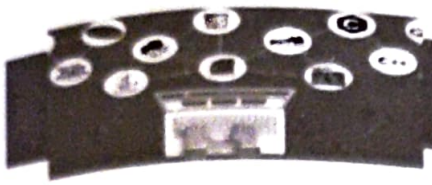

Chapter 4

INTRODUCTION TO JAVASCRIPT

CHAPTER CONTENTS

4.1 Introduction
4.2 Syntax of JavaScript
4.3 JavaScript Placement
4.4 Variables
  4.4.1 Variable Scope
  4.4.2 Data Types
  4.4.3 Identifiers
4.5 Operators
  4.5.1 Arithmetic Operators
  4.5.2 Comparison Operators
  4.5.3 Logical Operators
  4.5.4 Bitwise Operators
  4.5.5 Assignment Operators
  4.5.6 Conditional Operator
  4.5.7 typeof Operator
4.6 Decision Making
  4.6.1 *if* Statement
  4.6.2 *if..else* Statement
  4.6.3 *if..else if* Statement
  4.6.4 *switch* Statement
4.7 Loops
  4.7.1 *while* Loop
  4.7.2 *do..while* Loop
  4.7.3 *for* Loop
  4.7.4 *for..in* Loop
4.8 Loop Control
  4.8.1 *break* Statement
  4.8.2 *continue* Statement
  4.8.3 Using Labels to Control Loop
4.9 Functions
  4.9.1 Function Definition
  4.9.2 Function Calling
  4.9.3 Function Parameters
  4.9.4 *return* Statement
4.10 Conclusion

4.1 INTRODUCTION

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform. JavaScript is a dynamic computer programming language. It is most commonly used as a part of Web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.

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.

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser. It means that a Web page need not be a static HTML, but can include programs that interact with the user, control the browser and dynamically create HTML content. The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, we might use JavaScript to check if the user has entered a valid e-mail address in a form field. The JavaScript code is executed when the user submits the form and only if all the entries are valid, they would be submitted to the Web Server. JavaScript can be used to trap user-initiated events such as button clicks, link navigation and other actions that the user initiates explicitly or implicitly.

The advantages of using JavaScript are as follows.

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

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features.

- 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.

4.2 SYNTAX OF JAVASCRIPT

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script>. We can place the <script> tags, containing our JavaScript, anywhere within your Web page, but it is normally recommended that we 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 our JavaScript will appear as follows.

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

The script tag takes two important attributes.

Language – This attribute specifies what scripting language we are using. Typically, its value will be javascript.

Type – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

Example Program

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

Output

Hello World!

We have called a function `document.write()` which writes a string into our HTML document.

JavaScript ignores spaces, tabs and newlines that appear in JavaScript programs. Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++ and Java. JavaScript, however, allows us to omit this semicolon if each of our statements are placed on a separate line. 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.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments.

- 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 //-->.

4.3 JAVASCRIPT PLACEMENT

JavaScript has the flexibility to include code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows.

- **Script in <head>...</head> section**

If we want to have a script run on some event, such as when a user clicks somewhere, then we will place that script in the head as follows.

Example Program

```
<html>
   <head>
      <script type="text/javascript">
        <!--
           function sayHello() {
              alert("Hello World")
           }
        //-->
      </script>
       </head>
      <body>
      <input type="button" onclick="sayHello()" value="Say Hello" />
   </body>
   </html>
```

Output

On clicking on the above button, it will display "Hello World".

- **Script in <body>...</body> section**

If we need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, we would not have any function defined using JavaScript. We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript.

Example Program

```
<html>
   <head>
   </head>
     <body>
```

```
    <script type="text/javascript">
      <!--
         document.write("Hello World")
      //-->
    </script>
    <p>This is web page body </p>
  </body>
</html>
```

Output

Hello World

This is web page body

- **Script in <body>...</body> and <head>...</head> sections**

  We can put our JavaScript code in <head> and <body> section altogether as follows.

Example Program

```
<html>
  <head>
    <script type="text/javascript">
      <!--
         function sayHello() {
            alert("Hello World")
         }
      //-->
    </script>
  </head>
  <body>
    <script type="text/javascript">
      <!--
         document.write("Hello World")
      //-->
    </script>

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

Output

Hello World [Say Hello]

On clicking on this button Say Hello, it will display "Hello World".

- **Script in an external file and then include in <head>...</head> section**

We will be likely to find that there are cases where we are reusing identical JavaScript code on multiple pages of a site. We are not restricted to maintain identical code in multiple HTML files. The script tag provides a mechanism to allow us to store JavaScript in an external file and then include it into our HTML files. Here is an example to show how we can include an external JavaScript file in our HTML code using script tag and its `src` attribute.

Example Program

```
<html>
   <head>
      <script type="text/javascript" src="filename.js" ></script>
   </head>
   <body>
      .......
   </body>
</html>
```

To use JavaScript from an external file source, we need to write all our JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above. For example, we can keep the following content in `filename.js` file and then we can use `sayHello()` function in our HTML file after including the `filename.js` file.

```
function sayHello() {
   alert("Hello World")
}
```

4.4 VARIABLES

Variables can be thought of as named containers. We can place data into these containers and then refer to the data simply by naming the container. Before we use a variable in a JavaScript program, we must declare it. Variables are declared with the var keyword as follows.

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

We can also declare multiple variables with the same `var` keyword as follows.

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

Storing a value in a variable is called variable initialization. We can do variable initialization at the time of variable creation or at a later point in time when you need that variable. For instance, we might create variable named money and assign the value 2000 to it later. For another variable, we can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
  <!--
     var name = "Adam";
     var money;
     money = 2000;
  //-->
</script>
```

We can use the var keyword only for declaration or initialization, once for the life of any variable name in a document. We 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, we 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.

4.4.1 Variable Scope

The scope of a variable is the region of our 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 our 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 we declare a local variable or function parameter with the same name as a global variable, we effectively hide the global variable. Consider the following example.

```
<html>
  <body onload = checkscope();>
     <script type = "text/javascript">
        <!--
```

```
        var myVar = "global"; // Declare a global variable
        function checkscope( ) {
            var myVar = "local";  // Declare a local variable
            document.write(myVar);
        }
    //-->
    </script>
  </body>
</html>
```

This produces the following result.

```
local
```

4.4.2 Data Types

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 us to work with three primitive data types.

- Numbers, eg. 123, 120.50 etc.
- Strings of text e.g. "This is an apple" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object. We will cover objects later. JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format.

4.4.3 Identifiers

All JavaScript variables must be identified with unique names. These unique names are called identifiers. Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are as follows.

1. Names can contain letters, digits, underscores, and dollar signs.
2. Names must begin with a letter.
3. Names can also begin with $ and _ .
4. Names are case sensitive (y and Y are different variables).
5. Reserved words cannot be used as identifiers.

4.5 OPERATORS

JavaScript supports the following types of operators.

4.5.1 Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers. Following Table shows various arithmetic operators available in JavaScript.

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

Example Program

```
<html>
   <body>
        <script type="text/javascript">
        <!--
           var a = 33;
           var b = 10;
           var linebreak = "<br />";
           document.write(a,"+",b,"=",a+b);
           document.write(linebreak);
           document.write(a,"-",b,"=",a-b);
           document.write(linebreak);
           document.write(a,"*",b,"=",a*b);
           document.write(linebreak);
           document.write(a,"/",b,"=",a/b);
           document.write(linebreak);
           document.write(a,"%",b,"=",a%b);
           document.write(linebreak);
           document.write("++",a,"=",++a);
           document.write(linebreak);
           document.write("--",b,"=",--b);
           document.write(linebreak);
           //-->
```

```
    </script>
    </body>
</html>
```

Output

```
33+10=43
33-10=23
33*10=330
33/10=3.3
33%10=3
++33=34
--10=9
```

4.5.2 Comparison Operators

JavaScript supports the following comparison operators given in the below Table.

Operator	Description
= = (Equal)	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!= (Not Equal)	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
> (Greater than)	Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition becomes true.
< (Less than)	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.
>= (Greater than or Equal to)	Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition becomes true.
<= (Less than or Equal to)	Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true.

Example Program

```
<html>
   <body>
    <script type="text/javascript">
        <!--
           var a = 10;
           var b = 20;
           var linebreak = "<br />";
           document.write(a," == ",b,"=>",a==b);
           document.write(linebreak);
```

```
        document.write(a," < ",b,"=>",a<b);
        document.write(linebreak);
        document.write(a," > ",b,"=>",a>b);
        document.write(linebreak);
        document.write(a," != ",b,"=>",a!=b);
        document.write(linebreak);
        document.write(a," >= ",b,"=>",a>=b);
        document.write(linebreak);
        document.write(a," <= ",b,"=>",a<=b);
        document.write(linebreak);
    //-->
  </script>
 </body>
</html>
```

Output

10 == 20=>false
10 < 20=>true
10 > 20=>false
10 != 20=>true
10 >= 20=>false
10 <= 20=>true

4.5.3 Logical Operators

JavaScript supports the following logical operators as shown in the below Table.

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.

Example Program

```
<html>
 <body>
  <script type="text/javascript">
   <!--
      var a = true;
      var b = false;
```

```
            var linebreak = "<br />";
            document.write(a," && ",b,"=>",a&&b);
            document.write(linebreak);
            document.write(a," || ",b,"=>",a||b);
            document.write(linebreak);
            document.write(a," !&& ",b,"=>",!(a&&b));
          //-->
      </script>
    </body>
</html>
```

Output

true && false=>false
true || false=>true
true !&& false=>true

4.5.4 Bitwise Operators

JavaScript supports the following bitwise operators as shown in below Table.

Operator	Description
& (Bitwise AND)	It performs a Boolean AND operation on each bit of its integer arguments.
| (Bitwise OR)	It performs a Boolean OR operation on each bit of its integer arguments.
^ (Bitwise XOR)	It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.
~ (Bitwise Not)	It is a unary operator and operates by reversing all the bits in the operand.
<< (Left Shift)	Binary Left Shift Operator. It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4 and so on.
>> (Right Shift)	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.
>>> (Right shift with Zero)	This operator is just like the >> operator, except that the bits shifted in on the left are always zero.

Example Program

```
<html>
  <body>
    <script type="text/javascript">
      <!--
        var a = 2; // Bit presentation 10
        var b = 3; // Bit presentation 11
        var linebreak = "<br />";
        document.write(a," & ",b,"=>",a&b);
        document.write(linebreak);
        document.write(a," | ",b,"=>",a|b);
        document.write(linebreak);
        document.write(a," ^ ",b,"=>",a^b);
        document.write(linebreak);
        document.write("~",b,"=>",~b);
        document.write(linebreak);
        document.write(a," << ",b,"=>",a<<b);
        document.write(linebreak);
        document.write(a," >> ",b,"=>",a>>b);
        document.write(linebreak);
      //-->
    </script>
    </body>
</html>
```

Output

2 & 3=>2
2 | 3=>3
2 ^ 3=>1
~3=>-4
2 << 3=>16
2 >> 3=>0

4.5.5 Assignment Operators

JavaScript supports the following assignment operators as shown in Table below.

Operator	Description
= (Simple Assignment)	Assigns values from the right side operand to the left side operand.
+= (Add and Assignment)	It adds the right operand to the left operand and assigns the result to the left operand.

-= (Subtract and Assignment)	It subtracts the right operand from the left operand and assigns the result to the left operand.
***= (Multiply and Assignment)**	It multiplies the right operand with the left operand and assigns the result to the left operand.
/= (Divide and Assignment)	It divides the left operand with the right operand and assigns the result to the left operand.
%= (Modulus and Assignment)	It takes modulus using two operands and assigns the result to the left operand.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var linebreak = "<br />";
            document.write(a,"+=",b,"=>",a+=b);
            document.write(linebreak);
            document.write(a,"-=",b,"=>",a-=b);
            document.write(linebreak);
            document.write(a,"*=",b,"=>",a*=b);
            document.write(linebreak);
            document.write(a,"/=",b,"=>",a/=b);
            document.write(linebreak);
            document.write(a,"%=",b,"=>",a%=b);
            document.write(linebreak);
         //-->
      </script>
      </body>
</html>
```

Output

33+=10=>43
43-=10=>33
33*=10=>330
330/=10=>33
33%=10=>3

4.5.6 Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

(Condition)? X : Y

If condition is true? Then value X: otherwise value Y.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";
            result = (a > b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
   </body>
</html>
```

Output

```
200
```

4.5.7 typeof Operator

The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand. The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var a = 10;
            var b = "String";
            var linebreak = "<br />";
```

```
            result = (typeof b == "string" ? "B is String" : "B
is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);
            result = (typeof a == "string" ? "A is String" : "A
is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);
        //-->
    </script>
  </body>
</html>
```

Output

Result => B is String
Result => A is Numeric

4.6 DECISION MAKING

While writing a program, there may be a situation when we need to adopt one out of a given set of paths. In such cases, we need to use conditional statements that allow our program to make correct decisions and perform right actions. JavaScript supports conditional statements which are used to perform different actions based on different conditions.

The various decision making statements are the following.

- *if* Statement
- *if..else* Statement
- *if..else if* Statement
- *switch* Statement

4.6.1 *if* Statement

The *if* statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally. The syntax for a basic if statement is as follows.

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

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, we will use comparison operators while making decisions.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
        <!--
           var age = 20;
           if( age > 18 ){
              document.write("<b>Qualifies for voting</b>");
           }
        //-->
      </script>
      </body>
</html>
```

Output

Qualifies for voting

4.6.2 *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. The syntax is as follows.

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

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example Program

```
<html>
   <body>
        <script type="text/javascript">
         <!--
            var age = 15;
```

```
            if( age > 18 ){
               document.write("<b>Qualifies for voting</b>");
            }
            else{
               document.write("<b>Does not qualify for
voting</b>");
            }
         //-->
      </script>
   </body>
</html>
```

Output

Does not qualify for voting

4.6.3 *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. 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
}
```

It is a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
       <!--
```

```
        var fruit = "apple";
        if( fruit == "apple" ){
           document.write("<b>Fruit is apple</b>");
        }
        else if( fruit == "orange" ){
           document.write("<b>Fruit is orange</b>");
        }
        else if( fruit == "grapes" ){
           document.write("<b>Fruit is grapes</b>");
        }
        else{
           document.write("<b>Unknown fruit</b>");
        }
      //-->
    </script>
  </body>
</html>
```

Output

Fruit is apple

4.6.4 *switch* Statement

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.

The syntax is as follows.

```
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 it is omitted, the interpreter would continue executing each statement in each of the following cases.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var grade='A';
            document.write("Entering switch block<br />");
            switch (grade)
            {
               case 'A': document.write("Good job<br />");
               break;
               case 'B': document.write("Pretty good<br />");
               break;
               case 'C': document.write("Passed<br />");
               break;
               case 'D': document.write("Not so good<br />");
               break;

               case 'F': document.write("Failed<br />");
               break;
               default:  document.write("Unknown grade<br />")
            }
            document.write("Exiting switch block");
         //-->
      </script>
   </body>
</html>
```

Output

Entering switch block
Good job
Exiting switch block

4.7 LOOPS

4.7.1 *while* Loop

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. The syntax of *while* loop in JavaScript is as follows.

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

}
```

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var count = 0;
            document.write("Starting Loop " + "<br />");
            while (count < 5){
               document.write("Current Count : " + count + "<br
/>");
               count++;
            }
            document.write("Loop stopped!");
         //-->
      </script>
   </body>
</html>
```

Output

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop stopped!

4.7.2 *do..while* Loop

The *do...while* loop is similar to the *while* loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false. The syntax for *do-while* loop in JavaScript is as follows.

```
do{
   Statement(s) to be executed;
} while (expression);
```

Example Program

```
<html>
   <body>
```

```
      <script type="text/javascript">
         <!--
            var count = 0;
            document.write("Starting Loop" + "<br />");
            do{
               document.write("Current Count : " + count + "<br
/>");
               count++;
            }while (count < 5);
            document.write ("Loop stopped!");
         //-->
      </script>
   </body>
</html>
```

Output

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop stopped!

4.7.3 *for* Loop

The *'for'* loop includes the following three important parts namely loop initialization, test statement and iteration statement. The loop initialization is where we initialize our counter to a starting value. The initialization statement is executed before the loop begins. The test statement is 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 is where we can increase or decrease our counter. The syntax of *for* loop is JavaScript is as follows.

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

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
```

```
            var count;
            document.write("Starting Loop" + "<br />");
            for(count = 0; count < 5; count++){
               document.write("Current Count : " + count );
               document.write("<br />");
            }
              document.write("Loop stopped!");
        //-->
      </script>
   </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop stopped!
```

4.7.4 *for..in* Loop

The *for...in* loop is used to loop through an object's properties. We will discuss about objects in the next Chapter. Once we understand how objects behave in JavaScript, we will find this loop very useful. The syntax of this loop is as follows.

```
for (variablename in object){
   statement or block to execute
}
```

In each iteration, one property from object is assigned to `variablename` and this loop continues till all the properties of the object are exhausted. Following program prints the Web browser's *Navigator* object.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var aProperty;
            document.write("Navigator Object Properties<br /> ");
            for (aProperty in navigator) {
```

```
            document.write(aProperty);
            document.write("<br />");
        }
        document.write ("Exiting from the loop!");
    //-->
  </script>
 </body>
</html>
```

Output

Navigator Object Properties
vendorSub
productSub
vendor
maxTouchPoints
hardwareConcurrency
cookieEnabled
appCodeName
appName
appVersion
platform
product
userAgent
language
languages
onLine
doNotTrack
geolocation
mediaDevices
connection
plugins
mimeTypes
webkitTemporaryStorage
webkitPersistentStorage
getBattery
sendBeacon
getGamepads
getUserMedia
webkitGetUserMedia
javaEnabled
vibrate
requestMIDIAccess
budget

4.8 LOOP CONTROL

There may be a situation when we need to come out of a loop without reaching its bottom. There may also be a situation when we want to skip a part of our code block and start the next iteration of the loop. To handle all such situations, JavaScript provides *break* and ***continue*** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

4.8.1 *break* Statement

The *break* statement is used to exit a loop early, breaking out of the enclosing curly braces. Following program illustrates the use of *break* statement.

Example Program

```
<html>
   <body>
   <script type="text/javascript">
        <!--
        var x = 0;
        document.write("Entering the loop<br /> ");
        while (x < 10)
        {
           if (x == 3){
              break; // breaks out of loop completely
           }
           x = x + 1;
           document.write( x + "<br />");
        }
        document.write("Exiting the loop!<br /> ");
        //-->
     </script>
   </body>
</html>
```

Output

```
Entering the loop
1
2
3
Exiting the loop!
```

4.8.2 *continue* Statement

The *continue* statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a *continue* statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop. Following program illustrates the use of *continue* statement.

Example Program

```
<html>
   <body>
      <script type="text/javascript">
         <!--
            var x = 0;
            document.write("Entering the loop<br /> ");
            while (x < 5)
            {
               x = x + 1;

               if (x == 3){
                  continue; // skip rest of the loop body
               }
               document.write( x + "<br />");
            }
             document.write("Exiting the loop!<br /> ");
         //-->
      </script>
   </body>
</html>
```

Output

Entering the loop
1
2
4
5
Exiting the loop!

4.8.3 Using Labels to Control Loop

A label can be used with *break* and *continue* to control the flow more precisely. A label is simply an identifier followed by a colon (:) that is applied to a statement or a

block of code. We will see two different examples to understand how to use labels with break and continue. Line breaks are not allowed between the 'continue' or 'break' statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Example Program

```
<html>
  <body>
  <script type="text/javascript">
     <!--
        document.write("Entering the loop!<br /> ");
        outerloop: // This is the label name
        for (var i = 0; i < 5; i++)
        {
           document.write("Outerloop: " + i + "<br />");
           innerloop:
           for (var j = 0; j < 5; j++)
           {
              if (j > 3 ) break ; // Quit the innermost loop
              if (i == 2) break innerloop; // Do the same thing
              if (i == 3) continue outerloop; // continue the
                                                    outer loop

              document.write("Innerloop: " + j + " <br />");
           }
        }
       document.write("Exiting the loop!<br /> ");
      //-->
    </script>
  </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Outerloop: 4
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Exiting the loop!
```

4.9 FUNCTIONS

A function is a group of reusable code which can be called anywhere in our 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. We have seen functions like *alert()* and *write()* in the earlier sections. We were using these functions again and again, but they had been written in core JavaScript only once. JavaScript allows us to write our own functions as well. This section explains how to write our own functions in JavaScript.

4.9.1 Function Definition

Before we use a function, we need to define it. 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. The basic syntax is shown below.

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

The following example defines a function called `helloWorld()` that takes no parameters.

```
<script type="text/javascript">
   <!--
      function helloWorld()
      {
         alert("Hello World");
      }
   //-->
</script>
```

4.9.2 Function Calling

Following example illustrates how to call a function in JavaScript. The program will display a button "Hello" and on clicking this button, "Hello World" will be displayed.

Example Program

```
<html>
  <head>
    <script type="text/javascript">
      function helloWorld()
      {
        document.write ("Hello World!");
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="helloWorld()"
value="Hello">
    </form>
  </body>
</html>
```

Output

Click the following button to call the function

Hello

On clicking the above button, the output will be displayed as follows.

Hello World!

4.9.3 Function Parameters

We can pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma. Following program illustrates this.

Example Program

```
<html>
  <head>
    <script type="text/javascript">
      function sayHello(name, age)
      {
        document.write (name + " is " + age + " years old.");
      }
```

```
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
     <input    type="button"    onclick="sayHello('Tom',    20)"
value="Say Hello">
    </form>
  </body>
</html>
```

Output

Click the following button to call the function

Say Hello

On clicking the above button, the output will be displayed as follows.

Tom is 20 years old.

4.9.4 *return* Statement

A JavaScript function can have an optional *return* statement. This is required if we want to return a value from a function. This statement should be the last statement in a function. For example, we can pass two numbers in a function and then we can expect the function to return their sum in our calling program.

Example Program

```
<html>
  <head>
    <script type="text/javascript">
      function numberSum(a,b)
      {
        var s;
        s = a + b;
        return s;
      }
      function writeFunction()
      {
        var result;
        result = numberSum(6,3);
        document.write (result);
```

```
    }
  </script>
 </head>
<body>
  <p>Click the following button to call the function</p>
  <form>
   <input type="button" onclick="writeFunction()"
value="Sum">
  </form>
 </body>
</html>
```

Output

Click the following button to call the function

Sum

On clicking the above button, the output will be displayed as follows.

9

4.10 CONCLUSION

This Chapter gives a detailed explanation of the basic concepts of JavaScript. The syntax of JavaScript, where to place JavaScript, variables, scope of the variables, data types and identifiers are explained in detail. Different types of operators namely arithmetic operators, comparison operators, logical operators, bitwise operators, assignment operators, conditional operator and typeof operator are illustrated with examples. Decision making statements which includes *if* statement, *if..else* statement, *if..else if* statement and *switch* statement are well explained. Looping constructs which includes *while* loop, *do..while* loop, *for* loop and *for..in* loop are also explained. Loop control statements including *break* statement, *continue* statement and using labels to control loop is also explained in this Chapter. Functions which includes function definition, function calling, passing parameters to function and *return* statement are also covered in this Chapter.

REVIEW QUESTIONS

1. Explain the syntax of JavaScript.
2. Explain the different data types in JavaScript.
3. What are the rules for creating an identifier?

4. Explain local variables and global variables.
5. Explain various arithmetic operators used in JavaScript.
6. Explain various decision making statements in JavaScript.
7. Explain various types of loops used in JavaScript.
8. Explain loop control statements in JavaScript.
9. How will you define a function in JavaScript?
10. Explain how to call a function in JavaScript.
11. Explain how parameters can be passed to functions.
12. Explain the use of *return* statement.