JavaScript Array Functions and Methods

Introduction to Arrays

Arrays in JavaScript are dynamic, meaning they can grow or shrink, and they can hold multiple data types. Below is an array of numbers:

```
let arr = [1, 2, 3, 4, 5, 6];
```

Output: [1, 2, 3, 4, 5, 6]

Arrays in JavaScript are technically objects, but they come with built-in methods that make working with ordered data easier. Below, we will explore various array methods.

Array Methods

Let's start with some basic methods for manipulating arrays:

toString()

The `toString()` method converts an array to a string.

```
arr.toString();
```

Output: "1,2,3,4,5,6"

join()

The `join()` method joins all array elements into a string, with a custom separator if needed.

```
arr.join(" # ");
```

Output: "1 # 2 # 3 # 4 # 5 # 6"

pop() and push()

The `pop()` method removes the last element, and `push()` adds an element at the end of the array.

```
arr.pop(); // Removes the last element
arr.push(69); // Adds 69 to the array
```

> Output after pop: [1, 2, 3, 4, 5]
> Output after push: [1, 2, 3, 4, 5, 69]

shift() and unshift()

The `shift()` method removes the first element, and `unshift()` adds an element at the start of the array.

```
arr.shift(); // Removes the first element
arr.unshift(9); // Adds 9 to the start of the array
```

> Output after shift: [2, 3, 4, 5, 69]
> Output after unshift: [9, 2, 3, 4, 5, 69]

delete

`delete` removes the value but leaves the position undefined.

```
delete arr[5];
```

> Output: [9, 2, 3, 4, 5, undefined]

Advanced Methods

concat()

Concatenates arrays into a new array:

```
let a = [1, 2, 3];
let b = [4, 5, 6];
let c = [7, 8, 9];
a.concat(b, c);
```

> Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

splice()

`splice()` can be used to add or remove items from an array:

```
arr.splice(1, 3); // Removes 3 items starting from index 1
arr.splice(1, 3, 10, 11, 12); // Replaces 3 items starting at index 1 with 10, 11, and 12
```

> After removing: [9, undefined, 5]
> After replacing: [9, 10, 11, 12, 5]

slice()

Returns a portion of the array without modifying it:

```
arr.slice(1, 3);
```

> Output: [10, 11]

Looping through Arrays

Arrays can be looped through using different loops:

for Loop

```
for (let i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

forEach Loop

`forEach()` allows you to iterate over array elements with more details:

```
arr.forEach((value, index, array) => {
    console.log(value, index, array);
});
```

Higher-Order Functions

map()

`map()` creates a new array by applying a function to every element:

```
let newArr = arr.map(e => e ** 2);
```

> Output: [81, 100, 121, 144, 25]

filter()

`filter()` creates a new array with all elements that pass the test implemented by the provided function:

```
let filteredArr = newArr.filter(e => e > 100);
```

> Output: [121, 144]

reduce()

`reduce()` applies a function against an accumulator to reduce all array elements to a single value:

```
let sum = [1, 2, 3, 4].reduce((a, b) => a + b);
```

Output: 10

Array.from()

Converts an array-like object into an array:

```
Array.from("kartik");
```

Output: ['k', 'a', 'r', 't', 'i', 'k']