Arrays in JavaScript

An array in JavaScript is a special type of object used to store multiple values in a single variable. Arrays can hold values of different data types, including strings, numbers, objects, and even other arrays.

Creating Arrays

Using Array Literal Syntax:

```
let fruits = ["apple", "banana", "cherry"];
console.log(fruits); // ["apple", "banana", "cherry"]
```

Using the Array Constructor:

```
let numbers = new Array(10, 20, 30);
console.log(numbers); // [10, 20, 30]
```

Empty Array:

```
let emptyArray = [];
console.log(emptyArray); // []
```

Accessing Array Elements

You can access elements using indices. The first element is at index 0.

```
let colors = ["red", "green", "blue"];
console.log(colors[0]); // "red"
console.log(colors[2]); // "blue"
```

Common Array Methods

push(): Adds one or more elements to the end of an array.

```
let arr = [1, 2, 3];
arr.push(4);
console.log(arr); // [1, 2, 3, 4]
```

pop(): Removes the last element from an array.

```
arr.pop();
console.log(arr); // [1, 2, 3]
```

shift(): Removes the first element from an array.

```
arr.shift();
console.log(arr); // [2, 3]
```

unshift(): Adds one or more elements to the beginning of an array.

```
arr.unshift(0);
console.log(arr); // [0, 2, 3]
```

splice(): Adds or removes elements from a specific index.

```
let animals = ["cat", "dog", "elephant"];
animals.splice(1, 1, "fox"); // Removes 1 element at index 1 and adds "fox"
console.log(animals); // ["cat", "fox", "elephant"]
```

slice(): Returns a shallow copy of a portion of an array.

```
let part = animals.slice(1, 3);
console.log(part); // ["fox", "elephant"]
```

concat(): Combines two or more arrays.

```
let arr1 = [1, 2];
let arr2 = [3, 4];
let combined = arr1.concat(arr2);
```

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

forEach(): Iterates over each array element.

```
combined.forEach((num) => console.log(num));
```

map(): Creates a new array by applying a function to each element.

```
let squared = combined.map((num) => num * num);

console.log(squared); // [1, 4, 9, 16]
```

filter(): Filters elements based on a condition.

```
let evenNumbers = combined.filter((num) => num % 2 === 0);

console.log(evenNumbers); // [2, 4]
```

reduce(): Reduces the array to a single value.

```
let sum = combined.reduce((total, num) => total + num, 0);

console.log(sum); // 10
```

find(): Returns the first element that satisfies the condition.

```
let found = combined.find((num) => num > 2);

console.log(found); // 3
```

includes(): Checks if an array contains a value.

```
console.log(combined.includes(3)); // true
```

Iterating Through Arrays

Using a for loop:

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

Using for...of loop:

```
for (let num of combined)
{
   console.log(num);
```

```
}
```

Using forEach():

```
combined.forEach((num) => console.log(num));
```

Notes

1. Arrays in JavaScript are dynamic, meaning you can change their size by adding or removing elements.
2. Arrays can hold mixed data types:
    a. let mixed = [1, "hello", true, null];
    b. console.log(mixed);
3. Arrays are zero-indexed.
4. JavaScript arrays are objects, so they come with additional properties and methods.

Functions in JavaScript

A function is a block of code designed to perform a specific task. Functions in JavaScript can take inputs, process them, and return outputs.

Creating a Function

1. **Function Declaration**

The traditional way of defining a function.

```
function greet(name)
{
  return `Hello, ${name}!`;
}
console.log(greet("Alice")); // "Hello, Alice!"
```

2. **Function Expression**

Functions can also be assigned to variables.

```
const add = function (a, b)
{
  return a + b;
};
console.log(add(3, 5)); // 8
```

3. **Arrow Function**

A concise way to define functions, introduced in ES6.

```
const multiply = (x, y) => x * y;
console.log(multiply(4, 6)); // 24
```

Calling a Function

You call (or invoke) a function by writing its name followed by parentheses. Pass arguments inside the parentheses.

```
function sayHi()
{
   console.log("Hi there!");
}
sayHi(); // Calls the function and logs "Hi there!" to the console.
```

Parameters and Arguments

Parameters: Variables listed in the function definition.

Arguments: Values passed to the function when calling it.

```
function calculateArea(length, width)
{
   return length * width;
}
console.log(calculateArea(5, 10)); // 50
```

Default Parameters

You can set default values for parameters if no value is provided during the function call.

```
function greet(name = "Guest")
{
   return `Welcome, ${name}!`;
}
```

```
console.log(greet()); // "Welcome, Guest!"
console.log(greet("John")); // "Welcome, John!"
```

Returning a Value

Use the return keyword to send a value back from the function.

```
function square(num)
{
   return num * num;
}
let result = square(7);
console.log(result); // 49
```

If no return is specified, the function returns undefined.

Anonymous Functions

Functions without a name are called anonymous functions. They are often used as callbacks.

```
setTimeout(function () {
   console.log("This message is delayed by 2 seconds.");
}, 2000);
```

Immediately Invoked Function Expression (IIFE)

IIFEs are executed immediately after they are defined.

```
 (function () {
console.log("I am an IIFE!");
})();
```

Example: Function to Check Even or Odd

```
function isEven(number) {
   if (number % 2 === 0) {
      return `${number} is even.`;
   } else {
      return `${number} is odd.`;
   }
}
console.log(isEven(4)); // "4 is even."
console.log(isEven(7)); // "7 is odd."
```

Notes

First-Class Functions: Functions in JavaScript are treated as first-class citizens, meaning they can be:

- Assigned to variables
- Passed as arguments
- Returned from other functions

Arrow Functions and this: Arrow functions don't have their own this context. Instead, they inherit it from the enclosing scope.

Scope: Functions have their own local scope. Variables defined inside a function cannot be accessed outside of it.

```
function testScope()
{
   let localVar = "I am local!";
   console.log(localVar);
}
testScope(); // Logs: "I am local!"
console.log(localVar); // Error: localVar is not defined
```