**Practical Number 4**

**Aim:** Functions, Scope & Error Handling: Create and invoke functions using declarations, expressions, and arrow syntax. Explore scope and closures. Include try-catch blocks to handle input errors. Write a function to reverse a number and check for palindrome.

**Part 1:** Functions (declarations, expressions, arrow functions) + Scope + Closures  
**Part 2:** Error Handling + Reverse Number & Palindrome problem

## **Part 1 – Functions, Scope, and Closures**

### **1. Function Declarations**

**Definition:** A function declaration defines a named function. It can be called **before or after** it is defined (due to hoisting).

**Syntax:**

**function** functionName(parameters) {

// function body

}

**Examples:**

// Example 1

function greet() {

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

}

greet();

// Example 2 – with parameters

function add(a, b) {

return a + b;

}

console.log(add(5, 3));

// Example 3 – returning strings

function welcome(name) {

return `Welcome, ${name}!`;

}

console.log(welcome("Alice"));

// Example 4 – default parameter

function multiply(a, b = 2) {

return a \* b;

}

console.log(multiply(5));

### **2. Function Expressions**

**Definition:  
Functions can be stored in variables.** These are **not hoisted**, so they must be defined before use.

**Syntax:**

const functionName = function(parameters) {

// function body

};

**Examples:**

// Example 1

const greet = function() {

console.log("Hello from expression!");

};

greet();

// Example 2 – sum

const sum = function(a, b) {

return a + b;

};

console.log(sum(4, 6));

// Example 3 – anonymous inline

setTimeout(function() {console.log("Executed after 2 seconds");}, 2000);

### **Explanation**

* setTimeout() is a **built-in JavaScript function** that executes code **after a given delay** (in milliseconds).
* The **first argument** here is a **function expression** without a name → **anonymous function**.
* The **second argument** is 2000, meaning **2 seconds** delay.
* After 2 seconds, JavaScript calls the anonymous function and prints "Executed after 2 seconds".

**Why anonymous?**We don’t give it a name because we don’t need to reuse it — it’s just executed once after the delay.

// Example 4 – named function expression

const factorial = function fact(n) {

return n <= 1 ? 1 : n \* fact(n - 1);

};

console.log(factorial(5));

### **Explanation**

* This is a **function expression** stored in a constant variable factorial.
* The function **has a name** fact even though we store it in a variable.  
   This is called a **named function expression**.
* Why name it?  
   In **recursive functions**, the name lets the function **call itself** inside its own body.
* Logic:
  + If n <= 1, return 1 (base case).
  + Else return n \* fact(n - 1) (recursive call).
* Calling factorial(5):
  + 5 \* fact(4) → 4 \* fact(3) → 3 \* fact(2) → 2 \* fact(1) → returns 1
  + Result = 120.

**Why not just use factorial inside instead of fact?** If you renamed the variable later or passed the function around, the internal name fact would still work for recursion — it’s more stable inside the function body.

### **3. Arrow Functions**

**Definition:** Shorter syntax for writing functions. They do **not** bind their own this.

**Syntax:**

const functionName = (parameters) => {

// function body

};

**Examples:**

// Example 1

const greet = () => console.log("Hi from arrow function!");

greet();

// Example 2 – single parameter

const square = n => n \* n;

console.log(square(4));

// Example 3 – multiple parameters

const add = (a, b) => a + b;

console.log(add(5, 7));

p// Example 4 – multiline

const isEven = (num) => {

if (num % 2 === 0) return true;

return false;

};

console.log(isEven(8));

### **4. Scope**

**Definition:** Scope determines where variables can be accessed.

* **Global Scope:** Accessible anywhere.
* **Function Scope:** Accessible inside the function only.
* **Block Scope:** Accessible only inside { } if declared with let or const.

**Example:**

let globalVar = "I am global";

function testScope() {

let functionVar = "I am in a function";

console.log(globalVar); // Accessible

console.log(functionVar); // Accessible

}

testScope();

// console.log(functionVar); // Error

{

let blockVar = "I am block-scoped";

console.log(blockVar); // Accessible

}

// console.log(blockVar); // Error

### **5. Closures**

**Definition:** A closure is created when an inner function remembers variables from its outer function **even after the outer function has finished**.

**Example 1 – Counter**

function createCounter() {

let count = 0;

return function() {

count++;

return count;

};

}

const counter = createCounter();

console.log(counter()); // 1

console.log(counter()); // 2

**Example 2 – Private Data**

function bankAccount(initialBalance) {

let balance = initialBalance;

return {

deposit(amount) {

balance += amount;

return balance;

},

withdraw(amount) {

balance -= amount;

return balance;

}

};

}

const account = bankAccount(100);

console.log(account.deposit(50)); // 150

console.log(account.withdraw(30)); // 120

### **Student Tasks for Part 1**

1. Write a function declaration that takes two numbers and returns the greater number.
2. Write a function expression that calculates the factorial of a number.
3. Write an arrow function that checks if a given string contains the word "JavaScript".
4. Create a closure that stores a secret message and returns it only when a password matches.