Array Operations: Creating Stacks and Queues

In this lesson, we'll cover some very useful array operations in JavaScript.
Let's begin!

WE'LL COVER THE FOLLOWING

Array is a versatile type.

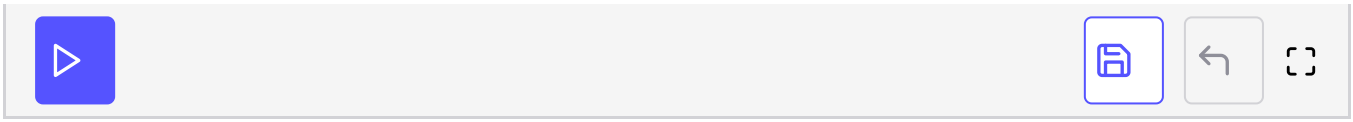
It provides powerful operations that make it possible to use arrays not only as **single lists** but also as **queues** or **stacks**.

The `Array type` is also a great example to demonstrate the *strength of JavaScript*.

Dynamic resizing of arrays

An `Array` instance **dynamically resizes itself**:

```
var arr = [1, 2, 3];
arr[5] = 6;
console.log(arr.length);     // 6
console.log(arr.toString()); // 1,2,3,,,6
```



In this sample the original array, `arr`, contains only three elements.

Setting the sixth element's value, with index 5, automatically augments the array to store six elements as the log output indicates.

Using arrays to create a queue

You can easily create a queue from an array, as Listing 7-16 demonstrates.

Listing 7-16: Exercise-07-16/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Array as a queue</title>
  <script>
    // --- Queue
    var arr = [1, 2, 3];
    arr.push(4);
    arr.push(5);
    console.log(arr.shift());    // 1
    console.log(arr.shift());    // 2
    console.log(arr.toString()); // 3,4,5

    // --- Reverse queue
    var arr = [1, 2, 3];
    arr.unshift(0);
    arr.unshift(-1);
    console.log(arr.pop());      // 3
    console.log(arr.pop());      // 2
    console.log(arr.toString()); // -1,0,1
  </script>
</head>
<body>
  Listing 7-16: View the console output
</body>
</html>
```

This code demonstrates two queues:

- The **first implementation** appends new elements to the end of the array after the element with the highest index, and reads the first element;
- the **second** does it the opposite way.

The **first way** leverages the `push()` method that appends a new element to the end of the array, and returns its new length. To read from the queue, the `shift()` method is used that removes the element at the first position and returns it back.

The original content of the array is `[1, 2, 3]`. As you can see, the code puts `4` and then `5` into the queue, then reads two elements that happen to be `1` and `2`. At the end, the array contains `[3, 4, 5]`.

The **second way** uses the `unshift()` method to add an element to the front of an array, and reads the array with `pop()` that removes the last element from the array and returns this removed element.

The comments in the code show how this second queue implementation changes the array content.

Show Useful Info

Using arrays to create a stack

Using the `push()` and `pop()` methods, it is very simple to create a stack, as Listing 7-17 shows.

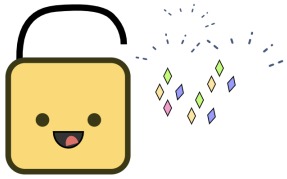
Listing 7-17: Exercise-07-17/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Array as a stack</title>
  <script>
    var arr = [1, 2, 3];
    arr.push(4);
    arr.push(5);
    console.log(arr.pop());      // 5
    console.log(arr.pop());      // 4
```

```
    console.log(arr.toString()); // 1,2,3
  </script>
</head>

<body>
  Listing 7-17: View the console output
</body>
</html>
```

Achievement unlocked!



Congratulations! You've learned how to create a simple stack and a queue using arrays in JavaScript.

Great work! Give yourself a round of applause! :)

In the *next lesson*, we will dive into some more array operations in JavaScript.