Lecture 3.2

JavaScript Arrays

Topics

- Intro to Data Structures
  - Stack
  - Queue

Intro to Data Structures

Stack Data Structure

The name *stack* comes from the analogy to a set of physical items e.g., DVD disc, books, stacked on top each other.

A stack is a data structure that holds a list of elements. A stack works based on the LIFO principle i.e., Last In, First out, meaning that the most recently added element is the first one to remove.

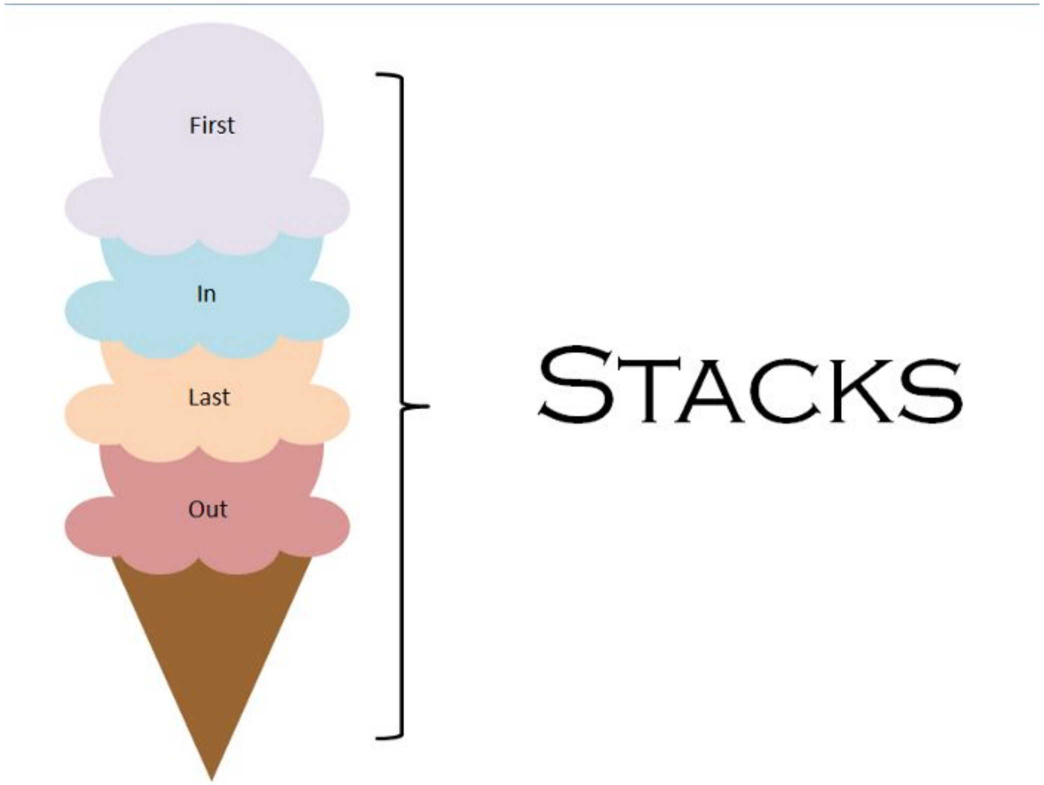
A stack has two main operations that occur only at the top of the stack: push and pop. The push operation places an element at the top of stack whereas the pop operation removes an element from the top of the stack.

Stack Data Structure

Stacks are simple to understand. if you look at the image above, you can understand most of the properties of the stack.

To put it simply, the first item in the stack is the last item that gets removed.

JavaScript Array type provides the push() and pop() methods that allow you to use an array as a stack.

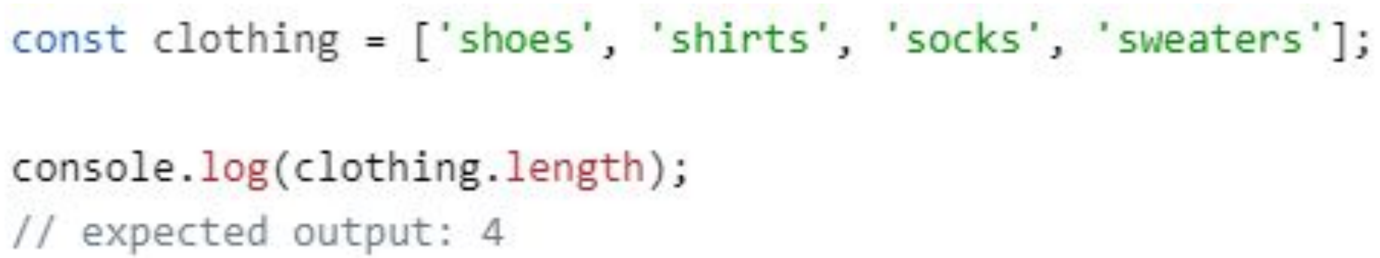


Array Properties

Video - Stacks

Array.Length

The **length** property of an object which is an instance of type Array sets or returns the number of elements in that array. The value is an unsigned, 32-bit integer that is always numerically greater than the highest index in the array.

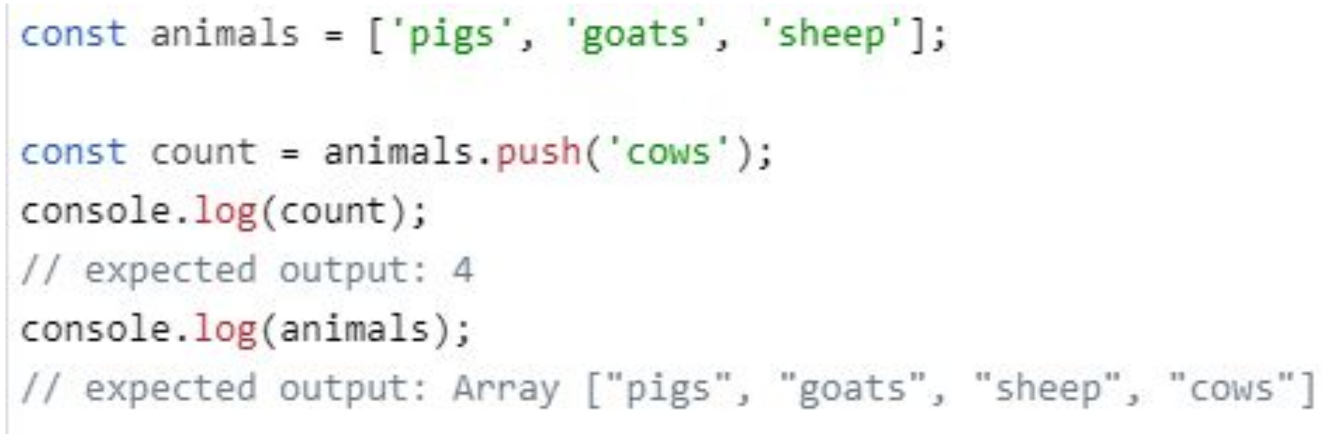


```
const clothing = ['shoes', 'shirts', 'socks', 'sweaters'];

console.log(clothing.length);
// expected output: 4
```

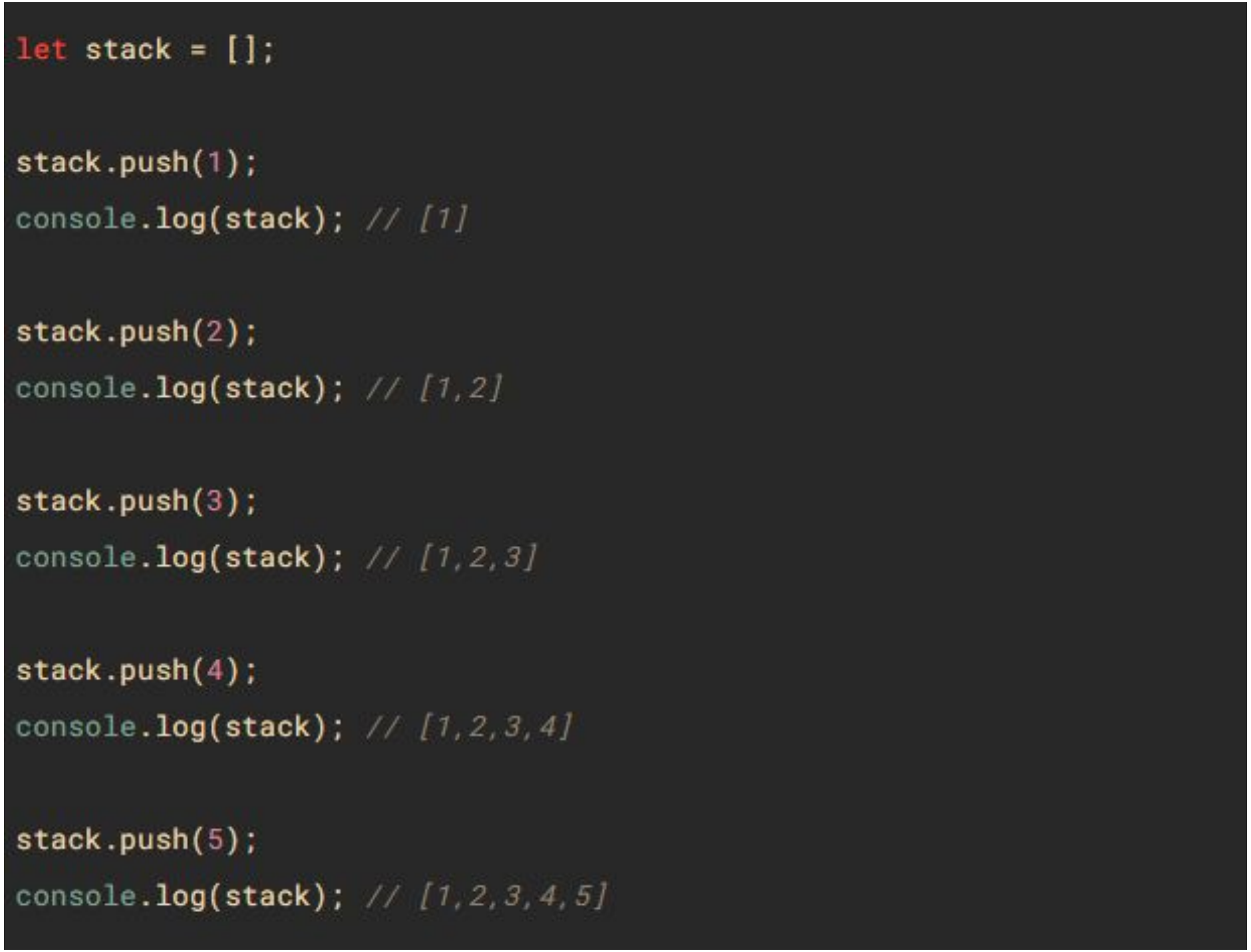
Array Methods

Array.Push()

The **push()** method adds one or more elements to the end of an array and returns the new length of the array.



```
const animals = ['pigs', 'goats', 'sheep'];

const count = animals.push('cows');
console.log(count);
// expected output: 4
console.log(animals);
// expected output: Array ["pigs", "goats", "sheep", "cows"]
```

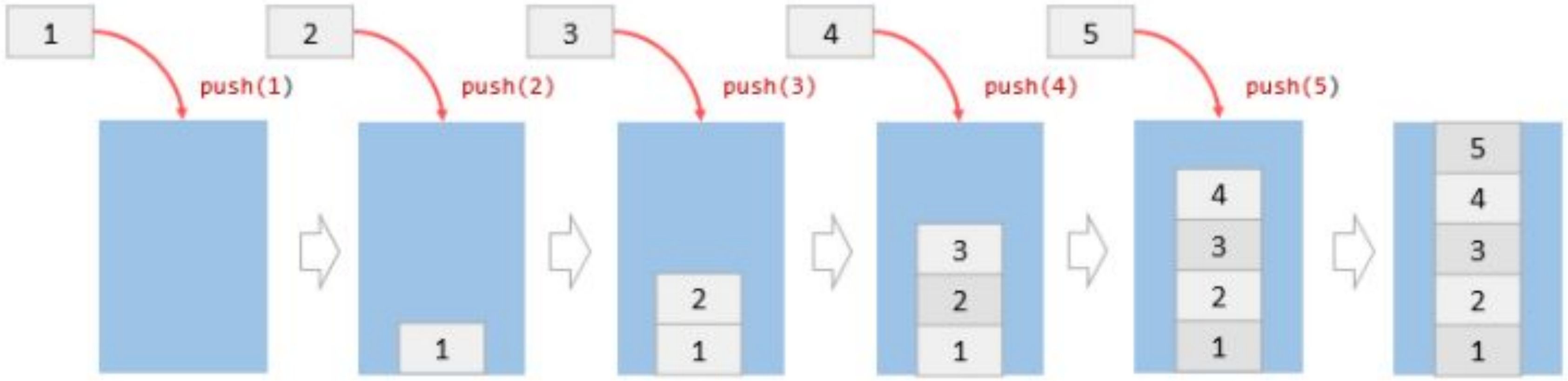


```javascript
let stack = [];

stack.push(1);
console.log(stack); // [1]

stack.push(2);
console.log(stack); // [1,2]

stack.push(3);
console.log(stack); // [1,2,3]

stack.push(4);
console.log(stack); // [1,2,3,4]

stack.push(5);
console.log(stack); // [1,2,3,4,5]
```

Array.Push() cont.

Initially, the stack is empty. Each time, we call the push() method to add a number to the stack.

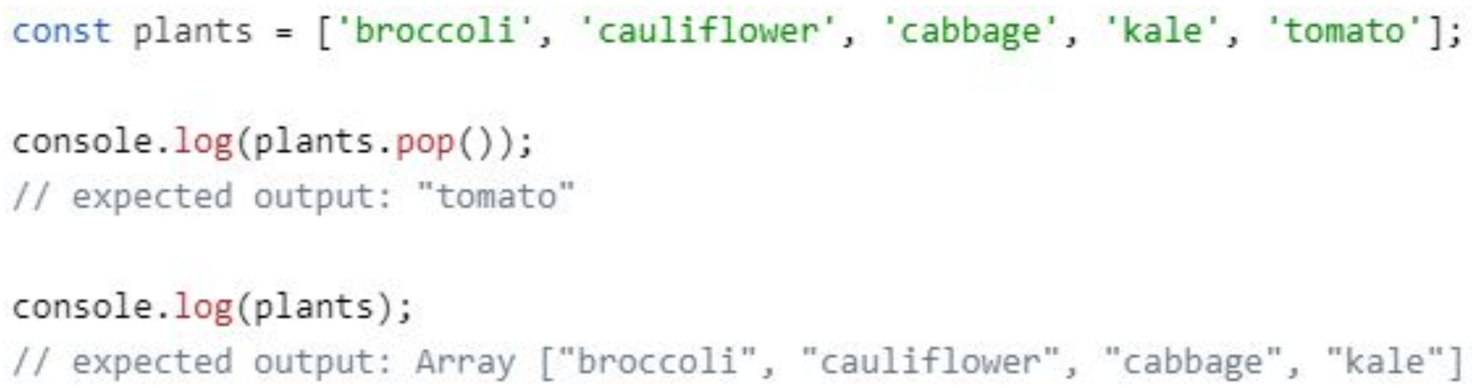
After 5 calls, the stack has 5 elements.



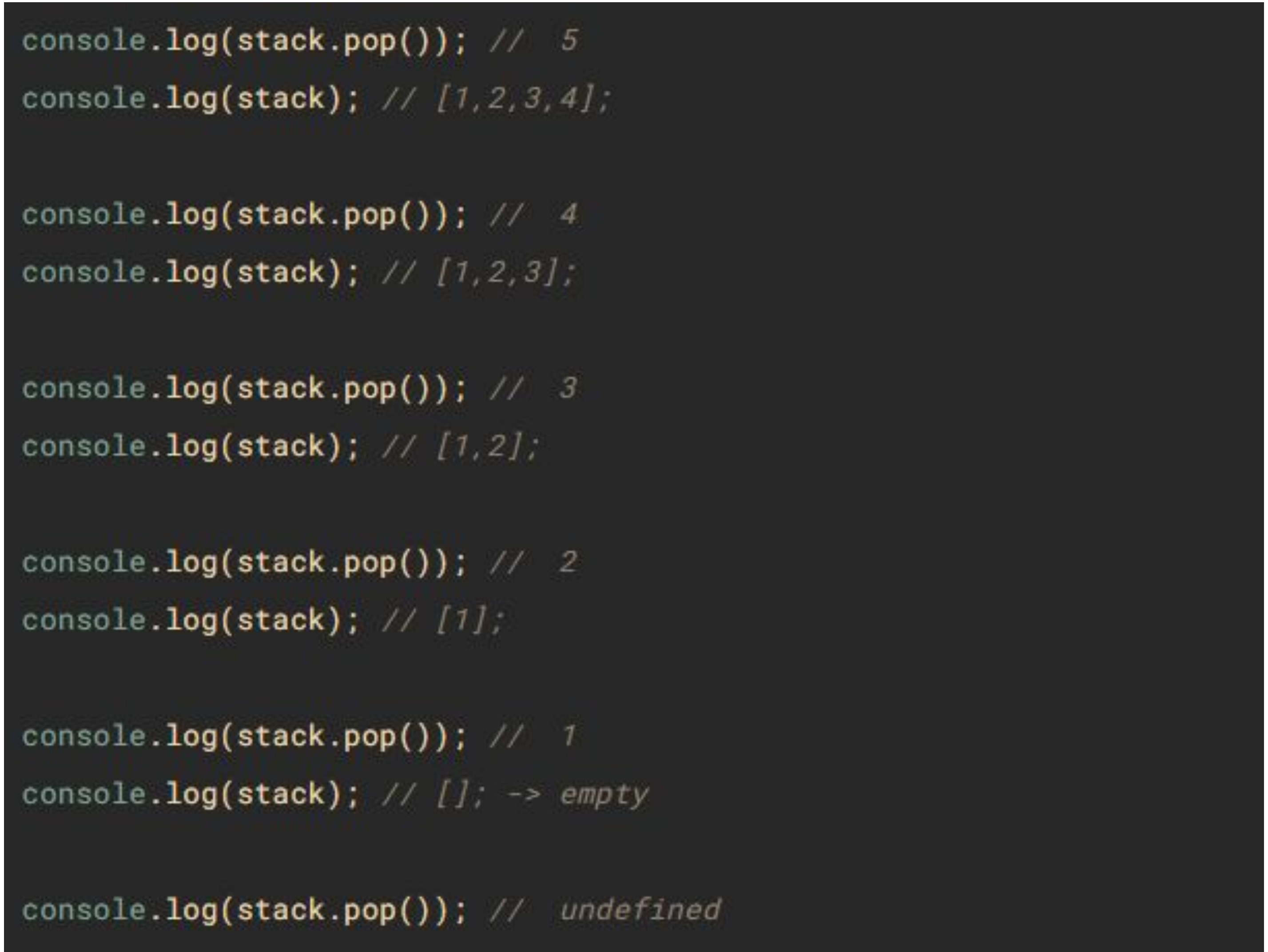
Array.Pop()

The **pop()** method removes the **last** element from an array and returns that element.

This method changes the length of the array.



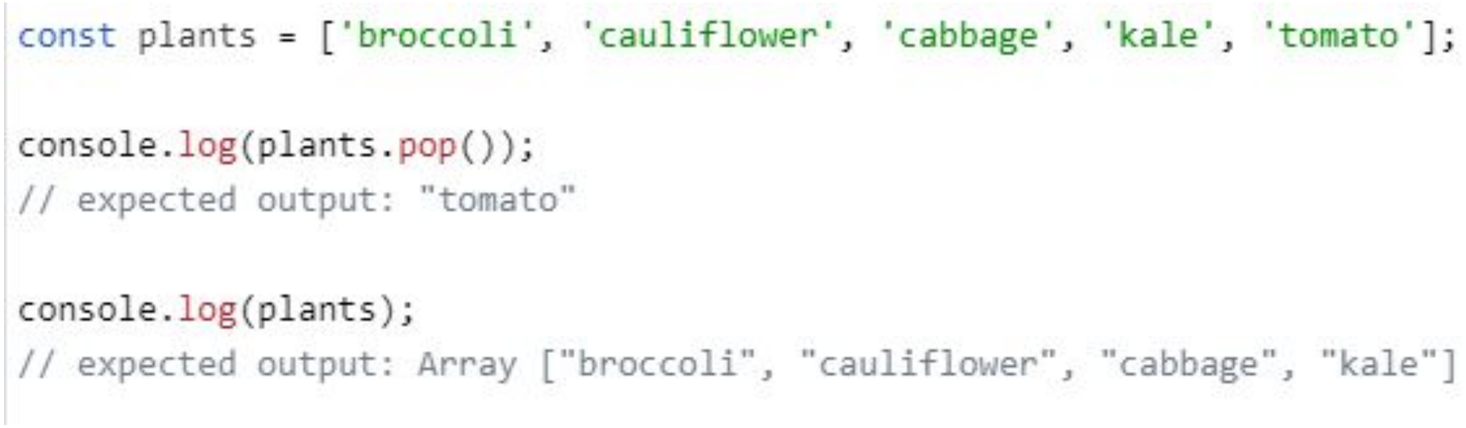
```
const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop());
// expected output: "tomato"

console.log(plants);
// expected output: Array ["broccoli", "cauliflower", "cabbage", "kale"]
```



```
console.log(stack.pop()); //  5
console.log(stack); // [1,2,3,4];

console.log(stack.pop()); //  4
console.log(stack); // [1,2,3];

console.log(stack.pop()); //  3
console.log(stack); // [1,2];

console.log(stack.pop()); //  2
console.log(stack); // [1];

console.log(stack.pop()); //  1
console.log(stack); // []; -> empty

console.log(stack.pop()); //  undefined
```

Array.Pop()

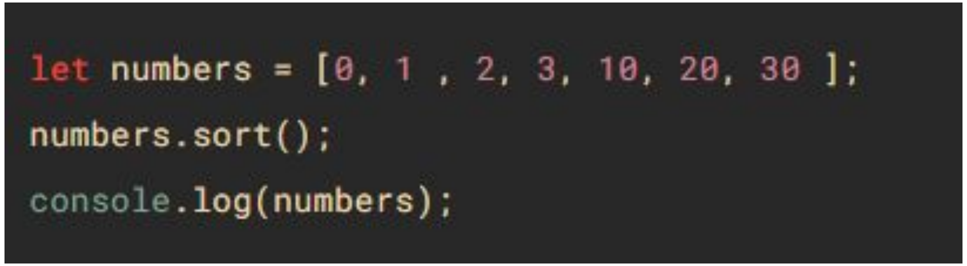
Initially, the stack has 5 elements. The pop() method removes the element at the end of the array i.e., at the top of the stack one at a time. After five operations, the stack is empty.



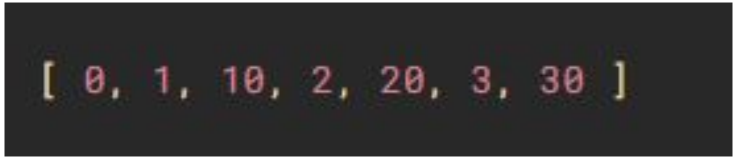
```
const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop());
// expected output: "tomato"

console.log(plants);
// expected output: Array ["broccoli", "cauliflower", "cabbage", "kale"]
```

Array.Sort()

The **sort()** method sorts the elements of an array *in place* and returns the sorted array.
The default sort order is ascending.



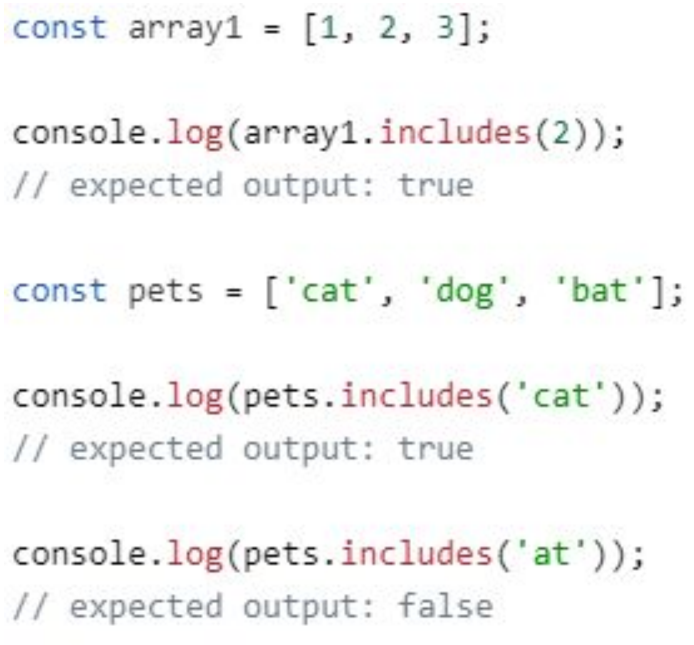
```
let numbers = [0, 1 , 2, 3, 10, 20, 30 ];
numbers.sort();
console.log(numbers);
```



```
[ 0, 1, 10, 2, 20, 3, 30 ]
```

Array.includes()

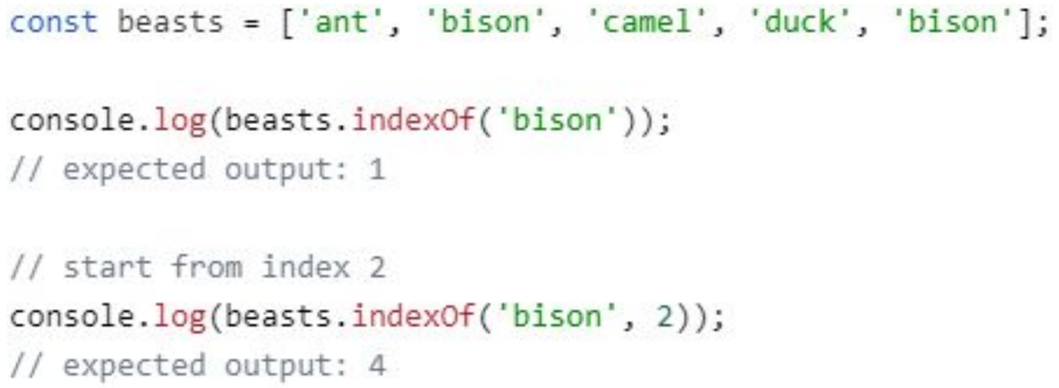
The **includes()** method determines whether an array includes a certain value among its entries, returning true or false as appropriate.



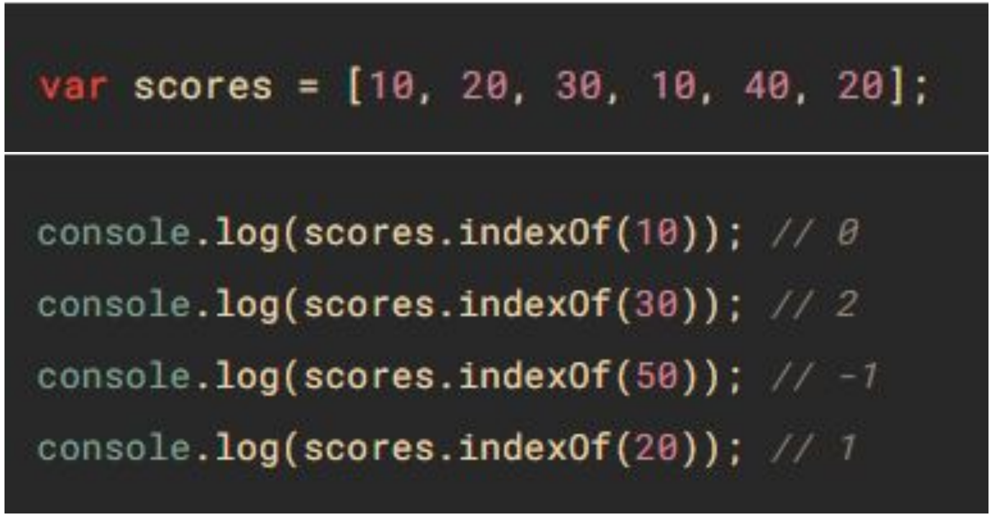
```
const array1 = [1, 2, 3];

console.log(array1.includes(2));
// expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// expected output: true

console.log(pets.includes('at'));
// expected output: false
```

Array.indexOf()

The **indexOf()** method returns the first index at which a given element can be found in the array, or -1 if it is not present.

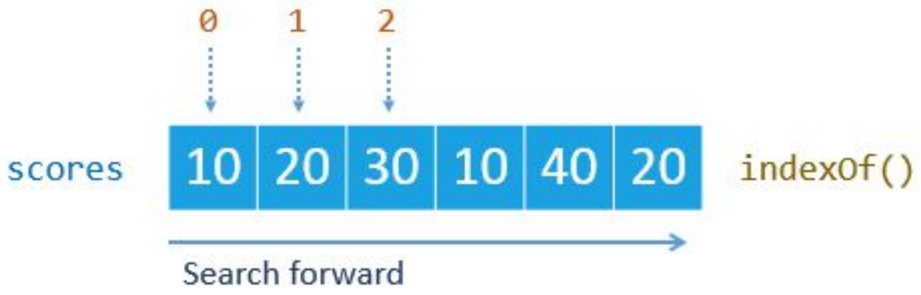


```
const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison'));
// expected output: 1

// start from index 2
console.log(beasts.indexOf('bison', 2));
// expected output: 4
```



```
var scores = [10, 20, 30, 10, 40, 20];
```

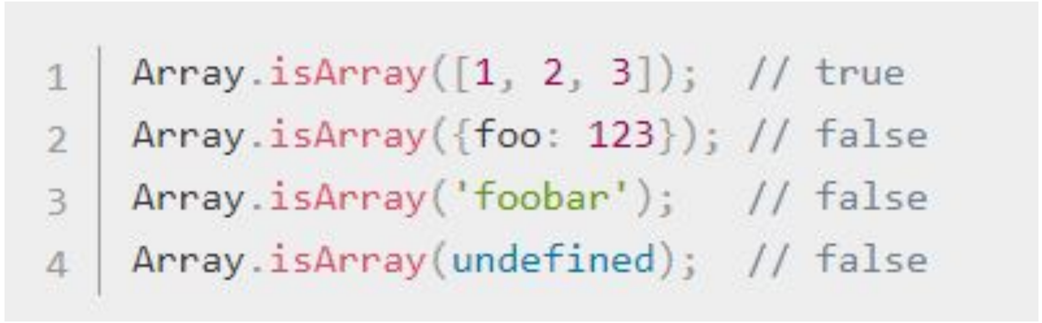
```
console.log(scores.indexOf(10)); // 0
console.log(scores.indexOf(30)); // 2
console.log(scores.indexOf(50)); // -1
console.log(scores.indexOf(20)); // 1
```



Example of Array.indexOf()

Array.isArray()

The **Array.isArray()** method determines whether the passed value is an Array.



```
Array.isArray([1, 2, 3]);  // true
Array.isArray({foo: 123}); // false
Array.isArray('foobar');   // false
Array.isArray(undefined);  // false
```

Advanced Data Structures

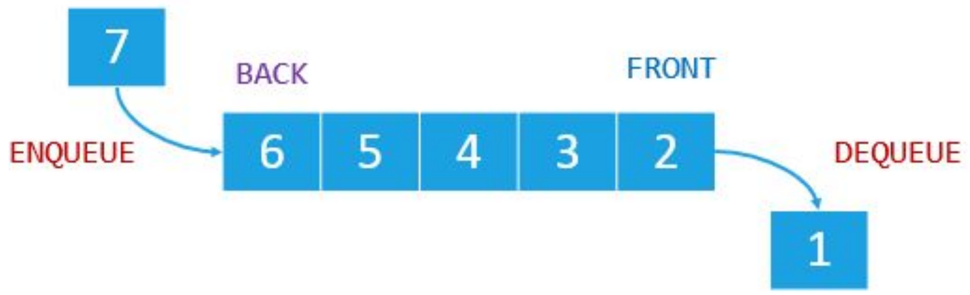
Queues

A queue is an ordered list of elements where an element is inserted at the end of the queue and is removed from the front of the queue.

Unlike a stack, which works based on the last-in, first-out (LIFO) principle, a queue works based on the first-in, first-out (FIFO) principle.

A queue has two main operations involving inserting a new element and removing an existing element.

The insertion operation is called *enqueue*, and the removal operation is called *dequeue*. The enqueue operation inserts an element at the end of the queue, whereas the dequeue operation removes an element from the front of a queue.

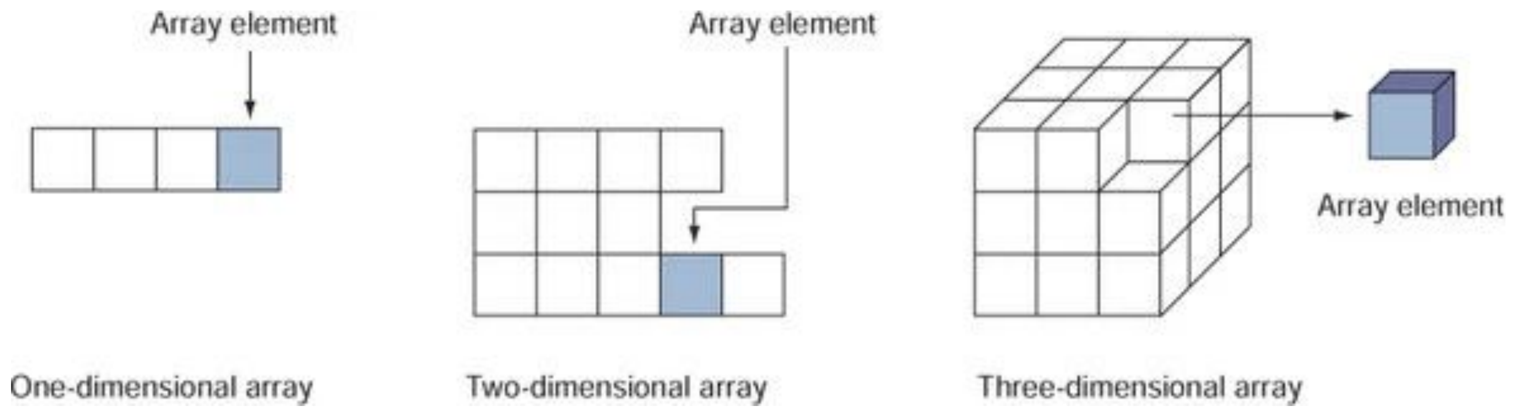


In JavaScript we can use push() to add to the end of array, similar to enqueue operation.
Also, we can use shift() method, similar to dequeue operation

Enqueue vs Dequeue

A JavaScript multidimensional array is an array of arrays, or, in other words, an array whose elements consist of arrays.

```
var ar = [
    ['apple', 'orange', 'pear'],
    ['carrots', 'beans', 'peas'],
    ['cookies', 'cake', 'muffins', 'pie']
];
```



But wait....there is Multi Dimensional arrays!