Day 2: Functions, Arrays, and Objects in JavaScript

1. Functions in JavaScript

Function Definition

Functions are blocks of reusable code that perform specific tasks.
Functions can have **parameters** (input values) and **arguments** (actual values passed to parameters).

Syntax:

```
function greet(name) {
    console.log("Hello, " + name + "!");
}
greet("Alice");  // Output: Hello, Alice!
```

Function Scope and Hoisting

- **Scope**: Variables declared inside a function are **local** to that function.
- **Hoisting**: JavaScript moves function declarations to the top of the scope before execution.

Example of Hoisting:

```
sayHello();  // Output: Hello!

function sayHello() {
    console.log("Hello!");
}
```

(Even though sayHello() is called before it's declared, it works due to hoisting.)

---

2. Function Expressions

A function can be stored in a variable.

Example:

```
const add = function(x, y) {
    return x + y;
};
console.log(add(5, 3));  // Output: 8
```

(Unlike function declarations, function expressions are NOT hoisted.)

---

3. Arrow Functions

A shorter syntax for writing functions.

Syntax:

```
const multiply = (a, b) => a * b;
console.log(multiply(4, 5));  // Output: 20
```

Arrow functions are useful for concise syntax and do not have their own this context.

---

4. Return Values

Functions can return values using the return keyword.

Example:

```
function square(num) {
    return num * num;
}
let result = square(4);
console.log(result);  // Output: 16
```

Arrays in JavaScript

An **array** is a collection of values stored in a single variable.

Creating and Accessing Arrays

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits[0]);  // Output: Apple
```

Array Methods

1. **Adding and Removing Elements**

   ```
   fruits.push("Orange");   // Adds at the end
   fruits.pop();            // Removes from the end
   fruits.unshift("Grapes"); // Adds at the beginning
   fruits.shift();          // Removes from the beginning
   ```

2. **Splice & Slice**

   ```
   let colors = ["Red", "Green", "Blue"];
   colors.splice(1, 1, "Yellow"); // Removes "Green" and adds "Yellow"
   console.log(colors);  // Output: ["Red", "Yellow", "Blue"]

   let slicedColors = colors.slice(0, 2);
   console.log(slicedColors);  // Output: ["Red", "Yellow"]
   ```

3. **Higher-Order Array Methods**

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

   let squaredNumbers = numbers.map(num => num * num);
   console.log(squaredNumbers);  // Output: [1, 4, 9, 16, 25]

   let evenNumbers = numbers.filter(num => num % 2 === 0);
   console.log(evenNumbers);  // Output: [2, 4]
   ```

```
        let sum = numbers.reduce((acc, num) => acc + num, 0);
        console.log(sum);  // Output: 15
```

Iterating Over Arrays

```
for (let fruit of fruits) {
    console.log(fruit);
}
```

Objects in JavaScript

Objects store data in **key-value** pairs.

Creating Objects

```
let person = {
    name: "John",
    age: 30,
    city: "New York"
};
console.log(person.name);  // Output: John
```

Accessing Properties

- **Dot Notation**: person.name
- **Bracket Notation**: person["age"]

Modifying and Deleting Properties

```
        person.age = 31;      // Modify
        person.country = "USA"; // Add
        delete person.city;   // Delete
```

Object Methods

```
let student = {
    name: "Alice",
    greet: function() {
        console.log("Hello, " + this.name + "!");
    }
};
student.greet();  // Output: Hello, Alice!
```
