JavaScript Variables

1. Variables

Definition:

A variable in JavaScript is a named container used to store data or values that your program can use and manipulate. Think of a variable like a labeled box: you can put something inside, retrieve it later, and sometimes even replace it with something else.

Purpose of Variables:

1. Store data dynamically instead of hardcoding values.
2. Improve code reusability, avoiding repeated literal values.
3. Enhance code readability by giving meaningful names to data.
4. Enable dynamic operations, like calculations or text manipulation.

Identifiers: Identifiers are the names given to variables.

Rules to write variable names:

1. Must be case-sensitive.
2. Cannot start with a number.
3. Can start with letters (a-z, A-Z), underscore (_), or dollar sign ($).
4. Numbers can be used in the middle or end.

Examples:

```
let  name;      // valid
let  name;     // valid
let $name;     // valid
let nam12eng;  // valid
let 12name;    // invalid
```

2. Keywords to Declare Variables

2.1 var - Can be redeclared and reassigned.

Scope: Function-scoped.

Syntax:

```
var variableName = value;
```

Example:

```
var age = 20;
var age = 25;
age = 30;
console.log(age); // Output: 30
```

2.2 let - Cannot be redeclared in the same scope. Can be reassigned. Scope: Block-scoped.

Syntax:

```
let variableName = value;
```

Example:

```
let city = "Hyderabad";
city = "Warangal";
console.log(city); // Output: Warangal
```

2.3 const - Cannot be redeclared or reassigned. Must be initialized at declaration.

Scope: Block-scoped.

Syntax:

```
const variableName = value;
```

Example:

```
const pi = 3.14;
console.log(pi); // Output: 3.14
```

3. Comments in JavaScript

Single-Line Comment: Use // Example:

```
// This variable stores the user's age
let age = 25;
```

Block-Level Comment: Use /* … */ Example:

```
/*
This program calculates
the area of a rectangle
*/
let length = 10;
let breadth = 5;
let area = length * breadth;
```

4. Data Types in JavaScript

Definition:
Data types in JavaScript determine the type of data a variable can store. This is essential for the JavaScript engine to **interpret and manipulate the data correctly**. JavaScript has two major categories of data types: **Primitive** and **Non-Primitive**.

4.1 Primitive Data Types

Definition:
Primitive data types are **immutable**, meaning their value cannot be changed once assigned. If you try to modify a primitive value, a new value is created in memory.

Types and Examples:

1. **Number:** Stores numeric values, both integers and floating-point numbers.

```
let age = 25;
let price = 499.99;
```

2. **String:** Stores a sequence of characters enclosed in single or double quotes.

```
let name = "Megha";
let greeting = 'Hello World';
```

3. **Boolean:** Stores logical values: true or false.

```
let isLoggedIn = true;
let hasAccess = false;
```

4. **Undefined:** When a variable is declared but not assigned any value.

```
let address;
console.log(address); // Output: undefined
```

5. **Null:** Represents the intentional absence of any value.

```
let result = null;
```

6. **BigInt:** Stores integers larger than 2^53 - 1. Can be created using an `n` at the end of a number or using `BigInt()`.

```
let bigNumber1 = 123456789012345678901234567890n;
let bigNumber2 = BigInt("123456789012345678901234567890");
```

Additional Explanation:

- Primitive types are **stored by value**, meaning each variable holds its own copy of the data.
- They are **fast and memory-efficient**.
- Immutability ensures **data integrity**, preventing unintended changes.

4.2 Non-Primitive Data Types

Definition:
Non-primitive data types are **mutable**, meaning their contents can be changed after creation. They are **reference types**, so multiple variables can refer to the same object in memory.

Types and Examples:

1. **Arrays:** Ordered collection of elements that can be of any data type.

```
let colors = ["Red", "Green", "Blue"];
```

2. **Objects:** Collections of key-value pairs for storing complex structured data.

```
let person = {name: "Megha", age: 21};
```

3. **Functions:** Reusable blocks of code that can be called with arguments.

```
function greet(name) {
  return `Hello, ${name}`;
}
```

4. **Date:** Represents date and time.

```
let today = new Date();
```

5. **RegExp:** Regular expressions for pattern matching in strings.

```
let pattern = /abc/;
```

Additional Explanation:

- Non-primitives are **stored by reference**, meaning changes in one variable can affect others referencing the same object.
- They allow for **complex data structures** and dynamic operations like adding/removing elements from arrays or updating object properties.
- Useful in real-world applications like **user profiles (objects), to-do lists (arrays), and form validations (RegExp)**.

Real-World Analogy:

- **Primitive:** Like individual ingredients in separate jars—each is independent.
- **Non-Primitive:** Like a recipe book where multiple cooks can modify the same recipe; changes affect everyone referencing the book.

5. typeof() Operator

Checks the data type of a variable.

Syntax: typeof variableName;

Example:

```
let age = 25;
console.log(typeof age); // number
```

```
let name = "Megha";
console.log(typeof name); // string
```

JavaScript Arrays

Definition:

JavaScript arrays are **special variables that allow us to store multiple values in a single variable**. Unlike Java, which requires all elements to be of the same datatype, JavaScript arrays can hold **heterogeneous elements** (numbers, strings, booleans, objects, etc.).

Key Points:

1. Arrays can store multiple data in one variable.
2. Arrays can hold **any type of data** (heterogeneous).
3. Array **index starts from 0**.

Example Using Individual Variables vs Array:

```
// Using variables
var a = 'Megha';
var b = 30;
var c = 'Female';

// Using array
var arr = ['Megha', 30, 'Female'];
```

Declaration of Arrays in JavaScript

JavaScript arrays can be declared in **two ways:**

1. Using **Array Literals**

2. Using **Array Object / new keyword**

1. Using Array Literals

Definition: Array literals are declared using **square brackets []**, which can either be empty or initialized with elements.

Syntax:

```
// Empty array
let arrayName = [];

// Array with elements
let arrayName = [element1, element2, element3];
```

Example 1: Initializing array while declaring

```
var arr = ['Megha', 30, 'Female'];
console.log(arr);
// Output: ['Megha', 30, 'Female']
```

Example 2: Declaring empty array and adding values using index

```
var arr = [];
arr[0] = 'Megha';
arr[1] = 30;
arr[2] = 'Female';
console.log(arr);
// Output: ['Megha', 30, 'Female']
```

Accessing Individual Elements Using Index:

```
console.log(arr[0]); // Output: 'Megha'
console.log(arr[2]); // Output: 'Female'
```

2. Using Array Object / new Keyword

Definition: JavaScript provides a built-in **Array() method** to declare arrays using the `new` keyword. Arrays can be initialized immediately or values can be added later.

Syntax:

```
// Empty array
var arrayName = new Array();

// Array with elements
var arrayName = new Array(element1, element2, element3);
```

Example 1: Declaring empty array and adding values

```
var arr = new Array();
arr[0] = 12323;
arr[1] = 'Megha';
arr[2] = true;
document.write(arr); // Output: [12323, 'Megha', true]
```

Example 2: Accessing individual elements

```
document.write(arr[0]); // Output: 12323
document.write(arr[1]); // Output: 'Megha'
```

Example 3: Declaring and initializing array at the same time

```
var arr = new Array(12323, 'Megha', true);
```

```
console.log(arr); // Output: [12323, 'Megha', true]
```