**ARRAYS**

Arrays in JavaScript are used to store multiple values in a single variable. They are a fundamental data structure in JavaScript and provide various features, properties, and methods for working with collections of data. Here's a detailed overview of arrays in JavaScript:

**Declaration:**

You can declare an array in JavaScript using square brackets **[]** and optionally initialize it with values separated by commas. For example:

let fruits = ['apple', 'banana', 'orange'];

**Features:**

1. **Ordered Collection**: Arrays in JavaScript maintain the order of elements they contain. Each element is associated with an index, starting from 0 for the first element, 1 for the second element, and so on.
2. **Dynamic Size**: JavaScript arrays are dynamic, meaning the number of elements can change dynamically after the array is created. You can add or remove elements from an array as needed.
3. **Heterogeneous Data Types**: JavaScript arrays can hold elements of any data type, including numbers, strings, objects, and even other arrays. This flexibility allows you to store mixed types of data in a single array.

**Properties:**

1. **length**: The **length** property of an array returns the number of elements in the array. It is automatically updated when you add or remove elements from the array.

let fruits = ['apple', 'banana', 'orange'];

console.log(fruits.length); // Output: 3

**Methods:**

Arrays in JavaScript come with a variety of built-in methods for manipulating array elements, iterating over arrays, and performing common tasks. Here are some commonly used array methods:

Array methods are built-in functions in JavaScript that allow you to perform various operations on arrays. These methods provide convenient ways to manipulate array elements, add or remove elements, iterate over arrays, and perform common tasks such as sorting and searching. Here's a brief overview of some commonly used array methods in JavaScript:

1. **push()**: Adds one or more elements to the end of an array and returns the new length of the array.

let fruits = ['apple', 'banana']; fruits.push('orange'); // fruits is now ['apple', 'banana', 'orange']

1. **pop()**: Removes the last element from an array and returns that element.

let fruits = ['apple', 'banana', 'orange']; let removedFruit = fruits.pop(); // fruits is now ['apple', 'banana'], removedFruit is 'orange'

1. **shift()**: Removes the first element from an array and returns that element.

let fruits = ['apple', 'banana', 'orange']; let removedFruit = fruits.shift(); // fruits is now ['banana', 'orange'], removedFruit is 'apple'

1. **unshift()**: Adds one or more elements to the beginning of an array and returns the new length of the array.

let fruits = ['banana', 'orange']; fruits.unshift('apple'); // fruits is now ['apple', 'banana', 'orange']

1. **slice()**: Returns a shallow copy of a portion of an array into a new array object selected from begin to end (end not included).

let fruits = ['apple', 'banana', 'orange', 'grape', 'kiwi']; let citrus = fruits.slice(2, 4); // citrus is now ['orange', 'grape']

1. **splice()**: Changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

let fruits = ['apple', 'banana', 'orange']; fruits.splice(1, 1, 'pear'); // fruits is now ['apple', 'pear', 'orange']

1. **forEach()**: Executes a provided function once for each array element.

let numbers = [1, 2, 3, 4, 5]; numbers.forEach(function(num) { console.log(num \* 2); }); // Output: 2, 4, 6, 8, 10

1. **map()**: Creates a new array populated with the results of calling a provided function on every element in the calling array.

let numbers = [1, 2, 3, 4, 5]; let doubledNumbers = numbers.map(function(num) { return num \* 2; }); // doubledNumbers is now [2, 4, 6, 8, 10]

1. **filter()**: Creates a new array with all elements that pass the test implemented by the provided function.

let numbers = [1, 2, 3, 4, 5]; let evenNumbers = numbers.filter(function(num) { return num % 2 === 0; }); // evenNumbers is now [2, 4]

1. **reduce()**: Executes a reducer function on each element of the array, resulting in a single output value.

let numbers = [1, 2, 3, 4, 5]; let sum = numbers.reduce(function(acc, curr) { return acc + curr; }, 0); // sum is now 15

These are just a few of the many array methods available in JavaScript. Understanding and mastering these methods will greatly enhance your ability to work with arrays efficiently in JavaScript.