Functions in JavaScript

Instructor: Pramod Kumar Jena

Part 1: Introduction to Functions

What is a Function?

A **function** is a block of reusable code designed to perform a specific task. It helps in making code modular, organised, and easier to maintain.

Basic Syntax of a Function:

```
function functionName(parameters) {
  // code to be executed
  return value; // optional
}
```

- **Function Name**: A unique identifier for the function.
- **Parameters**: Values that can be passed to the function when calling it.
- **Return**: A function can return a result to the code that called it.

Example 1: Simple Function

Let's start with a basic function that adds two numbers.

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

let result = addNumbers(5, 7);
console.log(result);  // Output: 12
```

Explanation:

- The function `addNumbers` takes two parameters `a` and `b`.
- It returns the sum of `a` and `b`.

- When we call the function with `addNumbers(5, 7)`, it returns 12.

Why Use Functions?

- **Code Reusability**: Write once, use multiple times.
- **Modularity**: Break the code into smaller chunks.
- **Maintainability**: Easier to maintain and debug.

Real-Life Example 1: Greeting Function

A function can be used to greet a user based on the time of day.

```
function greetUser(name, hour) {
  if (hour < 12) {
    return `Good morning, ${name}!`;
  } else if (hour < 18) {
    return `Good afternoon, ${name}!`;
  } else {
    return `Good evening, ${name}!`;
  }
}

let greeting = greetUser('John', 10);
console.log(greeting);  // Output: Good morning, John!
```

Exercise 1: Simple Calculator Function

Write a function called `calculate` that takes three parameters: two numbers and an operator (`+`, `-`, `*`, `/`). The function should return the result of the operation.

Hint:

- Use an `if` or `switch` statement to handle the operator.

```
function calculate(a, b, operator) {
  switch (operator) {
    case '+':
      return a + b;
    case '-':
      return a - b;
    case '*':
      return a * b;
    case '/':
      return a / b;
    default:
      return 'Invalid operator';
  }
}

console.log(calculate(10, 5, '+'));  // Output: 15
```

Part 2: Types of Functions

1. Function Declarations

A **function declaration** is the standard way of creating a function. These functions are hoisted, meaning they can be called before they are defined in the code.

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

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

---

2. Function Expressions

A **function expression** is when a function is assigned to a variable. These functions are not hoisted and can only be called after they are defined.

```
let sayHello = function() {
  console.log('Hello!');
};

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

---

3. Arrow Functions (ES6)

Arrow functions provide a more concise syntax and are especially useful in situations like callbacks or functional programming.

```
const multiply = (a, b) => a * b;

console.log(multiply(5, 3));  // Output: 15
```

Real-Life Example 2: Temperature Converter

Create a function that converts temperatures between Celsius and Fahrenheit.

```
function convertTemperature(temp, unit) {
  if (unit === 'C') {
    return (temp * 9/5) + 32;  // Convert Celsius to Fahrenheit
  } else if (unit === 'F') {
    return (temp - 32) * 5/9;  // Convert Fahrenheit to Celsius
  } else {
    return 'Invalid unit';
  }
}

console.log(convertTemperature(30, 'C'));  // Output: 86
console.log(convertTemperature(86, 'F'));  // Output: 30
```

Exercise 2: BMI Calculator Function

Create a function `calculateBMI` that takes weight (in kg) and height (in meters) and returns the BMI (Body Mass Index).

Hint:

- BMI formula: `BMI = weight / (height * height)`.

```
function calculateBMI(weight, height) {
  let bmi = weight / (height * height);
  return bmi.toFixed(2);  // Return BMI with two decimal places
}

console.log(calculateBMI(70, 1.75));  // Output: 22.86
```

Part 3: Advanced Function Concepts

1. Default Parameters

In JavaScript, you can provide default values for function parameters.

```
function greet(name = 'Guest') {
  console.log(`Hello, ${name}!`);
}

greet();  // Output: Hello, Guest!
greet('Alice');  // Output: Hello, Alice!
```

---

2. Rest Parameters

Rest parameters allow you to pass an indefinite number of arguments to a function.

```
function sumAll(...numbers) {
  return numbers.reduce((sum, current) => sum + current, 0);
}

console.log(sumAll(1, 2, 3, 4));  // Output: 10
```

---

3. Callback Functions

A **callback function** is a function passed as an argument to another function, and it is executed inside that function.

```
function greet(callback) {
  console.log('Greeting...');
  callback();
}

function sayHello() {
  console.log('Hello, world!');
}

greet(sayHello);  // Output: Greeting... Hello, world!
```

---

4. Higher-Order Functions

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

```
function multiplyBy(factor) {
  return function(number) {
    return number * factor;
  };
}

let multiplyBy2 = multiplyBy(2);
console.log(multiplyBy2(5));  // Output: 10
```

Real-Life Example 3: Authentication System

Let’s simulate an authentication system using a higher-order function that takes a callback.

```
function authenticate(user, callback) {
  if (user === 'admin') {
    callback(true);
  } else {
    callback(false);
  }
}

authenticate('admin', function(isAuthenticated) {
  if (isAuthenticated) {
    console.log('Welcome, Admin!');
  } else {
    console.log('Access Denied.');
  }
});
// Output: Welcome, Admin!
```

Exercise 3: Password Validator

Write a function called `passwordValidator` that takes a password as input and checks if it meets certain criteria (minimum 8 characters, contains a number, and a special character).

Hint:

- Use regular expressions for pattern matching.

```
function passwordValidator(password) {
  let hasNumber = /\d/;
  let hasSpecialChar = /[!@#\$%\^\&*\)\(+=._-]/;

  if (password.length >= 8 && hasNumber.test(password) &&
hasSpecialChar.test(password)) {
    return 'Password is valid!';
  } else {
    return 'Password is invalid!';
  }
}
```

```
console.log(passwordValidator('Password123!'));  // Output: Password
is valid!
```

Exercise 4: Countdown Timer Function

Create a function that acts as a countdown timer. The function should take a number (seconds) and print the countdown to zero every second.

Hint:

- Use `setInterval` to create the countdown.

```
function countdown(seconds) {
  let interval = setInterval(() => {
    console.log(seconds);
    seconds--;

    if (seconds < 0) {
      clearInterval(interval);
      console.log('Countdown finished!');
    }
  }, 1000);
}

countdown(5);
// Output: 5, 4, 3, 2, 1, Countdown finished!
```

Exercise 5: Tip Calculator

Write a function `tipCalculator` that takes a bill amount and a tip percentage, then returns the total bill including tip.

Hint:

- Formula: `Total = bill + (bill * tipPercentage / 100)`

```
function tipCalculator(bill, tipPercentage) {
  return bill + (bill * tipPercentage / 100);
}

console.log(tipCalculator(1000, 10));  // Output: 1100
```

Wrap-up and Conclusion

- **Functions** help in making your code modular, maintainable, and reusable.
- There are different types of functions, such as function declarations, expressions, arrow functions, and higher-order functions.
- Advanced concepts like default parameters, rest parameters, and callbacks enhance the flexibility of functions.
