
```

```
Test case FAILED for stack.isEmpty(). Result: undefined. Expected: false
```

```
Test case FAILED for stack.isEmpty(). Result: undefined. Expected: true
```

Does Javascript has a built-in Stack?

Yes. In fact, the Array in Javascript acts as a stack and already has the appropriate methods to use it like stack. Look at the below code.

```
var stack = [];
console.log('Push 100 in stack');
stack.push(100);
console.log('Popped from Stack: ' + stack.pop());
```

Summary

- Stack is a LIFO data structure
- Stack has 3 basic operations: Push, Pop and Peek.
- Javascript Array is actually a stack and supports Push and Pop.