JavaScript

Unit 3

Javascript

JavaScript is the programming language of HTML and the Web. Programming makes computers do what you want them to do. JavaScript is easy to learn.

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complementary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

What is javascript?

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name**LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

Advantages

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of javascript

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

What javascript can do?

JavaScript Can Change HTML Content

JavaScript Can Change HTML Attributes

JavaScript Can Change HTML Styles (CSS)

JavaScript Can Validate Data

How to write code?

```
<html>
   <body>
      <script language="javascript" type="text/javascript">
         <!--
            document.write("Hello World!")
         //-->
      </script>

   </body>
</html>
```

Syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
   JavaScript code
</script>
```

The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like –

```
<script language="javascript" type="text/javascript">
   JavaScript code
</script>
```

white spaces & line breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
   <!--
      var1 = 10
      var2 = 20
   //-->
</script>
```

Note – It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE – Care should be taken while writing variable and function names in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus –

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Types

1. inline
2. internal
3. external

JavaScript - Variables

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types –

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

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.

```
<script type="text/javascript">
      var money;
      var name;
</script>
```

OR

```
<script type="text/javascript">
      var money, name;
</script>
```

Variable initialization

```
<script type="text/javascript">
    <!--
        var name = "Ali";
        var money;
        money = 2000.50;
    //-->
</script>
```

Variables

```
<p id="demo"></p>

<script>
    var price1 = 5;
    var price2 = 6;
    var total = price1 + price2;
    document.getElementById("demo").innerHTML =
     "The total is: " + total;
</script>
```

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

Operators in javascripts

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and ‘+’ is called the **operator**. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Assignment Operators

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Logical Operators

Operator	Description
&& (Logical AND)	If both the operands are non-zero, then the condition becomes true.
|| (Logical OR)	If any of the two operands are non-zero, then the condition becomes true.
! (Logical NOT)	Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

conditional operator (? :)

? : (Conditional)

If Condition is true? Then value X : Otherwise value Y

```
<script>
    var x=10;
    (x>15) ? “x is greater than 15” : “x is less than 15”;
</script>
```

JavaScript - if...else Statement

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

if … else ….

- if statement
- if...else statement
- if...else if... statement.

if statement

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

```
if (expression){
   Statement(s) to be executed if expression is true
}
```

if...else statement:

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression){
   Statement(s) to be executed if expression is true
}

else{
   Statement(s) to be executed if expression is false
}
```

if...else if... statement

The **if...else if...** statement is an advanced form of **if...else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows –

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}
else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}
else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}
else{
   Statement(s) to be executed if no expression is true
}
```

Switch case

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

switch case syntax

```
switch (expression)
{
   case condition 1: statement(s)
   break;

   case condition 2: statement(s)
   break;
   ...

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

While Loops

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

```
while (expression){
   Statement(s) to be executed if expression is true
}
```

For Loop

The '**for**' loop is the most compact form of looping. It includes the following three important parts –

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

for loop syntax

```
for (initialization; test condition; iteration statement){
   Statement(s) to be executed if test condition is true
}
```

ex.

```
var count;

for(count = 0; count < 10; count++){
               document.write("Current Count : " + count );
               document.write("<br />");
            }
```

JavaScript - Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

Function Definition

The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

```
<script                                                type="text/javascript">
                                                                        <!--
                                        function      functionname(parameter-list)
                                                                             {
                                                                    statements
                                                                             }
                                                                        //-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<script type="text/javascript">
         function sayHello()
         {
            document.write ("Hello there!");
         }
</script>

<input type="button" onclick="sayHello()" value="Say Hello">
```

extra...

functions with parameters

function with return statement

Event handling

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript - The Math Object

The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

The Math object allows you to perform mathematical tasks.

The Math object includes several mathematical methods.

Example of math objects

Method	Description
abs()	Returns the absolute value of a number.
ceil()	Returns the smallest integer greater than or equal to a number.
floor()	Returns the largest integer less than or equal to a number.
log()	Returns the natural logarithm (base E) of a number.
max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.

Method	Description
pow()	Returns base to the exponent power, that is, base exponent.
random()	Returns a pseudo-random number between 0 and 1.
round()	Returns the value of a number rounded to the nearest integer.
sqrt()	Returns the square root of a number.

Javascript String Object

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

Syntax- String

```
var x = "John";

var y = new String("John");


// typeof x will return string

// typeof y will return object

// (x == y) is true because x and y have equal values
```

String Length

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

var sln = txt.length;
```

Special Characters

```
var y = “this is javascript “string” object”.
```

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **\ escape character**.

The backslash escape character turns special characters into string characters

```
var x = 'It\'s alright';
```

```
var y = "We are the so-called \"Vikings\" from the north."
```

escape character (\)

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace