JavaScript Arrays Review

JavaScript Array Basics

- **Definition**: A JavaScript array is an ordered collection of values, each identified by a numeric index. The values in a JavaScript array can be of different data types, including numbers, strings, booleans, objects, and even other arrays. Arrays are contiguous in memory, which means that all elements are stored in a single, continuous block of memory locations, allowing for efficient indexing and fast access to elements by their index.

Example Code

```
const developers = ["Jessica", "Naomi", "Tom"];
```

- **Accessing Elements From Arrays**: To access elements from an array, you will need to reference the array followed by its index number inside square brackets. JavaScript arrays are zero based indexed which means the first element is at index 0, the second element is at index 1, etc. If you try to access an index that doesn't exist for the array, then JavaScript will return undefined.

Example Code

```
const developers = ["Jessica", "Naomi", "Tom"];

developers[0] // "Jessica"

developers[1] // "Naomi"


developers[10] // undefined
```

- **length Property**: This property is used to return the number of items in a array.

Example Code

```
const developers = ["Jessica", "Naomi", "Tom"];

developers.length // 3
```

- **Updating Elements in an Array**: To update an element in an array, you use the assignment operator (=) to assign a new value to the element at a specific index.

Example Code

```
const fruits = ['apple', 'banana', 'cherry'];
fruits[1] = 'blueberry';

console.log(fruits); // ['apple', 'blueberry', 'cherry']
```

Two Dimensional Arrays

- **Definition**: A two-dimensional array is essentially an array of arrays. It's used to represent data that has a natural grid-like structure, such as a chessboard, a spreadsheet, or pixels in an image. To access an element in a two-dimensional array, you need two indices: one for the row and one for the column.

Example Code

```
const chessboard = [
  ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'],
  ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],
  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
  ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],
  ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r']
];

console.log(chessboard[0][3]); // "Q"
```

Array Destructuring

- **Definition**: Array destructuring is a feature in JavaScript that allows you to extract values from arrays and assign them to variables in a more concise and readable way. It provides a convenient syntax for unpacking array elements into distinct variables.

Example Code

```
const fruits = ["apple", "banana", "orange"];


const [first, second, third] = fruits;


console.log(first); // "apple"

console.log(second); // "banana"

console.log(third); // "orange"
```

- **Rest Syntax**: This allows you to capture the remaining elements of an array that haven't been destructured into a new array.

Example Code

```
const fruits = ["apple", "banana", "orange", "mango", "kiwi"];

const [first, second, ...rest] = fruits;


console.log(first); // "apple"

console.log(second); // "banana"

console.log(rest); // ["orange", "mango", "kiwi"]
```

Common Array Methods

- **push() Method**: This method is used to add elements to the end of the array and will return the new length.

Example Code

```
const desserts = ["cake", "cookies", "pie"];

desserts.push("ice cream");


console.log(desserts); // ["cake", "cookies", "pie", "ice cream"];
```

- **pop() Method**: This method is used to remove the last element from an array and will return that removed element. If the array is empty, then the return value will be undefined.

Example Code

```
const desserts = ["cake", "cookies", "pie"];

desserts.pop();


console.log(desserts); // ["cake", "cookies"];
```

- **shift() Method**: This method is used to remove the first element from an array and return that removed element. If the array is empty, then the return value will be undefined.

Example Code

```
const desserts = ["cake", "cookies", "pie"];

desserts.shift();


console.log(desserts); // ["cookies", "pie"];
```

- **unshift() Method**: This method is used to add elements to the beginning of the array and will return the new length.

Example Code

```
const desserts = ["cake", "cookies", "pie"];

desserts.unshift("ice cream");


console.log(desserts); // ["ice cream", "cake", "cookies", "pie"];
```

- **indexOf() Method**: This method is useful for finding the first index of a specific element within an array. If the element cannot be found, then it will return -1.

Example Code

```
const fruits = ["apple", "banana", "orange", "banana"];

const index = fruits.indexOf("banana");


console.log(index); // 1
console.log(fruits.indexOf("not found")); // -1
```

- **splice() Method**: This method is used to add or remove elements from any position in an array. The return value for the splice() method will be an array of the items removed from the array. If nothing is removed, then an empty array will be returned. This method will mutate the original array, modifying it in place rather than creating a new array. The first argument specifies the index at which to begin modifying the array. The second argument is the number of elements you wish to remove. The following arguments are the elements you wish to add.

Example Code

```
const colors = ["red", "green", "blue"];

colors.splice(1, 0, "yellow", "purple");


console.log(colors); // ["red", "yellow", "purple", "green", "blue"]
```

- **includes() Method**: This method is used to check if an array contains a specific value. This method returns true if the array contains the specified element, and false otherwise.

Example Code

```
const programmingLanguages = ["JavaScript", "Python", "C++"];


console.log(programmingLanguages.includes("Python")); // true

console.log(programmingLanguages.includes("Perl")); // false
```

- **concat() Method**: This method creates a new array by merging two or more arrays.

Example Code

```
const programmingLanguages = ["JavaScript", "Python", "C++"];

const newList = programmingLanguages.concat("Perl");


console.log(newList); // ["JavaScript", "Python", "C++", "Perl"]
```

- **slice() Method**: This method returns a shallow copy of a portion of the array at a specified index or the entire array. A shallow copy will copy the reference to the array instead of duplicating it.

Example Code

```
const programmingLanguages = ["JavaScript", "Python", "C++"];

const newList = programmingLanguages.slice(1);


console.log(newList); // ["Python", "C++"]
```

- **Spread Syntax**: The spread syntax is used to create shallow copies of an array.

Example Code

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

const shallowCopiedArray = [...originalArray];


shallowCopiedArray.push(4);


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

console.log(shallowCopiedArray); // [1, 2, 3, 4]
```

- **split() Method**: This method divides a string into an array of substrings and specifies where each split should happen based on a given separator. If no separator is provided, the method returns an array containing the original string as a single element.

Example Code

```
const str = "hello";

const charArray = str.split("");


console.log(charArray); // ["h", "e", "l", "l", "o"]
```

- **reverse() Method**: This method reverses an array in place.

Example Code

```
const desserts = ["cake", "cookies", "pie"];

console.log(desserts.reverse()); // ["pie", "cookies", "cake"]
```

- **join() Method**: This method concatenates all the elements of an array into a single string, with each element separated by a specified separator. If no separator is provided, or an empty string ("") is used, the elements will be joined without any separator.

Example Code

```
const reversedArray = ["o", "l", "l", "e", "h"];

const reversedString = reversedArray.join("");


console.log(reversedString); // "olleh"
```