Using JavaScript Variable and data types

Objectives : At the end of this lesson you shall be able to
- **explain variables in JavaScript**
- **explain various data types in JavaScript.**

Variables

JavaScript variables are containers for storing data values.

In example 1, a, b, and c, are variables:

Example 1

```
var a = 12;
var b = 10;
var c = a + b;
```

From example 1, we can understand that

- **a** stores the value 12
- **b** stores the value 10
- **c** stores the value 22

In example 2, mark1, mark2, and total, are variables:

Example 2

```
var mark1 = 85;
var mark2 = 66;
var total = marks1 + mark2;
```

In programming, just like in algebra, we use variables **mark1** and **mark2** to hold values and use variables in expressions like total = mark1 + mark2. From the example above, and calculate the total to be 151.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**. These unique names are called **identifiers**. Identifiers can be short names like a and b or more descriptive names like mark1, mark2, total, age, sum, total volume.

The general rules for constructing names for variables are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (a and A are different variables)
- Reserved words like JavaScript keywords cannot be used as names

Note: JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (=) is an “assignment” operator, not an “equal to” operator.

```
x = x + 10;
```

It assigns the value of x + 10 to x. It calculates the value of x + 10 and puts the result into x. The value of x is incremented by 10.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like “Santhosh kumar”.

In programming, text values are called text strings. Java Script can handle many types of data, but for now, just think of numbers and strings.**Strings** are written inside double or single quotes. **Numbers** are written without quotes. If you put a number in quotes, it will be treated as a text string.

Example 3

```
var pi = 3.14;
var person = “santhoshkumar”;
var city = “coimbatore”;
```

Declaring JavaScript Variables

Creating a variable in JavaScript is called **declaring** a variable. JavaScript variable is declared with the **var** keyword.

```
var traineeName;
```

After the declaration, the variable has no value. Technically it has the value of **undefined**. To **assign** a value to the variable, use the equal signs.

```
traineeName = “Santhosh Kumar”;
```

You can also assign a value to the variable when you declare it.

```
var traineeName = “Santhosh Kumar”;
```

In the example below, we create a variable called traineeName and assign the value “Santhosh Kumar” to it.

Then we “output” the value inside an HTML paragraph with id=”demo”:

```
<p id =“demo”></p>
<script>
var traineeName = “santhoshkumar”;
document.getElementById(“demo”).innerHTML = traineeName;
</script>
```

Note: It is a good programming practice to declare all variables at the beginning of a script.

You can declare many variables in one statement. Start the statement with **var** and separate the variables by **comma**.

Example 4

```
var traineeName = “santhoshkumar”,city = ‘’coimbatore”, total=”151";
```

Undefined value

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value **undefined**.

The variable **traineeName** will have the value undefined after the execution of this statement.

Example 5

```
var traineeName;
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.The variable traineeName will still have the value “santhoshkumar” after the execution of these statements.

Example 6

```
var traineeName = “santhoshkumar”;
var traineeName;
```

JavaScript Arithmetic

Do the arithmetic with JavaScript variables, using operators like **=** and **+**

Example 7

```
var x = 8 + 2 + 5;
```

Now x has the value **15.**

You can also add strings, but strings will be concatenated:

Example 8

```
var x = “Dharani” + “ “ + “Shree”
```

Now x has the value **Dharani Shree**

The result of the following example gives **725.**

Example 9

```
var x = “7” + 2 + 5;
```

Note: If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

The result of the following example gives **75.**

Example 10

```
var x = 3 + 4 + “5”;
```

Data types

In programming, data types is an important concept. To be able to operate on variables, it is important to know about the data type.

JavaScript variables can hold many **data types** like numbers, strings, objects and more.

Example 11

```
var side = 10;                                  // Number
var firstName = “Rithika”;                      // String
var x = {firstName:”Harini”, lastName:”Kumar”}; // Object
```

Without data types, a computer cannot safely solve this.

Example 12

```
var a = 10 + “Apple”;
```

JavaScript will treat the example above as,

```
var a = “10” + “Apple”;
```

The output is **10 Apple**

Note: When adding a number and a string, JavaScript will treat the number as a string.

JavaScript evaluates expressions from left to right. Different sequences can produce different results.

Example 13

```
var y = 20 + 5 + “Apple”;
```

The result is **25Apple**

Example 14

```
var y = “Apple”+20 + 5 ;
```

The result is **Apple205.**

Note: In the first example, JavaScript treats 20 and 5 as numbers, until it reaches “Apple”.In the second example, since the first operand is a string, all operands are treated as strings.

Dynamic data types

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example 15

```
var z;              // Now z is undefined
z = 10;             // Now z is a Number
z = “Sakthi”;       // Now z is a String
```

JavaScript Strings

A string or a text string is a series of characters like “Harini Kumar”. Strings are written with quotes. You can use single or double quotes.

Example 16

```
var bikeName = “Yamaha R15”;   // Using double quotes
var bikeName = ‘ Yamaha R15’;  // Using single quotes
```

You can use quotes inside a string, as long as they don’t match the quotes surrounding the string:

Example 17

```
var answer = “It’s OK”;              // Single quote inside
                                        double quotes
var answer = ‘Patel is called        // Double quotes inside
“Iron Man”’;                            single quotes
```

JavaScript Numbers

JavaScript has only one type of numbers. Numbers can be written with or without decimals.

Example 18

```
var num1 = 87.0;     // Written with decimals
var num2 = 87;       // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example 19

```
var exp1 =  232e5;   // result is 23200000
var z    =  123e-5;  // result is 0.00232
```

Example 20

```
var p = 3:
var q = 3;
var r = 5;
(p == q)             // Returns true
(p == r)             // Returns false
```

Note : Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets. Array items are separated by commas. The following code declares (creates) an array called bikes, containing three items (bike names):

Example 21

```
var bikes = [“Yamaha”, “TVS”, “Royal Enfield”];
```

Note: Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

JavaScript Objects

JavaScript objects are written with curly braces. Object properties are written as name:value pairs, separated by commas.

Example 22

```
var personName =    {firstName:”Harini”,lastName:
                    ”Kumar”, age:13,height.
                    ”155 cms”};
```

The object (personName) in the example 22 above has 4 properties: firstName, lastName, age and height.

The typeof Operator

The JavaScript **typeof** operator is used to find the type of a JavaScript variable.

The **typeof** operator returns the type of a variable or an expression.

Example 23

```
typeof ""                  // Returns "string"
typeof "Rithika"           // Returns "string"
typeof "Harini Kumar"      // Returns "string"
typeof0                    // Returns "number"
typeof81                   // Returns "number"
typeof8.14                 // Returns "number"
typeof(3+2)                // Returns "number"
```

Undefined

In JavaScript, a variable without a value, has the value **undefined**. The typeof is also **undefined**.

Example 24

```
var bike;          // Value is undefined, type is
                      undefined
```

Note : Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

Empty Values

An empty value has nothing to do with undefined. An empty string has both a legal value and a type.

Example 25

```
var bike = "";     // The value is "", the typeof
                      is "string"
```

Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist. In JavaScript, the data type of null is an object. You can empty an object by setting it to null.

Example 26

```
var personName =  {firstName:"Harini",last Name;
                  "Kumar", age:13, height:"155 cms"};
personName      = null;  //Now value in null, but
                  type is still an object
```

You can also empty an object by setting it to undefined:

Example 27

```
var personName = {firstName:"Harini", lastName:
                  "Kumar", age:13, height:"155 cms"};
personName    = undefined;      // Now both value and
                  type is undefined.
```

Difference Between Undefined and Null

Undefined and null are equal in value but different in type.

Example 28

```
typeof undefined          // undefined
typeof null               // object
null === undefined        // false
null == undefined         // true
```

Primitive Data

A primitive data value is a single simple data value with no additional properties and methods. The **typeof** operator can return one of these primitive types.

- string
- number
- boolean
- undefined

Example 29

```
typeof "Rajesh"      // Returns "string"
typeof 1.44          // Returns "number"
typeof true          // Returns "boolean"
typeof false         // Returns "boolean"
typeof a             //  if a has no value, it returns
                        "undefined"
```

Complex Data

The **typeof** operator can return one of two complex types:

- function·
- object

The type of operator returns object for both objects, arrays and null. It does not return object for functions.

Example 30

```
typeof {name, 'Karthik', age 27}   // Returns "object"
typeof [10, 20, 30, 40, 50]        // Returns "object"
                                      (not "array", see
                                       note below)
typeof null                        // Returns "object"
typeof function sampleFunc() { }   // Returns "function"
```

Note: The typeof operator returns "object" for arrays because in JavaScript arrays are objects.