Chapter 4

Functions and Scope

**Syntax of a Function Declaration**

function functionName(parameter1, parameter2, ...) {

// function body

// optional: return value

}

**Example of a Function Definition**

function greet(name) {

console.log("Hello, " + name + "!");

}

**Another Example**

function displayDate() {

let currentDate = new Date(); // Get the current date and time

console.log("Today's date is: " + currentDate.toDateString());

}

// Calling the function

displayDate();

**Function with Argument**

**Syntax:**

function functionName(parameters) {

// Function body: code to be executed

}

**Example:**

function greet(name) {

console.log(`Hello, ${name}!`);

}

// Invoking the function

greet("Rupali"); // Output: Hello,Rupali!

### **Invoking Functions**

greet("Rupali"); // Outputs: Hello, Rupali!

**Example 1**

function greetCustomer(customerName, isMember) {

if (isMember) {

console.log("Welcome back, " + customerName + "! Thank you for being a member.");

} else {

console.log("Hello, " + customerName + "! Consider joining our membership for benefits.");

}

}

greetCustomer("Rupali", true); // Welcome back, Rupali! ...

greetCustomer("Radha", false); // Hello, Radha! Consider joining...

**Example 2**

function calculateCircleArea(radius) {

const pi = 3.14159; // Constant value of pi

let area = pi \* radius \* radius; // Formula for area of a circle

console.log("The area of the circle is: " + area);

}

// Calling the function

calculateCircleArea(5);

calculateCircleArea(10);

**Output**

The area of the circle is: 78.53975

The area of the circle is: 314.159

**Function Declarations**

**Example**

console.log(square(5)); // 25

function square(num) {

return num \* num;

}

console.log(sum(2, 3)); // Output: 5

function sum(a, b) {

return a + b;

}

**Function Expressions**

**Example**

const multiply = function(a, b) {

return a \* b;

};

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

**Key Differences**

* **Hoisting:** Declarations are hoisted, expressions are not.
* **Usage:** Expressions allow anonymous functions and dynamic assignments.

**Syntax of Arrow Function**

const functionName = (parameters) => {

// function body

}

**1. Basic Arrow Function**

const greet = (name) => {

return 'Hello, ${name}!';

};

console.log(greet('RAM')); // Output: Hello, RAM!

**2. Arrow Function with a Single Parameter (No Parentheses)**

const square = x => x \* x;

console.log(square(4)); // Output: 16

**3. Arrow Function with No Parameters**

const sayHello = () => 'Hello!';

console.log(sayHello()); // Output: Hello!

**4. Arrow Function with a Block Body (Multiple Statements)**

const add = (a, b) => {

let sum = a + b;

return sum;

};

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

**Example of this in arrow functions:**

function Person(name) {

this.name = name;

setTimeout(() => {

console.log(this.name); // 'this' refers to the Person object

}, 1000);

}

const person = new Person('Radhe');

// Output after 1 second: Radhe

* Example with rest parameters:

const sum = (...args) => args.reduce((a, b) => a + b, 0);

console.log(sum(1, 2, 3)); // Output: 6

**Example (Incorrect use of arrow function as a constructor):**

const Person = (name) => {

this.name = name; // Error: 'this' is not defined

};

const person = new Person('Radhe'); // TypeError: Person is not a constructor

**Parameters and Arguments**

**Parameters**

**Example**

function greet(name) {

console.log(`Hello, ${name}!`);

}

**Arguments**

**Example**

greet("Radha"); // Radha is the argument

**Default Parameters**

**Example**

function greet(name = "Guest") {

console.log(`Hello, ${name}!`);

}

greet(); // Output: Hello, Guest!

**Syntax of Rest Parameters**

function exampleFunction(a, b, ...rest) {

console.log(a); // First argument

console.log(b); // Second argument

console.log(rest); // Array of remaining arguments

}

**Example:**

function sum(...numbers) {

return numbers.reduce((a, b) => a + b, 0);

}

console.log(sum(1, 2, 3)); // Output: 6

**Example 1: Basic Usage**

function sum(...numbers) {

return numbers.reduce((total, num) => total + num, 0);

}

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

console.log(sum(5, 10)); // Output: 15

**Example 2: Combining Rest Parameters with Regular Parameters**

function greet(greeting, ...names) {

console.log(`${greeting}, ${names.join(", ")}`);

}

greet("Hello", "Rupali", "Mahesh", "Chopade");

// Output: Hello, Rupali, Mahesh, Chopade

**Example 3: Default Values with Rest Parameters**

function multiply(multiplier = 1, ...numbers) {

return numbers.map(num => num \* multiplier);

}

console.log(multiply(2, 1, 2, 3)); // Output: [2, 4, 6]

console.log(multiply(undefined, 4, 5)); // Output: [4, 5]

**Example 4: Using Rest Parameters in Arrow Functions**

const concatenateStrings = (...strings) => strings.join(" ");

console.log(concatenateStrings("JavaScript", "is", "awesome!"));

// Output: JavaScript is awesome!

**Example 5: Forwarding Arguments with Rest Parameters**

function logMessage(message, ...args) {

console.log(message, ...args);

}

logMessage("Values:", 1, 2, 3, 4);

// Output: Values: 1 2 3 4

**Example 6: Rest Parameter Limitation**

**Valid:**

function example(a, b, ...rest) {

console.log(rest);

}

**Invalid:**

function example(...rest, a) {

// Syntax Error: Rest parameter must be last formal parameter

}

**Syntax of the return Statement**

function functionName(parameters) {

// Function logic

return value;

}

**Example 1: Basic Return Statement**

function add(a, b) {

return a + b;

}

let result = add(5, 3);

console.log(result); // Output: 8

**Example 2: Function Without a return Statement**

function greet(name) {

console.log(`Hello, ${name}!`);

}

let result = greet("Alice");

console.log(result); // Output: undefined

**Example 3: Returning an Object**

function createPerson(name, age) {

return { name: name, age: age };

}

let person = createPerson("Anuja", 30);

console.log(person); // Output: { name: 'Anuja', age: 30 }

**Example 4: Returning a Function**

function multiplier(factor) {

return function(number) {

return number \* factor;

};

}

let double = multiplier(2);

console.log(double(5)); // Output: 10

**Example 5: Early Return**

function checkNumber(num) {

if (num < 0) {

return "Negative number";

}

return "Positive number";

}

console.log(checkNumber(-5)); // Output: Negative number

console.log(checkNumber(10)); // Output: Positive number

**Example 6: Conditional Return**

function calculateGrade(score) {

if (score >= 90) {

return "A";

} else if (score >= 80) {

return "B";

} else if (score >= 70) {

return "C";

} else {

return "F";

}

}

console.log(calculateGrade(85)); // Output: B

console.log(calculateGrade(65)); // Output: F

**Example 7: Function Performing an Action**

function logMessage(message) {

console.log(message);

}

logMessage("This is a log message."); // Output: This is a log message.

**Example 8: Function Chaining**

function add(a, b) {

return a + b;

}

function square(num) {

return num \* num;

}

let result = square(add(2, 3)); // Adds first, then squares the result

console.log(result); // Output: 25

**Example 9: Mistake Example**

function add(a, b) {

console.log(a + b); // This logs the sum but does not return it

}

let result = add(2, 3);

console.log(result); // Output: undefined

**Corrected Example**

function add(a, b) {

return a + b; // Return the sum

}

let result = add(2, 3); // Call the function and store the returned value

console.log(result); // Output: 5

**Combined Example**

function add(a, b) {

let sum = a + b;

console.log(sum); // Log the sum

return sum; // Return the sum

}

let result = add(2, 3); // Store the returned value

console.log(result); // Output: 5

**Example**

let globalVar = "I am global";

function showGlobal() {

console.log(globalVar); // Accessible here

}

showGlobal(); // Output: I am global

console.log(globalVar); // Output: I am global

**Example:**

function myFunction() {

var localVar = "I am local";

console.log(localVar); // Output: I am local

}

myFunction();

console.log(localVar); // Error: localVar is not defined

**Block Scope**

if (true) {

let blockScoped = "I am block-scoped";

console.log(blockScoped); // Output: I am block-scoped

}

console.log(blockScoped); // Error: blockScoped is not defined

**Example:**

let outerVar = "Outer";

function outerFunction() {

let innerVar = "Inner";

console.log(outerVar); // Accessible

console.log(innerVar); // Accessible

}

outerFunction();

console.log(innerVar); // Error: innerVar is not defined

**Example:**

let a = "Global";

function first() {

let b = "Local";

function second() {

console.log(a); // Searches in global scope: Output: Global

console.log(b); // Searches in parent scope: Output: Local

}

second();

}

first();

**Example of Lexical Scoping**

function outerFunction() {

let outerVariable = "I am from the outer scope";

function innerFunction() {

console.log(outerVariable); // Accessible due to lexical scoping

}

innerFunction();

}

outerFunction(); // Output: I am from the outer scope

**Example of a Closure**

function outerFunction() {

let count = 0;

return function increment() {

count++;

console.log(count); // Accesses count even after outerFunction has finished

};

}

const counter = outerFunction(); // outerFunction is executed

counter(); // Output: 1

counter(); // Output: 2

**Example 1: Data Encapsulation**

function createCounter() {

let count = 0;

return {

increment: function () {

count++;

console.log(count);

},

decrement: function () {

count--;

console.log(count);

},

};

}

const counter = createCounter();

counter.increment(); // Output: 1

counter.decrement(); // Output: 0

**Example 2: Currying with Closures**

function multiply(a) {

return function (b) {

return a \* b;

};

}

const double = multiply(2);

console.log(double(5)); // Output: 10

console.log(double(10)); // Output: 20

1. **Memory Leaks**:

function leakyFunction() {

let largeData = new Array(1000000); // Large array

return function () {

console.log(largeData.length);

};

}

const leak = leakyFunction();

// largeData remains in memory as long as leak exists

1. **Unintended Shared State**:

function createFunctions() {

let funcs = [];

for (let i = 0; i < 3; i++) {

funcs.push(function () {

console.log(i);

});

}

return funcs;

}

let functions = createFunctions();

functions[0](); // Output: 3

functions[1](); // Output: 3

functions[2](); // Output: 3

**Solution**: Use block-scoped variables (let) or an IIFE.

for (let i = 0; i < 3; i++) {

funcs.push(((j) => () => console.log(j))(i));

}

**Structure of an IIFE**

**Syntax**

(function () {

// Code to execute immediately

})();

**Examples of IIFE**

**Example 1: Basic IIFE**

(function () {

console.log("IIFE executed!");

})();

**Output**

IIFE executed!

**Example 2: IIFE with Parameters**

(function (name) {

console.log("Hello, " + name + "!");

})("John");

**Output**

Hello, John!

**Example 3: Assigning IIFE to a Variable**

let result = (function () {

let x = 10;

return x \* x;

})();

console.log(result); // Output: 100

**Benefits of IIFEs**

(function () {

let privateVar = "I am private!";

console.log(privateVar);

})();

// console.log(privateVar); // Error: privateVar is not defined

1. **Avoid Global Scope Pollution**:

let myVar = "Global";

(function () {

let myVar = "Local";

console.log(myVar); // Output: Local

})();

console.log(myVar); // Output: Global

1. **Useful for Initialization**:

(function () {

let appConfig = {

version: "1.0",

author: "John Doe",

};

console.log("App initialized with version " + appConfig.version);

})();

1. **Create a Module-Like Structure**:

let counterModule = (function () {

let count = 0;

return {

increment: function () {

count++;

console.log(count);

},

decrement: function () {

count--;

console.log(count);

},

};

})();

counterModule.increment(); // Output: 1

counterModule.increment(); // Output: 2

counterModule.decrement(); // Output: 1

**Example of Block Scoping Replacing IIFE**

{

let privateVar = "Scoped Variable";

console.log(privateVar);

}

// console.log(privateVar); // Error: privateVar is not defined

**Common Use Cases for IIFEs**

1. **One-Time Execution**:

(function () {

console.log("One-time initialization");

})();

1. **Creating Isolated Environments**:

(function () {

let scriptSpecificVar = "Unique to this script";

console.log(scriptSpecificVar);

})();

* 1. **Passing Data Safely**:

(function (global) {

global.appName = "MyApp";

})(window);

console.log(window.appName); // Output: MyApp

**Syntax**

function recursiveFunction(parameters) {

if (baseCondition) {

return baseResult; // Base case

} else {

return recursiveFunction(smallerProblem); // Recursive call

}

}

**Example: Factorial Calculation**

**Formula**  
n ! = n × (n−1) !

Where 0 ! = 1 (base case).

**Code**

function factorial(n) {

if (n === 0) { // Base case

return 1;

} else {

return n \* factorial(n - 1); // Recursive call

}

}

console.log(factorial(5)); // Output: 120

**Example: Fibonacci Sequence**

F(n) = F(n−1) + F(n−2)

Where F(0) = 0 and F(1) = 1

**Code**:

function fibonacci(n) {

if (n === 0) { // Base case

return 0;

} else if (n === 1) { // Base case

return 1;

} else {

return fibonacci(n - 1) + fibonacci(n - 2); // Recursive call

}

}

console.log(fibonacci(6)); // Output: 8

**Memoized Fibonacci Example**

function memoizedFibonacci() {

const cache = {};

return function fib(n) {

if (n in cache) return cache[n];

if (n === 0) return 0;

if (n === 1) return 1;

cache[n] = fib(n - 1) + fib(n - 2);

return cache[n];

};

}

const fib = memoizedFibonacci();

console.log(fib(50)); // Output: 12586269025 (fast and efficient)

**Syntax**

function outerFunction() {

let outerVariable = "I'm from the outer function";

function innerFunction() {

console.log(outerVariable); // Access outer variable

}

innerFunction(); // Call inner function

}

outerFunction();

**Output**  
I'm from the outer function

**Example 1: Summation Using Nested Functions**

function calculateSum(a, b) {

function add() {

return a + b; // Access parameters from the outer function

}

return add();

}

console.log(calculateSum(5, 7)); // Output: 12

**Example 2: Nested Functions with Parameters**

function greetUser(greeting) {

function personalize(name) {

return `${greeting}, ${name}!`; // Combine outer and inner parameters

}

return personalize;

}

const sayHello = greetUser("Hello");

console.log(sayHello("Alice")); // Output: Hello, Alice!

**Example 3: Returning a Nested Function (Closure)**

function multiplier(x) {

return function(y) {

return x \* y; // Access variable x from the outer function

};

}

const double = multiplier(2);

console.log(double(5)); // Output: 10

**Example 4: Using Nested Functions for Validation**

function validateInput(input) {

function isNotEmpty(value) {

return value.trim().length > 0;

}

function isAlphanumeric(value) {

return /^[a-z0-9]+$/i.test(value);

}

return isNotEmpty(input) && isAlphanumeric(input);

}

console.log(validateInput("Valid123")); // Output: true

console.log(validateInput(" ")); // Output: false

**Example 5: Nested Functions for Recursion**

function factorial(n) {

function calculate(num) {

if (num === 0) return 1; // Base case

return num \* calculate(num - 1); // Recursive call

}

return calculate(n);

}

console.log(factorial(5)); // Output: 120

**Example 6: Scope and Nested Functions**

let globalVar = "I'm global";

function outerFunction() {

let outerVar = "I'm in outer";

function innerFunction() {

let innerVar = "I'm in inner";

console.log(globalVar); // Access global scope

console.log(outerVar); // Access outer scope

console.log(innerVar); // Access inner scope

}

innerFunction();

}

outerFunction();

/\*

Output:

I'm global

I'm in outer

I'm in inner

\*/

**Syntax of Anonymous Functions**

const myFunction = function() {

console.log("This is an anonymous function.");

};

myFunction(); // Output: This is an anonymous function.

**Example 1: Assigning to a Variable**

const greet = function() {

console.log("Hello, World!");

};

greet(); // Output: Hello, World!

**Example 2: Anonymous Function as a Callback**

setTimeout(function() {

console.log("Executed after 2 seconds");

}, 2000);

// Output: Executed after 2 seconds (after a 2-second delay)

**Example 3: Anonymous Function in Array Methods**

const numbers = [1, 2, 3, 4, 5];

const squares = numbers.map(function(num) {

return num \* num;

});

console.log(squares); // Output: [1, 4, 9, 16, 25]

**Example 4: Anonymous Function in Event Listener**

document.getElementById("myButton").addEventListener("click", function() {

console.log("Button clicked!");

});

Example 5: Anonymous Function in IIFE

(function() {

console.log("IIFE executed immediately!");

})(); // Output: IIFE executed immediately!

**Example 6: Named vs. Anonymous Functions**

**Named Function**

function add(a, b) {

return a + b;

}

console.log(add(2, 3)); // Output: 5

**Anonymous Function**

const add = function(a, b) {

return a + b;

};

console.log(add(2, 3)); // Output: 5