Variables and Data Types

Question 1: What are variables in JavaScript? How do you declare a variable using var, let,and const?

* **Variables** are used to store data values in JavaScript.
* There are **three ways** to declare variables:
* **var** (Old Way)
* **let** (Modern Way)
* **const** (Constant Values)

1. **Using var**

* var is **function-scoped**.
* It **can be redeclared** and **reassigned**.

**2. Using let**

* let is **block-scoped** (limited to the block {} where it is defined).
* It **can be reassigned** but **cannot be redeclared** within the same scope.

**3. Using const**

* const is also **block-scoped**.
* It **cannot be reassigned or redeclared**.
* Must be **initialized with a value** when declared.

Question 2: Explain the different data types in JavaScript. Provide examples for each.

**1. Primitive Data Types**

* **String** → Text data
* **Number** → Numeric data (Integer, Float)
* **Boolean** → true or false
* **Undefined** → Variable declared but not assigned a value
* **Null** → Intentional empty value
* **Symbol** → Unique and immutable value
* **BigInt** → Large integers beyond Number.MAX\_SAFE\_INTEGER

**2. Non-Primitive (Reference) Data Types**

* **Object** → Collection of key-value pairs
* **Array** → Ordered collection of values
* **Function** → Block of reusable code

Question 3: What is the difference between undefined and null in JavaScript?

| **undefined** | **Null** |
| --- | --- |
| Indicates a variable that has been declared but not yet assigned a value. | Represents the intentional absence of any object value. |
| Primitive data type in JavaScript. | Primitive data type in JavaScript. |
| Automatically assigned to variables that are declared but not initialized. | Must be explicitly assigned to a variable. |

let a;

console.log(a); // undefined

let user = null;

console.log(user); // null

Lab Task

1)Write a JavaScript program to declare variables for different data types (string,number, boolean, null, and undefined).

// String - Text data

let name = "Alice";

console.log("String:", name);

// Number - Numeric data

let age = 25;

console.log("Number:", age);

// Boolean - True or False

let isStudent = true;

console.log("Boolean:", isStudent);

// Null - Intentional absence of value

let job = null;

console.log("Null:", job);

// Undefined - Variable declared but not assigned a value

let address;

console.log("Undefined:", address);

2) Log the values of the variables and their types to the console using console.log().

// String - Text data

let name = "Alice";

console.log("Value:", name, "| Type:", typeof name);

// Number - Numeric data

let age = 25;

console.log("Value:", age, "| Type:", typeof age);

// Boolean - True or False

let isStudent = true;

console.log("Value:", isStudent, "| Type:", typeof isStudent);

// Null - Intentional absence of value

let job = null;

console.log("Value:", job, "| Type:", typeof job); // Note: typeof null returns "object" (it's a known JavaScript quirk)

// Undefined - Variable declared but not assigned a value

let address;

console.log("Value:", address, "| Type:", typeof address);

JavaScript Operators

Question 1: What are the different types of operators in JavaScript? Explain with examples.

### Arithmetic Operators

| Operator | Description | Example | Result |

|-----------|------------------------|---------------------|---------|

| + | Addition | 5 + 3 | 8 |

| - | Subtraction | 10 - 7 | 3 |

| \* | Multiplication | 4 \* 6 | 24 |

| / | Division | 12 / 4 | 3 |

| % | Modulus | 10 % 3 | 1 |

| \*\* | Exponentiation | 2 \*\* 3 | 8 |

| ++ | Increment (Post) | let a = 5; a++ | 6 |

| -- | Decrement (Post) | let b = 10; b-- | 9 |

### Assignment Operators

| Operator | Description | Example | Result |

|-----------|------------------------ |----------------------|---------|

| = | Assignment | let x = 10; | 10 |

| += | Add and Assign | x += 5; | 15 |

| -= | Subtract and Assign | x -= 3; | 12 |

| \*= | Multiply and Assign | x \*= 2; | 24 |

| /= | Divide and Assign | x /= 4; | 6 |

| %= | Modulus and Assign | x %= 5; | 1 |

| \*\*= | Exponentiation and Assign | x \*\*= 2; | 1 |

### Comparison Operators

| Operator | Description | Example | Result |

|-----------|-------------------------------------|----------------|---------|

| == | Equal (Value Only) | 5 == '5' | true |

| === | Strict Equal (Value + Type) | 5 === '5' | false |

| != | Not Equal | 10 != 5 | true |

| !== | Strict Not Equal | 10 !== '10' | true |

| > | Greater Than | 8 > 5 | true |

| < | Less Than | 3 < 7 | true |

| >= | Greater Than or Equal | 5 >= 5 | true |

| <= | Less Than or Equal | 6 <= 4 | false |

### Logical Operators

| Operator | Description | Example | Result |

|-----------|-------------------------------|------------------------|---------|

| && | Logical AND (Both True) | true && false | false |

| || | Logical OR (Either True) | true || false | true |

| ! | Logical NOT (Reverse) | !true | false |

### Bitwise Operators

| Operator | Description | Example | Result |

|-----------|----------------------------|------------------------|---------|

| & | Bitwise AND | 5 & 1 | 1 |

| | | Bitwise OR | 5 | 1 | 5 |

| ^ | Bitwise XOR | 5 ^ 1 | 4 |

| ~ | Bitwise NOT | ~5 | -6 |

| << | Left Shift | 5 << 1 |

Question 2: What is the difference between == and === in JavaScript?

**1. == (Loose Equality)**

* It **compares values only**, without considering the data type.
* If the values are of **different types**, JavaScript **converts them** to the same type before comparing (called **type coercion**).
* It may lead to unexpected results.

**2. === (Strict Equality)**

* It **compares both value and data type** without performing type conversion.
* If the values are of different types, it immediately returns **false**.

Lab Task

1. Create a JavaScript program to perform the following:

\*)Add, subtract, multiply, and divide two numbers using arithmetic operators.

// Perform arithmetic operations

let a = 20;

let b = 5;

console.log(`Addition: ${a} + ${b} = ${a + b}`);

console.log (`Subtraction: ${a} - ${b} = ${a - b}`);

console.log(`Multiplication: ${a} \* ${b} = ${a \* b}`);

console.log(`Division: ${a} / ${b} = ${a / b}`);

\*)Use comparison operators to check if two numbers are equal and if onenumber is greater than the other.

let a = 20;

let b = 5;

console.log(`Is a equal to b? ${a == b}`);

console.log(`Is a greater than b? ${a > b}`);

console.log(`Is a less than b? ${a < b}`);

\*)Use logical operators to check if both conditions (e.g., a > 10 and b < 5)are true.

let a = 20;

let b = 5;

// Check if both conditions are true using logical AND (&&)

console.log(`Is a greater than 10 AND b less than 5? ${a > 10 && b < 5}`);

Control Flow (If-Else, Switch)

Question 1: What is control flow in JavaScript? Explain how if-else statements work withan example.

* Control flow in JavaScript refers to the order in which individual statements, instructions, or functions are executed or evaluated.
* An if-else statement is a conditional statement that executes a block of code if a specified condition is true, and another block of code if the condition is false.

let age = 18;

if (age >= 18) {

console.log("You are eligible to vote.");

} else {

console.log("You are not eligible to vote.");

}

Question 2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

* A switch statement in JavaScript is used to perform different actions based on different conditions.
* It is an alternative to using multiple if-else statements, especially when comparing a single variable to multiple values.
* Syntax

switch (expression) {

case value1:

// Code to execute if expression === value1

break;

case value2:

// Code to execute if expression === value2

break;

default:

// Code to execute if no case matches

}

* When to Use Switch Instead of If-Else
* Multiple Fixed Values: Use switch when you are comparing one value against multiple fixed options, like days of the week or menu options.
* Clean and Readable: When you have a large number of else if conditions, a switch statement makes the code more organized and easier to read.
* Constant Comparison: If you're working with simple equality checks (===) rather than complex conditions.

Lab Assignment

1. Write a JavaScript program to check if a number is positive, negative, or zero usingan if-else statement.

let number = prompt("Enter a number:"); // Taking input from the user

number = Number(number); // Convert input to a number

if (number > 0) {

console.log("The number is positive.");

} else if (number < 0) {

console.log("The number is negative.");

} else {

console.log("The number is zero.");

}

2)Create a JavaScript program using a switch statement to display the day of theweek based on the user input (e.g., 1 for Monday, 2 for Tuesday, etc.).

let day = prompt("Enter a number (1-7) to get the day of the week:");

day = Number(day); // Convert input to a number

switch (day) {

case 1:

console.log("Monday");

break;

case 2:

console.log("Tuesday");

break;

case 3:

console.log("Wednesday");

break;

case 4:

console.log("Thursday");

break;

case 5:

console.log("Friday");

break;

case 6:

console.log("Saturday");

break;

case 7:

console.log("Sunday");

break;

default:

console.log("Invalid input. Please enter a number between 1 and 7.");

}

Loops (For, While, Do-While)

Question 1: Explain the different types of loops in JavaScript (for, while, do-while). Provide abasic example of each.

**1. For Loop**

A **for loop** is used when you know **how many times** you want to execute a statement or block of code

for (let i = 1; i <= 5; i++) {

console.log(i);

}

**2. While Loop**

A **while loop** runs as long as the specified **condition** is true.

let i = 1;

while (i <= 5) {

console.log(i);

i++; // Increment the value of i

}

**3. Do-While Loop**

A **do-while loop** is similar to a while loop, but the code inside the loop will **execute at least once** even if the condition is false.

let i = 1;

do {

console.log(i);

i++;

} while (i <= 5);

Question 2: What is the difference between a while loop and a do-while loop?

| **Aspect** | **While Loop** | **Do-While Loop** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Condition Check** | Condition is checked **before** executing the loop. | Condition is checked **after** executing the loop. |

|  |  |  |
| --- | --- | --- |
| **Execution Guarantee** | May **not execute** even once if the condition is false initially. | Will **execute at least once** even if the condition is false. |

|  |  |  |
| --- | --- | --- |
| **Use Case** | Use when you are unsure if the code should run or not. | Use when you want the code to run at least once. |

Lab Assignment

1)Write a JavaScript program using a for loop to print numbers from 1 to 10.

for (let i = 1; i <= 10; i++) {

console.log(i);

}

2)Create a JavaScript program that uses a while loop to sum all even numbers

let sum = 0;

let number = 1;

const limit = 100;

while (number <= limit) {

if (number % 2 === 0) {

sum += number;

}

number++;

}

console.log(`The sum of all even numbers up to ${limit} is: ${sum}`);

1. Write a do-while loop that continues to ask the user for input until they enter a number greater than 10.

let userInput;

do {

userInput = Number(prompt("Enter a number > 10:"));

} while (userInput <= 10);

console.log("Valid number entered:", userInput);

Function

Question 1: What are functions in JavaScript? Explain the syntax for declaring and calling a function.

* A **function** in JavaScript is a reusable block of code designed to perform a specific task. Functions help in writing modular and maintainable code by allowing code reuse.
* **Syntax for Declaring a Function**
* A function can be declared using the function keyword:

function greet() {

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

}

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

Question 2: What is the difference between a function declaration and a function expression?

| **Feature** | **Function Declaration** | **Function Expression** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Definition** | Declared with the function keyword followed by a name. | Assigned to a variable as an anonymous function or named function. |

|  |  |  |
| --- | --- | --- |
| **Hoisting** | Hoisted to the top of the scope and can be called before definition. | Not hoisted; cannot be called before declaration. |

|  |  |  |
| --- | --- | --- |
| **Syntax Example** | javascript function sayHello() { console.log("Hello!"); } | javascript const sayHello = function() { console.log("Hello!"); }; |

Question 3: Discuss the concept of parameters and return values in functions

* **Parameters in Functions**
* Parameters are variables used to pass values into a function. These values are specified when defining a function inside parentheses.
* **Return Values**
* A function can return a value using the return statement. This allows the function to produce and send back a result.

Lab assignment

* 1. Write a function greetUser that accepts a user’s name as a parameter and displaysa greeting message (e.g., "Hello, John!").

function greetUser(name) {

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

}

// Example usage:

greetUser("John"); // Output: Hello, John!

* 1. Create a JavaScript function calculateSum that takes two numbers as parameters,adds them, and returns the result.

function calculateSum(a, b) {

return a + b;

}

// Example usage:

let result = calculateSum(5, 10);

console.log(result); // Output: 15

Array

Question 1: What is an array in JavaScript? How do you declare and initialize an array?

* An **array** in JavaScript is a special variable that can store multiple values in a single variable. Arrays are useful for organizing and manipulating collections of data efficiently.
* **Declaring and Initializing an Array**
* **let fruits = ["Apple", "Banana", "Mango"]; // String array**
* **let numbers = [1, 2, 3, 4, 5]; // Number array**

Question 2: Explain the methods push(), pop(), shift(), and unshift() used in array

| **Method** | **Description** |
| --- | --- |

|  |  |
| --- | --- |
| **push()** | Adds an element at the end of the array |

|  |  |
| --- | --- |
| **pop()** | Removes the last element from the array |

|  |  |
| --- | --- |
| **shift()** | Removes the first element from the array |

|  |
| --- |
| **unshift()** |

|  |
| --- |
| Adds an element at the beginning of the array |

Lab assignment

* 1. Declare an array of fruits (["apple", "banana", "cherry"]). Use JavaScript to: Add a fruit to the end of the array. Remove the first fruit from the array. Log the modified array to the console

// Declare an array of fruits

let fruits = ["apple", "banana", "cherry"];

// Add a fruit to the end of the array using push()

fruits.push("orange");

// Remove the first fruit from the array using shift()

fruits.shift();

// Log the modified array to the console

console.log(fruits); // Output: ["banana", "cherry", "orange"]

2) Write a program to find the sum of all elements in an array of numbers.

function findSum(arr) {

let sum = 0;

for (let i = 0; i < arr.length; i++) {

sum += arr[i]; // Add each element to sum

}

return sum;

}

// Example usage:

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

console.log(findSum(numbers)); // Output: 15

Objects

Question 1: What is an object in JavaScript? How are objects different from arrays?

* An **object** in JavaScript is a collection of **key-value pairs** where keys (also called properties) are strings, and values can be any data type (string, number, array, function, or even another object). Objects are used to store and organize data in a structured way.
* **How are Objects Different from Arrays?**

| **Feature** | **Object** | **Array** |
| --- | --- | --- |
| **Definition** | A collection of key-value pairs | An ordered list of values |
| **Access Method** | Accessed using keys (obj.key or obj["key"]) | Accessed using indexes (arr[index]) |
| **Best For** | Storing structured data with named properties | Storing lists or collections of similar items |
| **Example** | {name: "Alice", age: 30} | ["apple", "banana", "cherry"] |

Question 2: Explain how to access and update object properties using dot notation and bracket notation.

JavaScript provides two ways to access and update object properties:

1. **Dot Notation (object.property)**

let person = {

name: "Alice",

age: 25

};

console.log(person.name); // Output: Alice

console.log(person.age); // Output: 25

1. **Bracket Notation (object["property"])**

let person = {

"first name": "Alice",

age: 25

};

console.log(person["first name"]); // Output: Alice

console.log(person["age"]); // Output: 25

Lab assignment

1)Create a JavaScript object car with properties brand, model, and year. UseJavaScript to: Access and print the car’s brand and model. Update the year property. Add a new property color to the car object.

// Create an object 'car' with properties brand, model, and year

let car = {

brand: "Toyota",

model: "Corolla",

year: 2022

};

// Access and print the car’s brand and model

console.log("Brand:", car.brand); // Output: Brand: Toyota

console.log("Model:", car.model); // Output: Model: Corolla

// Update the year property

car.year = 2025;

console.log("Updated Year:", car.year); // Output: Updated Year: 2025

// Add a new property 'color' to the car object

car.color = "Red";

console.log("Car Color:", car.color); // Output: Car Color: Red

// Final object after modifications

console.log(car);

/\* Output:

{

brand: 'Toyota',

model: 'Corolla',

year: 2025,

color: 'Red'

}

JavaScript Events

Question 1: What are JavaScript events? Explain the role of event listeners.

* JavaScript **events** are actions or occurrences that happen in the browser, usually as a result of user interaction. Examples of events include:
* **Role of Event Listeners**
* An **event listener** is a function that waits for a specific event to occur and then executes a callback function when the event is detected.
* It **listens** for events on elements like buttons, links, and inputs.
* It **triggers a function** when the event happens.
* It allows **separating HTML and JavaScript** for better code organization.

Question 2: How does the addEventListener() method work in JavaScript? Provide an example.

* element.addEventListener(event, function, useCapture);
* element → The target HTML element.
* event → The event type (e.g., "click", "mouseover", "keydown").
* function → The callback function that runs when the event occurs.
* useCapture (optional) → Boolean value (true or false) that controls event propagation (default is false).

// Select the button element

let button = document.getElementById("myButton");

// Add a click event listener

button.addEventListener("click", function() {

alert("Button Clicked!");

});

Lab task

1)Create a simple webpage with a button that, when clicked, displays an alert saying"Button clicked!" using JavaScript event listeners.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Button Click Event</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 50px;

}

button {

padding: 10px 20px;

font-size: 16px;

background-color: blue;

color: white;

border: none;

cursor: pointer;

border-radius: 5px;

}

button:hover {

background-color: darkblue;

}

</style>

</head>

<body>

<h2>JavaScript Event Listener Example</h2>

<button id="myButton">Click Me</button>

<script>

// Select the button element

let button = document.getElementById("myButton");

// Add an event listener to the button

button.addEventListener("click", function() {

alert("Button clicked!");

});

</script>

</body>

</html>

DOM Manipulation

Question 1: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

* The **Document Object Model (DOM)** is a programming interface that represents the structure of an HTML document as a **tree-like structure** where each HTML element is a **node**.
* **How Does JavaScript Interact with the DOM?**
* JavaScript allows us to:

**Select elements** (e.g., document.getElementById()).  
**Modify content** (e.g., element.innerText = "New Text").  
**Change styles** (e.g., element.style.color = "red").  
**Handle events** (e.g., addEventListener("click", function() {...})).  
**Add or remove elements** (e.g., element.appendChild(newElement)).

Question 2: Explain the methods getElementById(), getElementsByClassName(),and querySelector() used to select elements from the DOM.

* **getElementById()**
* 📌 **Used to select a single element by its id attribute.**
* **getElementsByClassName()**
* 📌 **Used to select multiple elements that share the same class name.**
* **querySelector()**
* 📌 **Used to select the first matching element using a CSS selector.**

Lab assignment

1) Create an HTML page with a paragraph (<p>) that displays "Hello, World!". Use JavaScript to: Change the text inside the paragraph to "JavaScript is fun!". Change the color of the paragraph to blue.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Modify Paragraph with JavaScript</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 50px;

}

p {

font-size: 20px;

color: black;

}

</style>

</head>

<body>

<p id="myParagraph">Hello, World!</p>

<script>

// Select the paragraph element

let paragraph = document.getElementById("myParagraph");

// Change the text inside the paragraph

paragraph.innerText = "JavaScript is fun!";

// Change the color of the paragraph to blue

paragraph.style.color = "blue";

</script>

</body>

</html>

JavaScript Timing Events (setTimeout, setInterval)

Question 1: Explain the setTimeout() and setInterval() functions in JavaScript. Howare they used for timing events?

* JavaScript provides **setTimeout()** and **setInterval()** to execute code after a delay or repeatedly at a set interval.
* **setTimeout() – Delays Execution Once**
* 📌 **Used to execute a function after a specified time (in milliseconds).**  
  📌 **Runs only once** unless called again.
* setTimeout(function, delay);
* **setInterval() – Repeats Execution at Intervals**
* 📌 **Used to execute a function repeatedly at a fixed time interval.**  
  📌 **Keeps running until manually stopped using clearInterval()**.
* setInterval(function, interval)

Question 2: Provide an example of how to use setTimeout() to delay an action by 2 seconds.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>setTimeout Example</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 50px;

}

#message {

font-size: 20px;

color: blue;

font-weight: bold;

}

</style>

</head>

<body>

<button onclick="delayedAction()">Click Me</button>

<p id="message">Wait for 2 seconds...</p>

<script>

function delayedAction() {

setTimeout(() => {

document.getElementById("message").innerText = "Action completed!";

}, 2000); // 2000ms = 2 seconds

}

</script>

</body>

</html>

Lab assignment

1)Write a program that changes the background color of a webpage after 5 secondsusing setTimeout().

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Change Background Color</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

transition: background-color 1s ease-in-out;

}

h1 {

margin-top: 20%;

}

</style>

</head>

<body>

<h1>Wait for 5 seconds...</h1>

<script>

setTimeout(() => {

document.body.style.backgroundColor = "lightblue";

}, 5000); // Change background color after 5 seconds

</script>

</body>

</html>

2)Create a digital clock that updates every second using setInterval().

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Digital Clock</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

background-color: black;

color: white;

margin-top: 15%;

}

#clock {

font-size: 50px;

font-weight: bold;

}

</style>

</head>

<body>

<h1>Digital Clock</h1>

<div id="clock">00:00:00</div>

<script>

function updateClock() {

let now = new Date();

let hours = now.getHours().toString().padStart(2, '0');

let minutes = now.getMinutes().toString().padStart(2, '0');

let seconds = now.getSeconds().toString().padStart(2, '0');

let timeString = `${hours}:${minutes}:${seconds}`;

document.getElementById("clock").innerText = timeString;

}

// Update the clock every second

setInterval(updateClock, 1000);

// Call once immediately to avoid 1-second delay

updateClock();

</script>

</body>

</html>

JavaScript Error Handling

Question 1: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.

**try Block**

* The **try block** contains the code that might throw an error.
* If an error occurs inside try, JavaScript stops executing the try block and jumps to catch.

**catch Block**

* The **catch block** handles the error if one occurs in try.
* It receives the error as a parameter (error or err).

**finally Block (Optional)**

* The **finally block** contains code that **always runs**, whether there was an error or not.
* It is often used for cleanup tasks, like closing files or releasing resources.

function divideNumbers(a, b) {

try {

if (b === 0) {

throw new Error("Cannot divide by zero!"); // Custom error

}

console.log("Result:", a / b);

} catch (error) {

console.log("Error:", error.message);

} finally {

console.log("Division operation completed.");

}

}

// Test cases

divideNumbers(10, 2); // ✅ No error

divideNumbers(5, 0); // ❌ Error: Cannot divide by zero!

Question 2: Why is error handling important in JavaScript applications?

1️ **Prevents Application Crashes** – Without handling, errors stop execution and crash the app.  
2️ **Improves User Experience** – Shows friendly error messages instead of system errors.  
3️ **Helps Debugging & Logging** – Developers can track and fix issues efficiently.  
4️ **Handles Asynchronous Errors** – Prevents API or database failures from breaking the app.  
5️ **Prevents Security Vulnerabilities** – Avoids exposing system details to hackers.

Lab assignment

* 1. Write a JavaScript program that attempts to divide a number by zero. Use try- catch to handle the error and display an appropriate error message.

function divideNumbers(a, b) {

try {

if (b === 0) {

throw new Error("Error: Cannot divide by zero!");

}

console.log(`Result: ${a / b}`);

} catch (error) {

console.log(error.message);

}

}

// Test cases

divideNumbers(10, 2); // ✅ Output: Result: 5

divideNumbers(5, 0); // ❌ Output: Error: Cannot divide by zero!