**Practical 2**

Part 2:

Aim: Variables, Data Types & ES6+ Features: Implement variable declarations using var, let, const. Demonstrate ES6 features such as template literals, destructuring, and default parameters.

#### **Theory Overview:**

## **JavaScript Data Types**

JavaScript data types are broadly categorized into:

### 

### **1. Primitive Data Types**

These are the most basic data types. They hold **single, immutable values**.

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **string** | Text data enclosed in quotes. | "Hello", 'World' |
| **number** | Integers and floating-point numbers. | 42, 3.14 |
| **boolean** | Logical value: true or false. | true, false |
| **undefined** | A variable declared but not assigned a value. | let a; // a is undefined |
| **null** | An intentional empty value. | let x = null; |
| **symbol** | Unique and immutable value often used as object keys. | Symbol('id') |
| **bigint** | Used to represent very large integers. | 12345678901234567890n |

Note: typeof null returns 'object' — this is a known JavaScript quirk.

### 

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

These can hold **collections of values** or **more complex entities**.

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **object** | Key-value pairs used to store structured data. | {name: "Firdous", age: 25} |
| **array** | Ordered collection of values (index-based). | [1, 2, 3], ["a", "b"] |
| **function** | A block of code designed to perform a task. | function greet() {} |

These data types are **mutable** and stored by **reference**.

## 

## **ES6+ JavaScript Features**

Modern JavaScript (ES6 and beyond) introduces powerful syntax improvements:

### 

### **1. Template Literals**

Use backticks ( ` ` ) instead of quotes and embed variables using ${}.

const name = "Firdous";

console.log(`Hello, ${name}! Welcome.`); // Output: Hello, Firdous! Welcome.

**Benefits**:

* Multiline support
* Easier to concatenate variables and strings

### **2. Destructuring**

Extract values from arrays or objects in a single line.

#### 

#### **a) Array Destructuring**

const colors = ["Red", "Green", "Blue"];

const [first, second] = colors;

console.log(first); // Red

#### **b) Object Destructuring**

const student = { id: 1, name: "Riya", branch: "CSE" };

const { name, branch } = student;

console.log(name); // Riya

**Benefits**:

* Cleaner code
* Avoids repetitive access like obj.name, obj.age

### **3. Default Parameters**

Set a default value for function parameters if not passed.

function greet(user = "Guest") {

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

}

greet("Ramesh"); // Hello, Ramesh!

greet(); // Hello, Guest!

**Benefits**:

* Prevents undefined values
* Simplifies function definitions

### 

### **What is ES6?**

**ES6**, also known as **ECMAScript 6** or **ECMAScript 2015**, is the **6th edition** of the ECMAScript language specification. It was officially released in **June 2015** and represents a **major update** to JavaScript with many **modern, cleaner, and more powerful features**.

ECMS **(European Computer Manufacturers Association)**

### **Why is ES6 Important?**

Before ES6, JavaScript lacked many features found in other modern programming languages. ES6 introduced:

* Cleaner syntax
* Better variable handling
* Enhanced functions
* Support for modular and scalable code

It helps developers **write more concise, readable, and efficient code**.

**Key Features of ES6 (With Examples)**

| **Feature** | **Description** | **Example** |
| --- | --- | --- |
| let and const | Block-scoped variable declarations | let x = 5; const y = 10; |
| Template Literals | Use backticks and ${} for string interpolation | `Hello, ${name}` |
| Arrow Functions | Shorter syntax for functions | const add = (a, b) => a + b; |
| Default Parameters | Function parameters with default values | function greet(name = "Guest") {} |
| Destructuring | Extract values from arrays/objects easily | const [a, b] = arr; |
| Spread & Rest | Expand or collect items in arrays/functions | ...args |
| Classes | Introduce OOP-style class syntax | class Person { constructor() {}} |
| Modules | Import/export between files | export default, import |
| Promises | Handle asynchronous operations | fetch().then().catch() |

### **ES6 vs Previous Versions**

| **Feature** | **Pre-ES6** | **ES6** |
| --- | --- | --- |
| Variable declaration | var only | let, const |
| Function syntax | Traditional | Arrow (=>) |
| Modules | None (global script) | import, export |
| String formatting | + operator | Template literals `Hello ${x}` |

##### **Variable Declarations:**

| **Keyword** | **Scope** | **Reassignment** | **Hoisting** | **Usage** |
| --- | --- | --- | --- | --- |
| var | Function | Yes | Yes | Legacy |
| let | Block | Yes | No | Modern |
| const | Block | No | No | Constants |

##### 

### **Spread Operator Example (Expands Arrays or Objects)**

const fruits1 = ["Apple", "Banana"];

const fruits2 = ["Mango", "Orange"];

const allFruits = [...fruits1, ...fruits2];

console.log(allFruits);

// Output: ["Apple", "Banana", "Mango", "Orange"]

**Explanation:**

* The ... operator spreads the elements of fruits1 and fruits2 into the new array allFruits.

### **Rest Operator Example (Collects Values into an Array)**

function showColors(first, second, ...others) {

console.log("First color:", first);

console.log("Second color:", second);

console.log("Other colors:", others); // Rest operator

}

showColors("Red", "Green", "Blue", "Yellow", "Purple");

**Output:**

First color: Red

Second color: Green

Other colors: ["Blue", "Yellow", "Purple"]

**Explanation:**

* The ...others parameter collects all remaining arguments after the first two into an array.

**Hoisting Example Function:**

console.log(myVar); // Output: undefined (hoisted, but not initialized)

foo(); // Output: "Hello, world!" (function declaration is hoisted)

console.log(myLet); // Output: ReferenceError (myLet is not accessible before declaration)

var myVar = "This is var";

let myLet = "This is let";

function foo() {

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

}

##### **Example 1: var vs let vs const**

<script>

var x = 5;

let y = 10;

const z = 15;

console.log("var x:", x); // 5

console.log("let y:", y); // 10

console.log("const z:", z); // 15

x = 20;

y = 25;

// z = 30; // Error

console.log("Updated x:", x); // 20

console.log("Updated y:", y); // 25

</script>

##### **Example 2: Data Types Demo**

<script>

let str = "Hello";

let num = 42;

let isReady = true;

let empty = null;

let notDefined;

console.log(typeof str); // string

console.log(typeof num); // number

console.log(typeof isReady); // boolean

console.log(typeof empty); // object (special case)

console.log(typeof notDefined); // undefined

</script>

##### **Example 3: Template Literals**

<script>

const name = "Firdous";

const course = "JavaScript Lab";

console.log(`Welcome ${name} to your ${course}.`);

</script>

##### **Example 4: Destructuring Arrays and Objects**

<script>

const colors = ["Red", "Green", "Blue"];

const [first, second] = colors;

console.log(first); // Red

console.log(second); // Green

const student = { id: 1, name: "Riya", branch: "CSE" };

const { name, branch } = student;

console.log(name); // Riya

console.log(branch); // CSE

</script>

##### **Example 5: Default Parameters**

<script>

function greet(user = "Guest") {

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

}

greet("Ramesh"); // Hello, Ramesh!

greet(); // Hello, Guest!

</script>

##### **Example 6: Hoisting Concepts**

console.log(x); // Output: undefined (because of var hoisting)

var x = 10;

console.log(myFunction()); // Output: "Hello" (because of function declaration hoisting)

function myFunction()

{

return "Hello";

}

// console.log(myFuncExpression()); // Output: ReferenceError: Cannot access 'myFuncExpression' before initialization (because of function expression hoisting)

const myFuncExpression = function()

{

return "World";

}

**Task:**

| **Task No.** | **Task Description** | **Hints** |
| --- | --- | --- |
| 1 | Declare variables using var, let, const and reassign values. | Note which one throws an error. |
| 2 | Create a function that greets a user using **template literals**. | Use ${variable} inside backticks. |
| 3 | Destructure a student object and print details. | Use {} or []. |
| 4 | Write a function that calculates sum with default parameters. | function sum(a = 5, b = 10) |
| 5 | Identify data types using typeof. | Use at least 6 types. |