3-Functions

Chapter 3: Working with Functions

Functions are one of the most important concepts in JavaScript. They allow you to organize your code into reusable blocks, making your programs more efficient and easier to maintain.

3.1 What are Functions?

A function is a block of code designed to perform a specific task. You can think of it as a machine: you give it some inputs, it processes them, and returns an output.

Why Use Functions?

- Reusability: Write once, use multiple times.
- Organization: Make your code cleaner and more modular.
- Maintenance: Easily update and debug code.

3.2 Creating and Calling Functions

To define a function, use the `function` keyword followed by the function name, parentheses, and curly braces.

Syntax:

```
function functionName(parameters) {
  // Code to execute
  return value; // Optional
}
```

- **Function Name:** Describes what the function does.
- **Parameters:** Placeholders for input values.
- **Return Statement:** Outputs the result (optional).

Example:

```
```

```
function greet(name) {
  return "Hello, " + name + "!";
}

console.log(greet("Rick")); // Output: "Hello, Rick!"
```

3.3 Parameters and Arguments

Functions can take inputs (parameters) and process them. Arguments are the actual values passed when calling the function.

Example with Multiple Parameters:

```
function add(a, b) {
  return a + b;
}

console.log(add(5, 3)); // Output: 8
```

- Parameters: `a` and `b` (placeholders).
- Arguments: `5` and `3` (actual values).

3.4 Function Expressions

In JavaScript, functions can also be assigned to variables. These are called function expressions.

Example:

```
const multiply = function(a, b) {
  return a * b;
};

console.log(multiply(4, 5)); // Output: 20
```

3.5 Arrow Functions

Arrow functions provide a shorter syntax for writing functions. They are especially useful for concise one-liners.

Syntax:

```
const functionName = (parameters) => expression;
```

Example:

```
const subtract = (a, b) => a - b;

console.log(subtract(10, 4)); // Output: 6
```

3.6 Default Parameters

Sometimes, you might want to set default values for parameters. JavaScript allows you to do this.

Example:

```
function greet(name = "Guest") {
  return "Hello, " + name + "!";
}

console.log(greet());          // Output: "Hello, Guest!"
console.log(greet("Rick"));    // Output: "Hello, Rick!"
```

3.7 Nested Functions

Functions can be defined inside other functions. Inner functions have access to variables of their parent function.

Example:

```
function outerFunction(outerVar) {
  function innerFunction(innerVar) {
    return outerVar + innerVar;
  }
  return innerFunction;
```

```
}

const addTen = outerFunction(10);
console.log(addTen(5)); // Output: 15
```

3.8 Higher-Order Functions

A higher-order function takes another function as an argument or returns a function.

Example:

```
function applyFunction(func, value) {
  return func(value);
}

function double(num) {
  return num * 2;
}

console.log(applyFunction(double, 4)); // Output: 8
```

3.9 Practical Examples

Let's apply what we've learned in real-world scenarios.

1. **Simple Temperature Converter:**

   ```
   function convertToFahrenheit(celsius) {
     return celsius * 9 / 5 + 32;
   }

   console.log(convertToFahrenheit(0));  // Output: 32
   console.log(convertToFahrenheit(100)); // Output: 212
   ```

2. **Generate a Random Number:**

   ```
   function getRandom(min, max) {
     return Math.floor(Math.random() * (max - min + 1)) + min;
   }

   ```

```
console.log(getRandom(1, 10)); // Random number between 1 and 10
```

3. **Simple To-Do List Manager:**

```
let tasks = [];

function addTask(task) {
  tasks.push(task);
}

function listTasks() {
  return tasks.join(", ");
}

addTask("Learn functions");
addTask("Write a program");
console.log(listTasks()); // Output: "Learn functions, Write a program"
```