Lesson:

Array Methods

PW SKILLS

Topics Covered

- What are Arrays Methods
- Explain different types of Array methods with examples

What are Arrays Methods

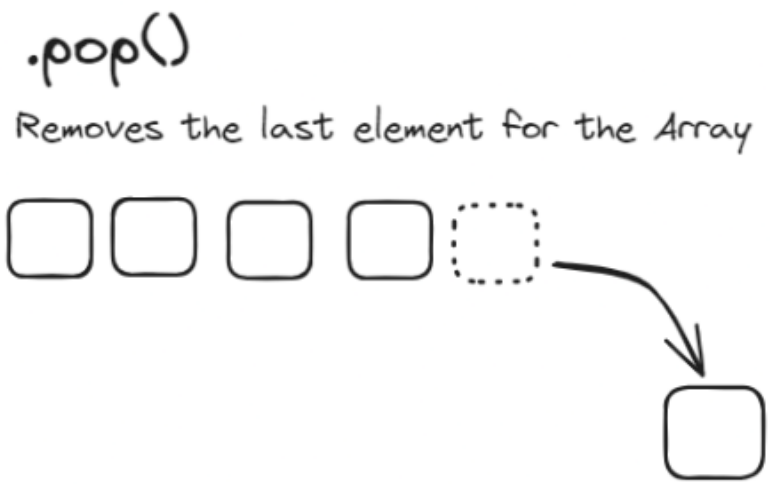
Array methods are built-in functions in JavaScript that allow you to perform various operations on arrays. They can be used to manipulate arrays in various ways, including adding and removing elements, sorting, transforming, and filtering the elements of an array.

Explain different types of Array methods with example

Some of the different types of basic Array methods include the following -
pop(), push(), shift(), unshift(), concat(), join(), slice(), splice(), reverse(), indexOf(), toString(), flat(), isArray().

pop()

The pop() method in JavaScript is used to remove the last element of an array and return the removed element. The method modifies the original array, reducing its length by one.

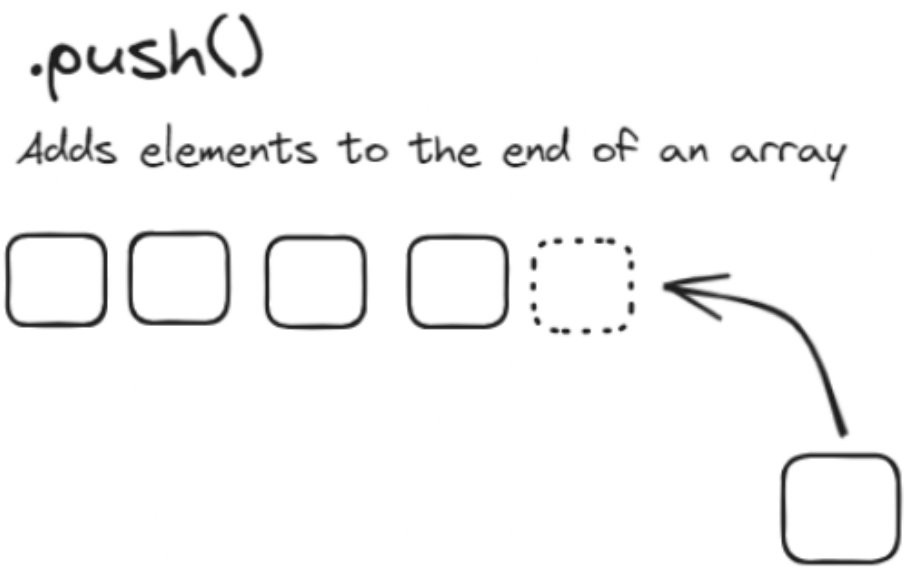


Examples -

```JavaScript
let fruits = ['apple', 'banana', 'cherry'];
let last = fruits.pop();
console.log(fruits); // Output: [ 'apple', 'banana' ]
console.log(last); // Output: 'cherry'
```

In the example above, fruits.pop() removed the last element 'cherry' from the fruits array and returned it. The fruit array now only contains two elements, 'apple' and 'banana'.
It's important to note that the pop() method modifies the original array, so if you need to keep the original array intact, you should make a copy of it before using pop().

push()
The push() method in JavaScript is used to add one or more elements to the end of an array and return the new length of the array. The push() method modifies the original array, so the elements are added to the end of the original array



Here's an example of using the push() method:

```javascript
let fruits = ['apple', 'banana'];
let length = fruits.push('cherry');
console.log(fruits); // Output: ['apple', 'banana', 'cherry']
console.log(length); // Output: 3
```

In the example above, the push() method adds the element 'cherry' to the end of the fruits array, and the new length of the fruits array is returned, which is 3.
It's important to note that the push() method modifies the original array. If you need to keep the original array intact, you should make a copy of it before using the push() method.

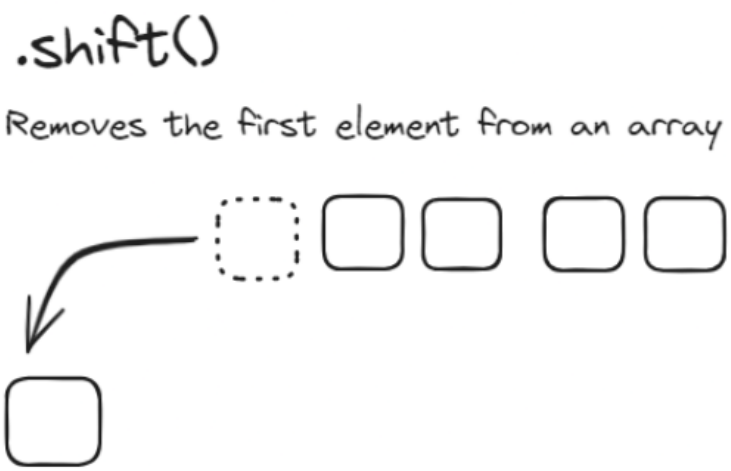
You can also use the push() method to add multiple elements to an array, like this:

```javascript
let fruits = ['apple', 'banana'];
let length = fruits.push('cherry', 'orange');
console.log(fruits); // Output: ['apple', 'banana', 'cherry',
'orange']
console.log(length); // Output: 4
```

In the example above, the push() method adds the elements 'cherry' and 'orange' to the end of the fruits array, and the new length of the fruits array is returned, which is 4.

shift()

The shift() method in JavaScript is used to remove the first element from an array and return the removed element. This method changes the length of the array, shifting all other elements to a lower index, to fill the gap left by the removed element.



Here's an example:

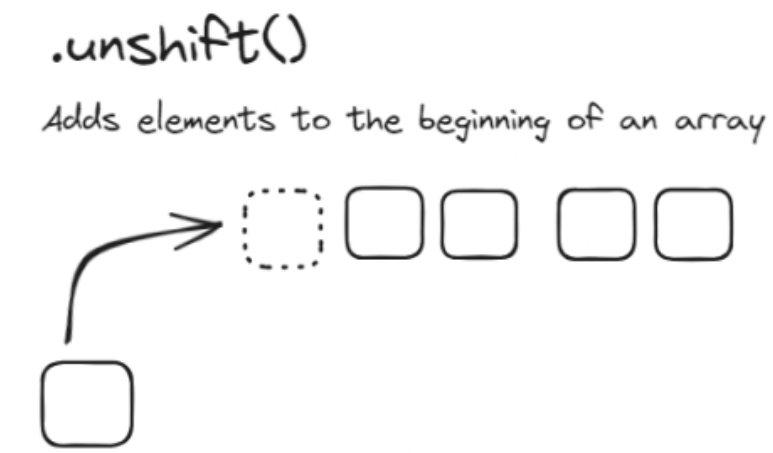
```javascript
let fruits = ['apple', 'banana', 'cherry'];
let first = fruits.shift();
console.log(fruits); // Output: ['banana', 'cherry']
console.log(first); // Output: 'apple'
```

In the example above, the shift() method removes the first element ('apple') from the fruits array and returns it. The returned element can then be stored in a variable, as in this example.

It's important to note that the shift() method only removes the first element of an array. If you need to remove elements from other positions in an array, you can use the splice() method.

unshift()

The unshift() method in JavaScript is used to add one or more elements to the beginning of an array and return the new length of the array. This method changes the length of the array and shifts all existing elements to a higher index to make room for the new elements.



Here's an example of using the unshift() method:

```javascript
let fruits = ['banana', 'cherry'];
let length = fruits.unshift('apple');
console.log(fruits); // Output: ['apple', 'banana', 'cherry']
console.log(length); // Output: 3
```

In the example above, the unshift() method adds the element 'apple' to the beginning of the fruits array, and the new length of the fruits array is returned, which is 3.

It's important to note that the unshift() method modifies the original array. If you need to keep the original array intact, you should make a copy of it before using the unshift() method.

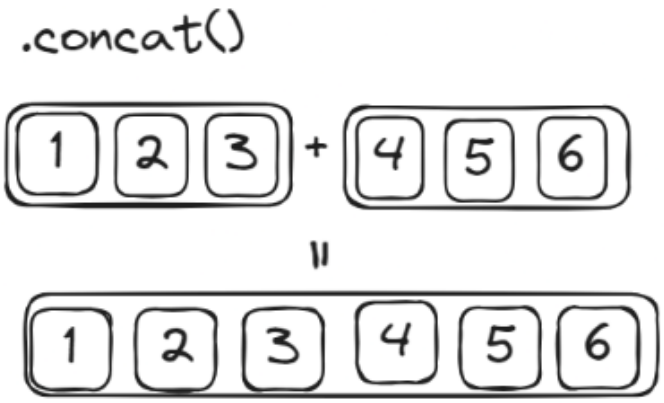
You can also use the unshift() method to add multiple elements to an array, like this:

```javascript
let fruits = ['banana', 'cherry'];
let length = fruits.unshift('apple', 'orange');
console.log(fruits); // Output: ['apple', 'orange', 'banana',
'cherry']
console.log(length); // Output: 4
```

In the example above, the unshift() method adds the elements 'apple' and 'orange' to the beginning of the fruits array, and the new length of the fruits array is returned, which is 4.

concat():

The concat method in JavaScript is used to concatenate two or more arrays into a single array. It returns a new array that consists of the elements from the original array(s) and the elements from one or more additional arrays. The original arrays are not modified.



Here's an example of how to use the concat method:

```javascript
let array1 = [1, 2, 3];
let array2 = [4, 5, 6];
let newArray = array1.concat(array2);
console.log(newArray);
// Output: [1, 2, 3, 4, 5, 6]
```

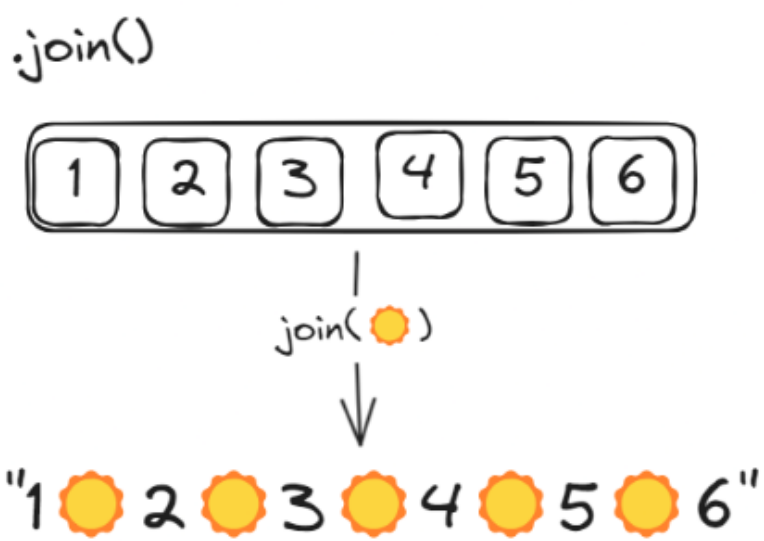
In this example, the concat method is used to concatenate the array array1 and array2 into a new array called newArray. The resulting array contains all the elements from array1 and array2.

The concat method can also be used to concatenate more than two arrays by passing multiple arrays as arguments:

```javascript
let array1 = [1, 2, 3];
let array2 = [4, 5, 6];
let array3 = [7, 8, 9];
let newArray = array1.concat(array2, array3);
console.log(newArray);
// Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In this example, the concat method is used to concatenate three arrays into a single array. The resulting array contains all the elements from array1, array2, and array3.
It's important to note that the concat method does not modify the original arrays, but returns a new array that is the result of the concatenation. If you need to modify the original array, you can assign the result of the concat method back to the original array.

join()
In JavaScript, the join() method is used to join all elements of an array into a string. The elements are separated by a specified separator also known as delimiter. If no separator is provided, the elements are joined with a comma (,) by default.

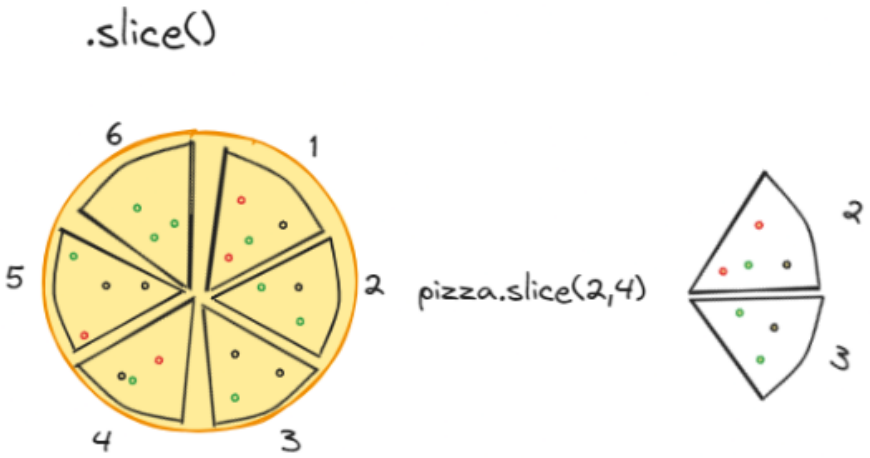


Here's an example:

```javascript
let fruits = ['apple', 'banana', 'cherry'];
let result = fruits.join();

console.log(result); // 'apple,banana,cherry'

let result = fruits.join('-');
console.log(result); // 'apple-banana-cherry'
```

Note that the join() method does not modify the original array, it returns a new string that consists of all elements joined together.

slice()
The slice() method in JavaScript is used to extract a portion of an array and return a new array. The original array is not modified.



Here's the syntax for using the slice() method:

```
JavaScript
array.slice(start, end);
```

The slice() method takes two optional parameters: start and end.

- start is the index at which to begin extraction. If the start is negative, it is treated as array.length + start.
- end is the index before which to end extraction. The element at this index is not included in the extracted portion. If the end is omitted, all elements from the start to the end of the array will be included in the extracted portion. If the end is negative, it is treated as an array.length + end.

Here's an example of using the slice() method:

```
JavaScript
let fruits = ['apple', 'banana', 'cherry', 'orange'];
let citrus = fruits.slice(1, 3);
console.log(fruits); // Output: ['apple', 'banana', 'cherry',
'orange']
console.log(citrus); // Output: ['banana', 'cherry']
```

In the example above, the slice() method is used to extract the portion of the fruit array from index 1 ('banana') to index 3 ('cherry'), exclusive. The extracted portion is stored in the citrus array, which is ['banana', 'cherry'].

It's important to note that the slice() method does not modify the original array, but returns a new array that contains the extracted portion.

Here's another example of using the slice() method:

```javascript
let numbers = [1, 2, 3, 4, 5];
let part = numbers.slice(2);
console.log(numbers); // Output: [1, 2, 3, 4, 5]
console.log(part); // Output: [3, 4, 5]
```

In the example above, the slice() method is used to extract the portion of the numbers array from index 2 to the end of the array. The extracted portion is stored in the part array, which is [3, 4, 5].

splice()

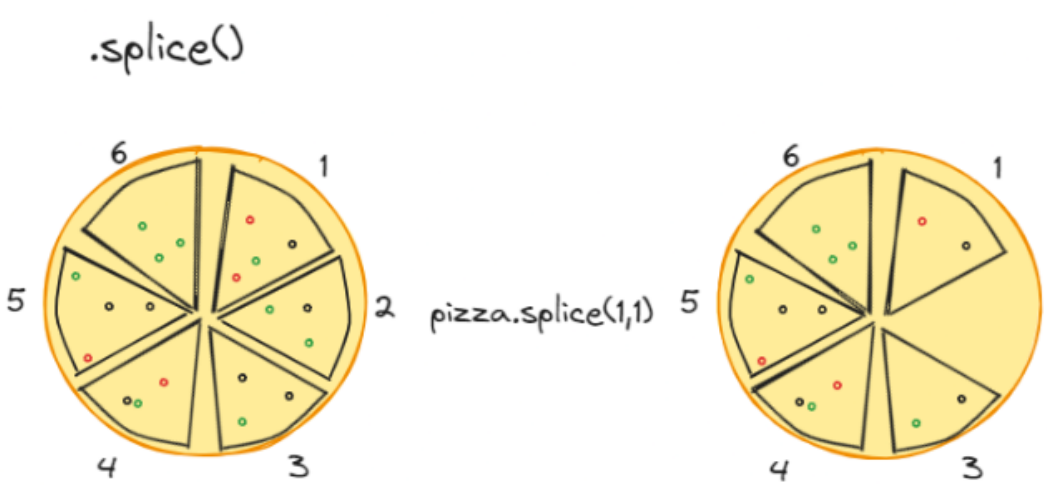
The splice() method in JavaScript is used to add or remove elements from an array. The method modifies the original array and returns the removed elements (if any) in a new array.

Here's the syntax for using the splice() method:

```javascript
array.splice(start, deleteCount, item1, item2, ...);
```

The splice() method takes three mandatory parameters:

- **start:** The index at which to start changing the array. If the start is negative, it is treated as array.length + start.
- **deleteCount:** The number of elements to remove. If deleteCount is 0, no elements are removed. If deleteCount is greater than the number of elements that exist after a start, all elements from the start to the end of the array will be removed.
- **item1, item2, ... :** The elements to add to the array, starting at the start. If no elements are added, this parameter can be omitted.



Here's an example of using the splice() method:

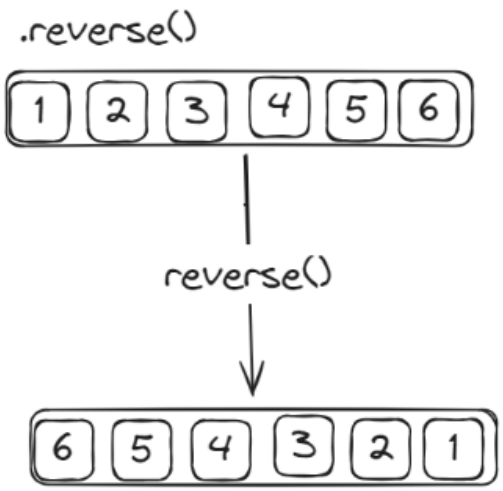
```javascript
let numbers = [1, 2, 3, 4, 5];
let removed = numbers.splice(2, 2, 6, 7);
console.log(numbers); // Output: [1, 2, 6, 7, 5]
console.log(removed); // Output: [3, 4]
```

In the example above, the splice() method is used to remove two elements (3 and 4) from the numbers array starting at index 2 and add two elements (6 and 7) in their place. The removed elements are returned in the removed array, which is [3, 4]. The modified numbers array is now [1, 2, 6, 7, 5].

It's important to note that the splice() method can be used to both add and remove elements from an array, and that it modifies the original array. This makes it a powerful tool for manipulating arrays, but it's also important to be careful when using it to ensure that the original array is changed as intended.

reverse()

The array.reverse() method in JavaScript is used to reverse the order of the elements in an array in place. The method modifies the original array and does not create a new array.



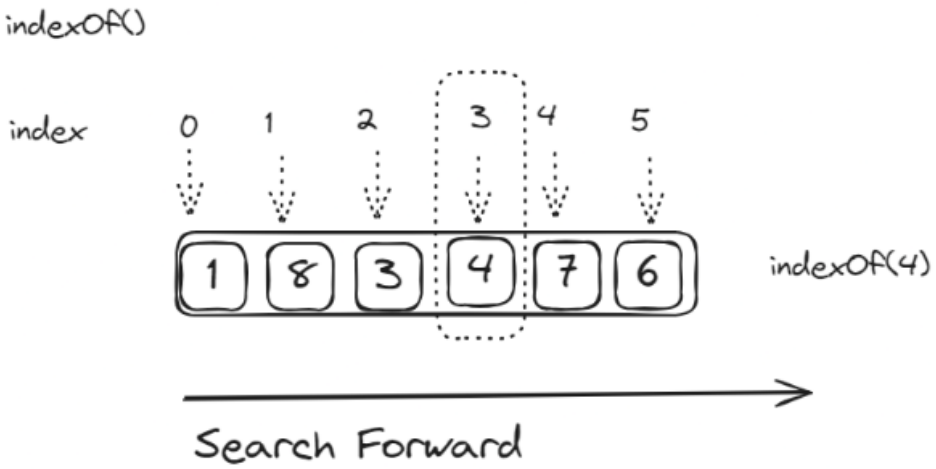
Here's an example:

```javascript
let fruits = ['apple', 'banana', 'kiwi', 'mango'];
fruits.reverse();
console.log(fruits);
// Output: ["mango", "kiwi", "banana", "apple"]
```

Note that the reverse method only reverses the order of the elements in the array and does not sort the elements in any way. If you want to sort the elements in ascending or descending order, you should use the Array.sort() method in conjunction with the reverse method.

indexOf()

The Array.indexOf() method in JavaScript is used to search the array for an element and return its index. The method returns the first index at which the given element can be found in the array, or -1 if it is not present.



Here's an example of indexOf()

```JavaScript
let fruits = ['apple', 'banana', 'kiwi', 'mango'];
let index = fruits.indexOf('kiwi');
console.log(index);
// Output: 2
```

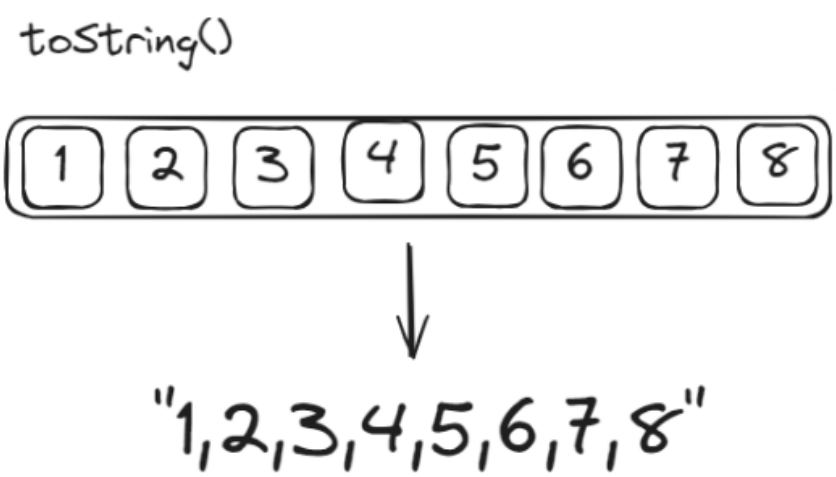
You can also provide a second argument, which specifies the index at which to start searching for the element:

```JavaScript
let fruits = ['apple', 'banana', 'kiwi', 'mango', 'kiwi'];
let index = fruits.indexOf('kiwi', 3);
console.log(index);
// Output: 4
```

Note that the indexOf method uses the strict equality operator (===) to compare the elements, so if you are searching for a value that is not a string or number, make sure to provide the correct type.

toString()

The array.toString method in JavaScript is used to convert the array of elements to a string. It returns a string object. The **toString** method of arrays always calls the **join()** array method internally, which joins the array and returns one string containing each array element separated by commas.



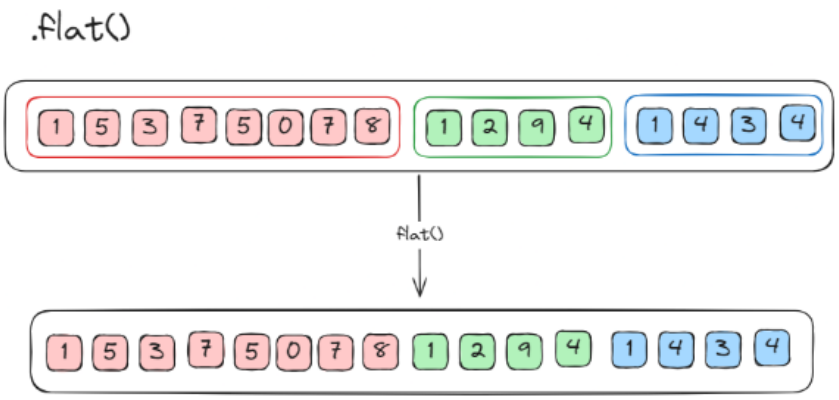
Here is the example of toString() method -

```javascript
const numbers = [1, 2, 3, 4, 5, 6, 7, 8];

const numberOfString = numbers.toString();
console.log("Type of - ", typeof numberOfString);
console.log("Number of String", numberOfString);
//---- output ----

// Type of - string
// Number of String "1, 2, 3, 4, 5, 6, 7, 8"
```

Note - it's important to note that toString() does not modify the original array, it returns a new string representation of the array.

flat()

The array.flat() method in JavaScript is used to create a new array with all sub multiples array elements present inside a particular array, concatenating all the sub-arrays.
The flat() method accepts an optional parameter (dept) that specifies the depth level of flattening. By default depth is set to 1, meaning it flattens one level of nested arrays.

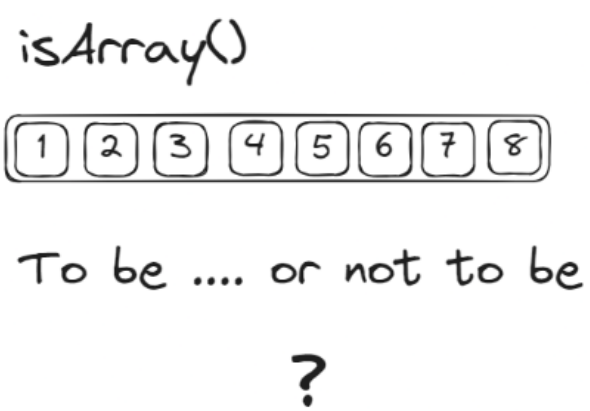


Example -

```javascript
const nestArray = [1, 2, 3, [4, 5]];
const flatArray = nestArray.flat(); // default
const flatArrayOne = nestArray.flat(1); // default
console.log(flatArray); // [1,2 3, 4, 5]

console.log(flatArrayOne) // [1,2 3, 4, 5]

const nestDepth = [0, 1, 2, [3, 4, [5, 6, [7, 8]]]];
const flatOneDefault = nestDepth.flat(1);
console.log(flatOneDefault); // [ 0, 1, 2, 3, 4, [ 5, 6, [ 7,
8 ] ] ]

const flatTwo = nestDepth.flat(2);
console.log(flatTwo); // [ 0, 1, 2, 3, 4, 5, 6, [ 7, 8 ] ]

const flatThree = nestDepth.flat(3);
console.log(flatThree); // [  0, 1, 2, 3, 4,5, 6, 7, 8 ]

const infinity = nestDepth.flat(Infinity);
console.log(infinity); // [  0, 1, 2, 3, 4,5, 6, 7, 8 ]
```

In the above example, the parameter depth value is passed with a desired number, to flatten multiple levels of the nesting array.
Infinity can be also passed in the parameter to convert a multi-level nested array to a single array.

isArray()
The array.isArray() method in javascript is used to determine whether a given value is an array or not. The method returns true if the value passed to it is an array, and false if the value passed is not an array.



Example -

```javascript
const lanArr = ["javascript", "python", "java", "golang"];
console.log(Array.isArray(lanArr)); // true

const cityName = "Bangalore";
console.log(Array.isArray(cityName)); // false
```

Interview Points

1. Concatenate two arrays of numbers and then sort them in descending order.

```javascript
const array1 = [5, 3, 8, 1];
const array2 = [7, 2, 6, 4];
```

Ans:

```javascript
const array1 = [5, 3, 8, 1];
const array2 = [7, 2, 6, 4];

const concatenatedAndSorted = (() => {
  const result = array1.slice().concat(array2.slice()); //
Concatenate without modifying original arrays
  return result.sort((a, b) => b - a); // Sort in descending
order
})();

console.log(concatenatedAndSorted); // Result: [8, 7, 6, 5, 4,
3, 2, 1]
```

2. Given an array of numbers, find the sum of unique numbers. Ignore duplicates in the sum calculation.

```javascript
const numbers = [1, 2, 3, 2, 4, 5, 5, 6];
```

Ans:

```javascript
const numbers = [1, 2, 3, 2, 4, 5, 5, 6];
const uniqueNumbers = [];
let uniqueSum = 0;

for (let i = 0; i < numbers.length; i++) {
  if (!uniqueNumbers.includes(numbers[i])) {
    uniqueNumbers.push(numbers[i]);
    uniqueSum += numbers[i];
  }
}

console.log(uniqueSum); // Result: 21 (1 + 2 + 3 + 4 + 5 + 6)
```

3. **Given an array and an integer k, rotate the array to the right by k steps. For example, given the array [1, 2, 3, 4, 5] and k = 2, the rotated array should be [4, 5, 1, 2, 3].**

```javascript
const originalArray = [1, 2, 3, 4, 5];
const k = 2;

// Calculate the effective rotation index
const rotationIndex = k % originalArray.length;

// Rotate the array using array slicing
const rotatedArray =
originalArray.slice(-rotationIndex).concat(originalArray.slice
(0, -rotationIndex));

console.log(rotatedArray); // Result: [4, 5, 1, 2, 3]
```