Introduction to JavaScript Programming

Module 5 - Variables and Data Types in Java Script

Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

Definition

It is a quantity whose value can be change during the execution of the program.

Declaration of variable

Variables are declared with the var keyword as follows.

```
var money;
var name;
```

You can also declare multiple variables with the same var keyword as follows:

```
var money, name;
```

NOTE: In JavaScript, variables are declared using the **keywords var, let, or const.**

The **var keyword** is used to declare a variable. It has a function-scoped or globally-scoped behaviour.

```
var n = 10;
var n = 12; // Re-declaration is allowed
```

The **let keyword** is introduced in ES6, has block scope and cannot be re-declared in the same scope.

```
let  n= 10;
n = 20; // Value can be updated
// let n = 15; //can not redeclare
```

The **const keyword** declares variables that cannot be reassigned. It's block-scoped as well.

```
const n = 100;
// n = 200; This will throw an error
```

Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character

Introduction to 'JavaScript' Language

Variable Initialization

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create variable named money and assign the value 523.50 to it later.

For another variable, you can assign a value at the time of initialization as follows.

```
var name = "Shan";
var money;
money = 523.50;
```

NOTE: Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped language.** This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold.

The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- Global Variables: A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- Local Variables: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
var myVar = "global"; // Declare a global variable
function checkscope( )
{
var myVar = "local"; // Declare a local variable
```

```
document.write(myVar);
}
</script>
```

Why we use + in document.write()?

In JavaScript, the **+ operator** is used to **join (concatenate)** strings and variables together.

For example:

```
var name = "Amit";
document.write("Name: " + name);
```

Here's what happens:

- "Name: " → is a string (text)
- name → is a variable that holds the value "Amit"
- "Name: " + name → joins them together → "Name: Amit"

**Why we use
?**

 is an **HTML tag** that means "line break."

When you use document.write(), it writes **HTML directly into the page** – not plain text.

So, if you want each output to appear on a new line, you need to insert an HTML line break.

Solved Example: Variables in Java Script using var

```
<!DOCTYPE html>
<html>
<body>

<script>
// Declaring variables using var
var name = "Amit"; // Initial name
var age = 28;      // Initial age
var city = "Mumbai"; // Initial city

// Printing the values
document.write("Name: " + name + "<br>");  // Output: Name: Amit
document.write("Age: " + age + "<br>");    // Output: Age: 28
document.write("City: " + city + "<br><br>"); // Output: City: Mumbai

// Changing variable values (reassignment is allowed with var)
name = "Rahul"; // Updated name
age = 32;       // Updated age
city = "Delhi"; // Updated city
```

```
// Printing updated values
document.write("Updated Name: " + name + "<br>"); // Output: Updated Name: Rahul
document.write("Updated Age: " + age + "<br>");    // Output: Updated Age: 32
document.write("Updated City: " + city + "<br>"); // Output: Updated City: Delhi
</script>

</body>
</html>
```

OUTPUT:

```
Name: Amit
Age: 28
City: Mumbai

Updated Name: Rahul
Updated Age: 32
Updated City: Delhi
```

Explanation

- Using var, all variables (name, age, and city) can be reassigned freely.
- Variables declared with var are function-scoped or globally scoped, meaning their scope isn't restricted to the block they are declared in.

Solved Example: Variables in Java Script using let

```
<!DOCTYPE html>
<html>
<body>

<script>
// Declaring variables using let
let name = "Amit"; // Initial name
let age = 28;      // Initial age
let city = "Mumbai"; // Initial city

// Printing the values
document.write("Name: " + name + "<br>");  // Output: Name: Amit
document.write("Age: " + age + "<br>");    // Output: Age: 28
document.write("City: " + city + "<br><br>"); // Output: City: Mumbai

// Changing variable values (reassignment is allowed with let)
name = "Rahul"; // Updated name
age = 32;       // Updated age
```

```
city = "Delhi"; // Updated city

// Printing updated values
document.write("Updated Name: " + name + "<br>"); // Output: Updated Name: Rahul
document.write("Updated Age: " + age + "<br>");   // Output: Updated Age: 32
document.write("Updated City: " + city + "<br>"); // Output: Updated City: Delhi
</script>

</body>
</html>
```

OUTPUT

```
Name: Amit
Age: 28
City: Mumbai

Updated Name: Rahul
Updated Age: 32
Updated City: Delhi
```

Data Types in Javascript

In JavaScript, there are several data types that you can use to represent various kinds of data. They are broadly categorized into primitive and non-primitive types. Here's a breakdown of them:

1. Primitive Data Types (Simple types):
These types represent a single value and are immutable (cannot be changed once created).

Number: Represents numeric values (both integers and floating-point numbers).

```
let age = 25;
let price = 19.99;
```

String: Represents a sequence of characters (text).

```
let name = "John";
let greeting = 'Hello, world!';
```

Boolean: Represents a value of either true or false.

```
let isActive = true;
let isComplete = false;
```

Introduction to 'JavaScript' Language

Undefined: Represents a variable that has been declared but not assigned a value.

```
let someVariable;
console.log(someVariable);  // undefined
```

Null: Represents the intentional absence of any value or object.

```
let emptyValue = null;
```

Symbol (ES6): Represents a unique identifier, primarily used for object property keys.

```
let symbol1 = Symbol('description');
```

2. Non-Primitive Data Types (Complex types):

These are more complex and can hold collections of data.

Object: A collection of key-value pairs, where keys are strings (or Symbols), and values can be any data type.

```
let person = {
  name: "Alice",
  age: 30,
  isActive: true
};
```

Array: An ordered list of values, which can be of any data type.

```
let numbers = [1, 2, 3, 4];
let mixed = ["apple", 42, true];
```

In addition, arrays are technically a type of object in JavaScript.

Summary:

- Primitive types: Number, String, Boolean, Undefined, Null, Symbol, BigInt.

Non-Primitive types: Object, Array.