Chapter 4

Adding Interactivity to Web Pages

A Web page may require users to enter some input and generate output. For example, you may require creating a Web page for a shopping website that enables users to select the product they want to buy from a list and enter its quantity in a text box. The moment users specify these values, the total payable amount should be generated in another text box. This type of functionality cannot be performed by HTML alone. Therefore, a scripting language is required to add such functionality to the Web page.

This chapter discusses the basics of a scripting language. It explains the process of implementing JavaScript in Web pages. In addition, it discusses the usage of expressions and control structures. Moreover, it explains the need and use of functions.

Objectives

In this chapter, you will learn to:

- Understand scripting
- Implement JavaScript in Web pages
- Use variables, operators, and control structures
- Implement functions

Understanding Scripting

Consider a scenario where you need to create a Web page for a website called OnlineShop.com. The website facilitates users to register by using an online registration form. At the time of registration, the users are asked to select either of the two options, email or Mail, as the mode of communication. Upon selecting the email option, the user is provided a text field to enter the email address. However, if the user selects the Mail option, a text area is provided to enter the postal mailing address. The users are provided the statements of their transactions, acknowledgement receipts, or promotional offer mailers at the specified address. Such Web pages can be developed by using a scripting language.

Controls, such as text boxes, radio buttons, and text areas, will be discussed in the next chapter.

Types of Scripting

To create a dynamic and interactive Web page, you need to incorporate a block of code, which is known as script, in the Web page. The script can be executed either by the Web browser or by the Web server.

When a user requests for a Web page through a Web browser, the request is sent to a computer that is placed at a different location on the World Wide Web (WWW). The computer on which the browser is running is known as the client, and the computer that receives the request is known as the Web server.

When the Web server receives the request, it processes the request and sends the requested Web page to the client, which is, then, displayed in the browser window of the client.

Consider the example of the website of a famous fortnightly science journal. It allows its customers to subscribe online for the journal. While subscribing, the user needs to provide the contact details in the form of the permanent address and communication address. The communication address of the user can be the same as the permanent address. After filling the permanent address details, if the user selects the check box titled Same as Communication Address, the same address should be filled in the Communication Address field. This functionality can be implemented by using the script that is interpreted at the client-end itself.

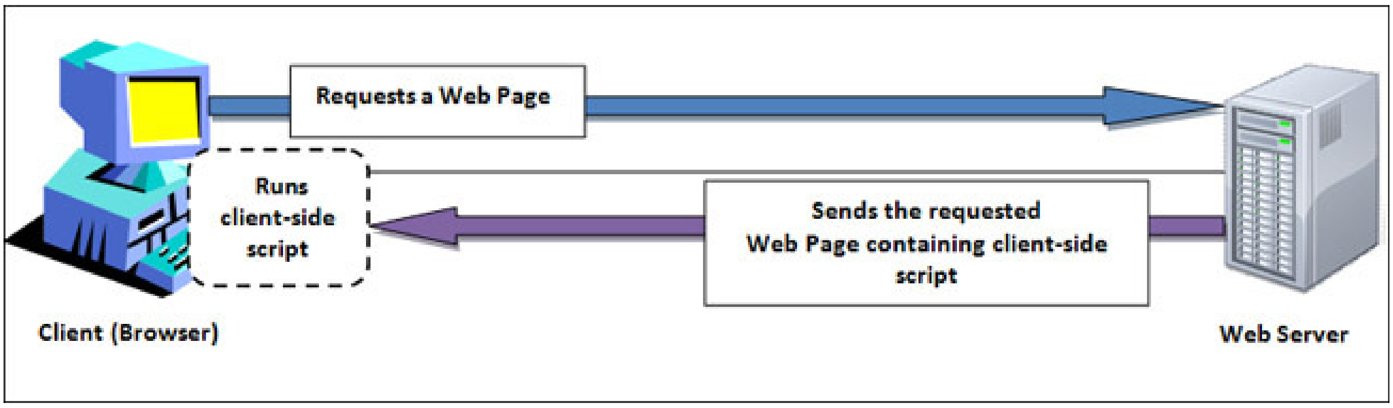
Scripting can be of two types. These are:

- Client-side scripting
- Server-side scripting

Client-side Scripting

Client-side scripting refers to the scripts that are executed at the client-side by the Web browser, running on the user's computer. Some of the languages used for creating client-side scripts are client-side JavaScript (CSJS) and Visual Basic Script (VBScript). When the Web browser requests a Web page, the server sends the requested Web page, which includes both, the HTML statement and the script statement, over the network. The Web browser reads the Web page and displays the results generated by interpreting the HTML statements. In addition, the Web browser executes the script statements as and when they are encountered while rendering the Web page.

The communication between a client and a server in case of client-side scripting is shown in the following figure.



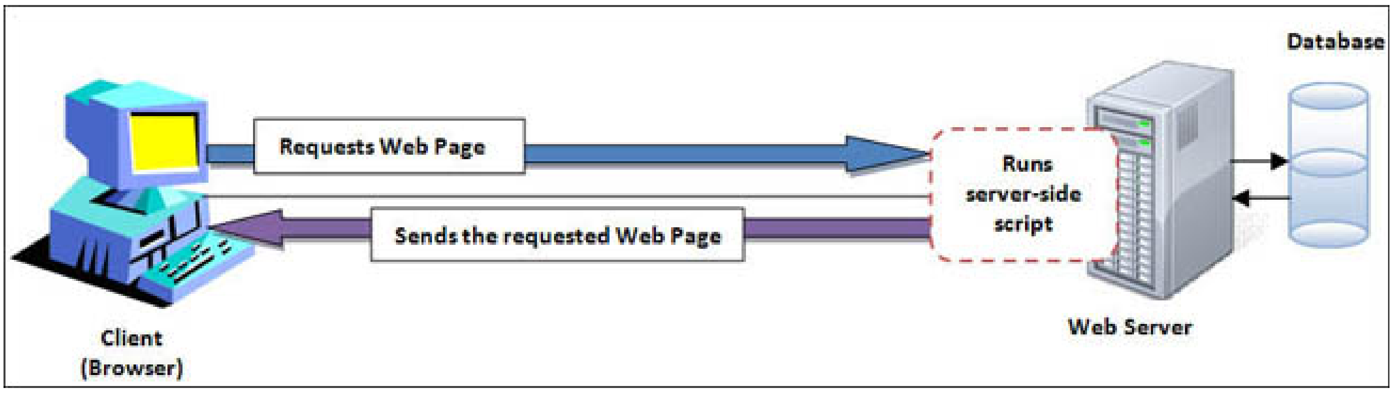
The Communication in Client-side Scripting

Server-side Scripting

Server-side scripting refers to the scripts that are executed by the Web server on the basis of the user's request. Some of the languages used for creating server-side scripts are Server-side JavaScript (SSJS), Perl, PHP, and Visual Basic Script (VBScript).

Server-side scripts are executed on the Web server. In this case, the information needs to be collected from the Web browser on the client side and is passed to a program or script that is executed on the Web server. The script executing on the Web server performs certain tasks, such as establishing database connection and verifying data sent by the Web browser from the client computer.

Server-side scripting allows database interaction and can be used to process client-side queries and store client data in the database. This feature enables users to share and access information with other users of an application or a server. The communication between a client and a server in case of server-side scripting is shown in the following figure.



The Communication in Server-side Scripting

JavaScript and VBScript can be used for client-side scripting as well as server-side scripting.

Identifying the Benefits of JavaScript

JavaScript provides the following important benefits:

- **Handle events**: JavaScript can be used to execute functions whenever an event is triggered. For example, when a user rolls the mouse over any image, its background changes.
- **Gather browser information**: JavaScript can be used to gather browser information, such as the browser name and version. This information can be useful for the server to respond to client requests.
- **Manipulate cookies**: JavaScript can be used to access and store user information, such as client preferences and authentication information of a client computer, in the form of cookies.

A cookie is a piece of data that is used to identify a user. It is stored in a user's Web browser and is sent from a Web server while the user is browsing a website.

Implementing JavaScript in Web Pages

Consider the scenario of the BookYourHotel website. While booking a hotel, the customers are requested to specify their area of interest by clicking a radio button against certain options, such as adventure sports, movies, spa, and site seeing. The customers can avail the selected facility free of cost.

If the customer selects any of these radio buttons, the details about that facility should open as a list. For example, if a customer selects the adventure sports radio button, the type of sports, such as paragliding, bungee-jumping, or river rafting, should be displayed as a list of check boxes, where the customer can select the preferred activity. Such functionalities can be implemented by using JavaScript.

JavaScript is one of the most popular scripting languages. It can be used to provide functionality to a Web page, such as populating a text box when a user selects an option in a list box. These functionalities can be triggered by the user action.

A script can be embedded directly into a Web page by writing the JavaScript code inside the `<SCRIPT>` tag or by writing the entire JavaScript code in an external JavaScript (.js) file. While using an external file for the JavaScript code, you need to refer to this file on the Web page.

Embedding a Script into a Web Page

The JavaScript code can be embedded into a Web page by using the `<SCRIPT>` tag. The following syntax is used for embedding a script into a Web page:

```
<SCRIPT type="text/javascript"> JavaScript statements
</SCRIPT>
```

In the preceding syntax, the `<SCRIPT>` and `</SCRIPT>` tags indicate the start and end of the script, respectively. The `type` attribute of the `<SCRIPT>` tag indicates the type of scripting language. One or more JavaScript statements between the `<SCRIPT>` tag and the `</SCRIPT>` tag form a script block. The script block is executed by the browser's built-in JavaScript interpreter.

The `<SCRIPT>` tag can be inserted into the body section of a Web page or the head section of a Web page or both. If the script is meant to be executed in response to an action performed by the user, it is normally placed in the head section. It helps in placing all scripts at one place without interfering with the content of the page. However, if the script needs to be executed as soon as the page is loaded, it is placed in the body section of the Web page.

Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
alert('Welcome to JavaScript');
</SCRIPT>
</BODY>
</HTML>
```

The preceding code illustrates the use of the `<SCRIPT>` tag in the body section of a Web page. It will display the message, **Welcome to JavaScript**, in a message box as soon as the Web page is loaded.

Creating and Using an External File

When you embed a script in an HTML page, it becomes difficult to manage both, the HTML code and the JavaScript code. To avoid this issue, you can write the JavaScript code in an external file and refer to the same in an HTML file. The external JavaScript file is saved with the .js extension. The following syntax is used to refer to an external JavaScript file:

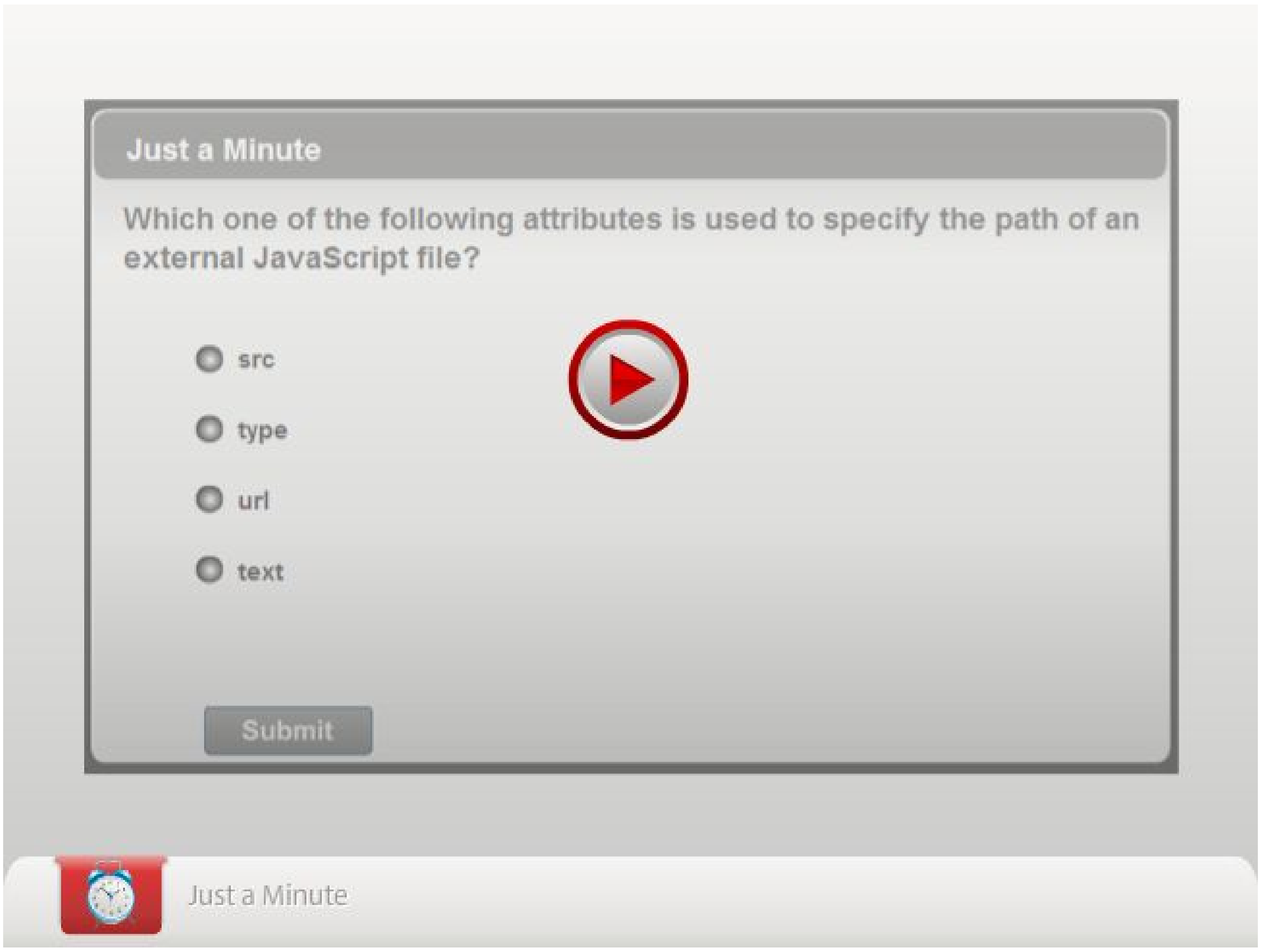
```
<SCRIPT type="text/javascript" src="URL">
```

In the preceding syntax, URL refers to the path of the external JavaScript file.

Consider the example of the ShopForYou.com website. When a user visits the home page of the website, the list of products available for sale needs to be displayed in a message box. You can write the code to display the list of products in an external JavaScript file and add a reference to the file in the home page of the website.

To create and use the external JavaScript file, you need to perform the following steps:

1. Open Notepad and write the following code:
   ```
   alert( " PRODUCTS ON SALE : \n" +" 1. LEO Mobile \n" +" 2. LEO
   Camera\n" + " 3. RED shoes \n"+" 4. KP Watch \n");
   ```
2. Save the file as **sale.js**.
3. Create the home page of the website in which you want to refer to the external JavaScript file. For this, write the following code in a Notepad file:
   ```
   <!DOCTYPE HTML>
   <HTML>
   <BODY>
   <H1>Buy Products</H1>
   </BODY>
   </HTML>
   ```
4. Save the file as **home.html**.
5. Add the highlighted code snippet, as shown in the following code in the **home.html** file:
   ```
   <!DOCTYPE HTML>
   <HTML>
   <BODY>
   <H1>Buy Products </H1>
   <SCRIPT type="text/javascript" src="sale.js">
   </SCRIPT>
   </BODY>
   </HTML>
   ```
   The `src` attribute of the `<SCRIPT>` tag is used to specify the path to the external JavaScript file, **sale.js.** The path to be specified in the `src` attribute can be either absolute or relative.



Just a Minute

Which one of the following attributes is used to specify the path of an external JavaScript file?

- src
- type
- url
- text

The absolute path is the complete address of a file. For example, if the file, sale.js, is stored in the D: drive, its absolute path is D:\sale.js. The relative path is the path of the file with respect to the current working directory. For example, if the HTML file containing the reference to the external JavaScript file is stored in the same directory as the sale.js file, the relative path of the file is given as sale.js.

Identifying Rules and Conventions Used in JavaScript

Every programming/scripting language has its own set of rules and conventions. Some of the rules and conventions for JavaScript are:

- **Semicolons**: JavaScript does not compulsorily require a semicolon to indicate the end of a statement. However, it is a good practice to insert a semicolon after a scripting statement as it enhances the readability of the code. For example, while writing two statements in the same line, you can use a semicolon to separate the statements. The following code snippet illustrates the usage of semicolons:

  ```
  alert ("Product on sale is:"); alert("1.LEO Mobile");
  ```

- **Quotes**: JavaScript allows you to use either double-quotation marks (" ") or single-quotation marks (' ') to enclose a string of characters. The following code snippet illustrates the usage of double quotes:

  ```
  alert(" PRODUCTS ON SALE : \n" +" 1.LEO Mobile \n" +" 2. LEO Camera\n"
  ```

```
+ "3. RED shoes \n"+" 4.KP Watch \n");
```

The following code snippet illustrates the usage of single quotes:

```
alert(' PRODUCTS ON SALE : \n' +' 1.LEO Mobile \n' +' 2. LEO Camera\n'
+ ' 3. RED shoes \n'+' 4.KP Watch \n');
```

- **Case sensitivity**: JavaScript is a case-sensitive scripting language. This means that the statement, a=b, and the statement, A=B, are treated differently by JavaScript.
- **Comments**: Comments are statements that are not executed by the interpreter but are used to enhance the readability and understandability of the code. JavaScript allows usage of single-line as well as multi-line comments. Single-line comments are indicated by using `//`and multi-line comments are indicated by using the symbols, `/*` and `*/`. The following code snippet illustrates the usage of a single-line comment:

```
//Comments are necessary for readability
```

The following code snippet illustrates the usage of a multi-line comment:

```
/* Comments are necessary for readability */
```

Activity 4.1: Understanding Scripting

Using Variables, Operators, and Control Structures

Consider the scenario wherein you need to create a Web page for ShopForYou.com that allows users to specify whether or not to display the list of products in the ascending or descending order of their prices.

To implement the preceding functionality on a Web page, you need to compare the prices of these items and display them either in the ascending or descending order, as requested by the user. To perform such tasks, you need to write scripts that use variables, operators, conditional constructs, and looping constructs provided in JavaScript.

Defining Variables

Consider a situation where you have created a Web page that accepts the quantity and unit price of a product that users wish to purchase and displays the total price of the product. While reading the quantity provided by the user, you need to store this value so that it can be used to calculate the total price. In addition, you need to store the price of the product.

A variable is a named location in memory that is used to store a value. In the preceding example, if the user provides 5 as the quantity of a product that needs to be purchased and the price of the product is $250, you need two variables, one for storing the quantity and the other for the price. The following code snippet shows the assignment of values to variables:

```
quantity = 5
price = 250
```

Declaring a Variable

Before using a variable in your program, you should declare it first. JavaScript allows you to declare a variable by using the `var` keyword. The syntax to declare a variable is:

```
var var_name;
```

In the preceding syntax, `var_name` refers to the name of the variable and the variable is declared by using the `var` keyword.

The following code snippet declares a variable named `employeeName`:

```
var employeeName;
```

JavaScript, which is a loosely-typed language, allows you to initialize a variable without specifying its data type. Numeric, String, and Boolean are the commonly-used data types in JavaScript.

Variables are divided into two categories, local and global. Local variables are declared in a block of code, such as within the body of a function or looping constructs. These variables can be accessed only in the specific block. Global variables are declared outside any block of code, but within the script. These can be accessed anywhere within the script.

Assigning a Value to a Variable

Values can be assigned to a variable in the following ways:

- Assigning a value to a variable after its declaration
- Initializing a variable while declaring it
- Initializing a variable without declaring it explicitly

Assigning a Value to a Variable After Its Declaration

Consider the following code snippet to initialize a variable after its declaration:

```
var employeeName;
employeeName="Peter";
```

The preceding code snippet declares the variable, `employeeName`, and then assigns the value, `Peter`, to it.

Initializing a Variable While Declaring It

Consider the following code snippet to initialize a variable while declaring it:

```
var employeeName="Peter";
```

The preceding code snippet declares the variable, `employeeName`, and assigns it the value, `Peter`.

Initializing a Variable Without Declaring It Explicitly

JavaScript provides you the flexibility to use a variable without declaring it. When a variable is used in a script without declaring it explicitly, it is automatically declared when used for the first time. However, it is a good practice to declare a variable before using it in a script as it increases the readability of the program.

Consider the following code snippet to initialize a variable without declaring it:

```
employeeName="Peter";
```

In the preceding code snippet, the variable, `employeeName`, is assigned a value but not declared explicitly. Therefore, the statement, `employeeName="Peter"`, declares the variable, `employeeName`, implicitly and assigns the value, `Peter`, to it.

An array is a special type of variable that is used to store multiple values arranged in a definite sequence. These values are stored in indexed locations within the array. Before using arrays, you need to declare them. The following syntax is used for declaring an array:

```
var category = new Array(3)
```

In the preceding syntax, an array named category is created with a size of three.
Consider the following code snippet to assign a value at each indexed location:

```
category[0] = "Soaps"
category[1] = "Oils"
category[2] = "Food Products"
```

The values stored in the array can be accessed by using the index location where they are stored. The following syntax is used for accessing the first element in the array:

```
category[0];
```

Using Operators

An operator is a set of one or more characters that is used for computations or comparisons. Operators can be used to modify the values stored in variables. The values or variables on which the operator acts are known as operands. An operand can be a literal or a variable. A literal is a constant value of any data type, such as a number, string, or boolean.

The following code illustrates the use of operators in a script:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var employeeName;
var basicSalary;
var othersAllowance; var TotalSalary; employeeName="Peter";
basicSalary=20000; othersAllowance=1000;
TotalSalary=basicSalary+othersAllowance;
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the operands used are variables `employeeName`, `basicSalary`, `TotalSalary`, and `othersAllowance` including string literal, `Peter`, and numeric literals, `20000` and `1000`. The `+` operator performs the addition operation on the value contained in `basicSalary` and `othersAllowance` and the value is assigned to the variable, `TotalSalary`.

Identifying Operator Categories

You can use the following categories of operators in JavaScript:

- Arithmetic Operators
- Assignment Operator
- Arithmetic Assignment Operators
- Comparison Operators
- Logical Operators

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and literals. The following table describes the commonly-used arithmetic operators.

Operator	***Description***	***Example***
`+`	*Used to add two numbers.*	`X=Y+Z;` *If* `Y` *is equal to 20 and* `Z` *is equal to 2,* `X` *will have the value, 22.*
`-`	*Used to subtract two numbers.*	`X=Y-Z;` *If* `Y` *is equal to 20 and* `Z` *is equal to 2,* `X` *will have the value, 18.*
`*`	*Used to multiply two numbers.*	`X=Y*Z;` *If* `Y` *is equal to 20 and* `Z` *is equal to 2,* `X` *will have the value, 40.*
`/`	*Used to divide one number by another. Returns the quotient of the division.*	`X=Y/Z;` If `Y` is equal to 21 and `Z` is equal to 2, `X` will have the value, 10.5.
`%`	*Used to divide two numbers and return the remainder. The operator is called as modulus operator.*	`X=Y%Z;` *If* `Y` *is equal to 21 and* `Z` *is equal to 2,* `X` *will contain the value, 1.*

The Arithmetic Operators

Assignment Operator

An assignment operator (=) is used to assign a value or a result of an expression to a variable. For example, the expression, `x=5`, stores the value, 5, in the variable, x.

Arithmetic Assignment Operators

Arithmetic assignment operators are used to perform arithmetic operations and assign the value to the variable at the left side of the operator.

The following table describes the commonly-used arithmetic assignment operators and their usage.

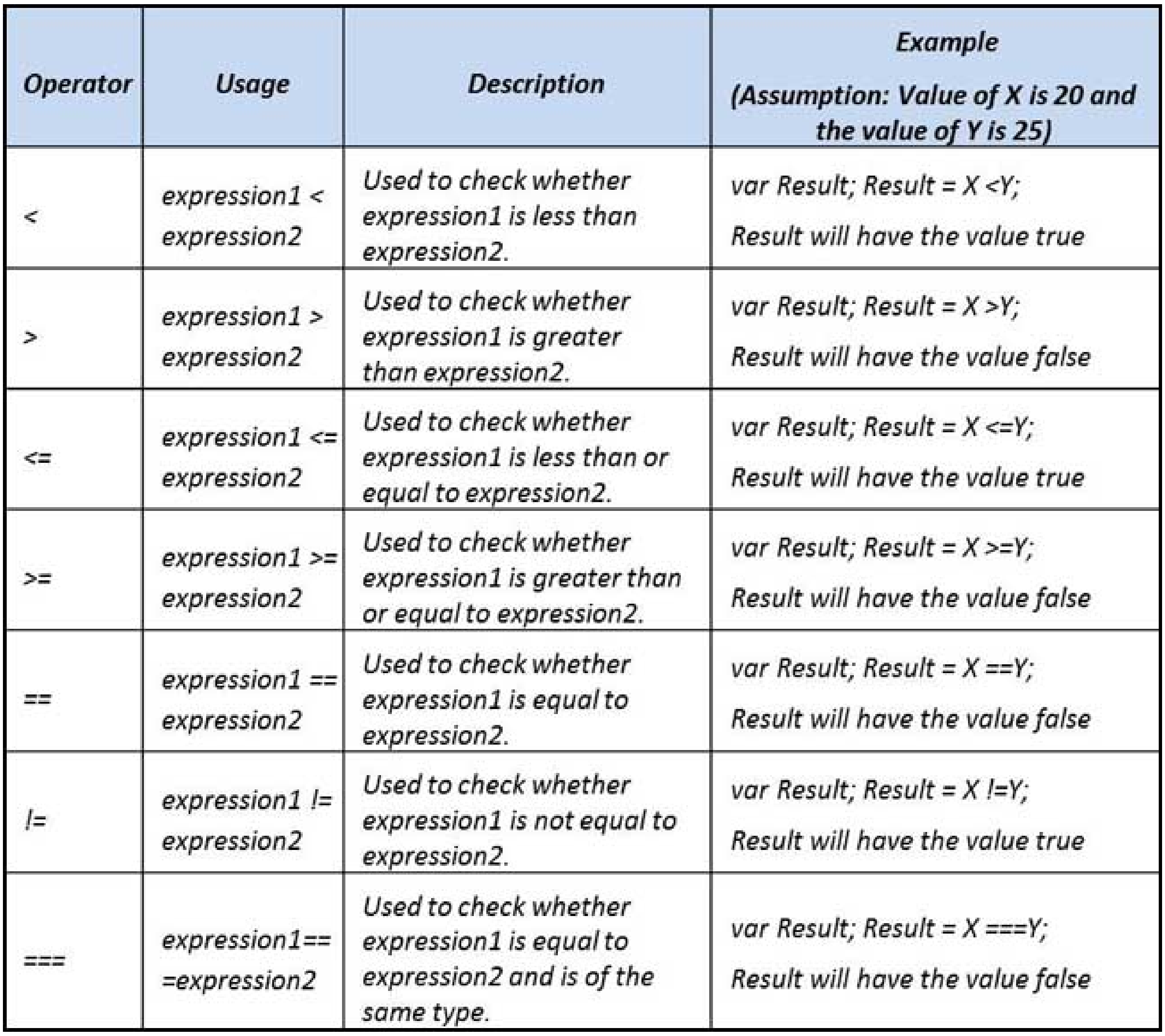
Operator	***Usage***	***Description***
`+=`	`X+=Y;`	*Same as:* `X = X + Y;`
`-=`	`X-=Y;`	*Same as:* `X = X - Y;`
`*=`	`X*=Y;`	*Same as:* `X = X * Y;`
`/=`	`X/=Y;`	*Same as:* `X = X / Y;`
`%=`	`X%=Y;`	*Same as:* `X = X % Y;`

The Arithmetic Assignment Operators

Comparison Operators

Comparison operators are used to compare two values and perform an action on the basis of the comparison. Whenever you use a comparison operator, the resulting expression holds a boolean value.

The following table describes the usage of comparison operators.

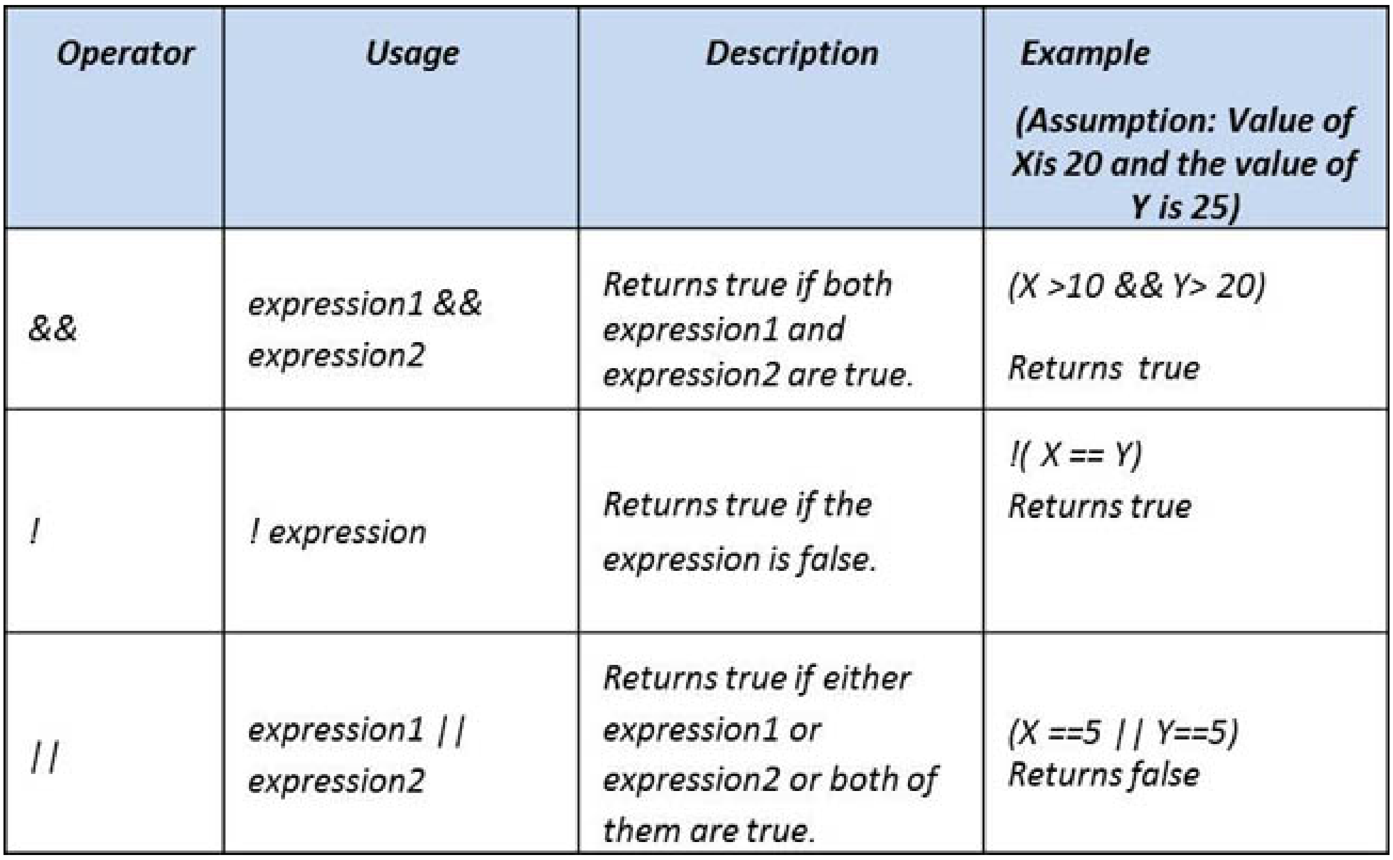


Operator	Usage	Description	Example (Assumption: Value of X is 20 and the value of Y is 25)
<	expression1 < expression2	Used to check whether expression1 is less than expression2.	var Result; Result = X <Y; Result will have the value true
>	expression1 > expression2	Used to check whether expression1 is greater than expression2.	var Result; Result = X >Y; Result will have the value false
<=	expression1 <= expression2	Used to check whether expression1 is less than or equal to expression2.	var Result; Result = X <=Y; Result will have the value true
>=	expression1 >= expression2	Used to check whether expression1 is greater than or equal to expression2.	var Result; Result = X >=Y; Result will have the value false
==	expression1 == expression2	Used to check whether expression1 is equal to expression2.	var Result; Result = X ==Y; Result will have the value false
!=	expression1 != expression2	Used to check whether expression1 is not equal to expression2.	var Result; Result = X !=Y; Result will have the value true
===	expression1===expression2	Used to check whether expression1 is equal to expression2 and is of the same type.	var Result; Result = X ===Y; Result will have the value false

The Comparison Operators

Logical Operators

Logical operators are used to evaluate complex expressions in which there is a need to evaluate a single expression or multiple expressions to assess the result. They return a boolean value. The following table describes the usage of logical operators.



Operator	Usage	Description	Example (Assumption: Value of Xis 20 and the value of Y is 25)
&&	expression1 && expression2	Returns true if both expression1 and expression2 are true.	(X >10 && Y> 20) Returns true
!	! expression	Returns true if the expression is false.	!(X == Y) Returns true
||	expression1 || expression2	Returns true if either expression1 or expression2 or both of them are true.	(X ==5 || Y==5) Returns false

The Logical Operators

Using Conditional Constructs

Consider a situation where you need to execute certain statement(s) in a script on the basis of some condition. For example, you want to check whether a given number is even or odd and display a message accordingly. For this, you need to make decisions in the script code and execute a different set of statements depending upon the decision taken.

You can do this by using conditional constructs. These constructs allow you to execute a selective statement or a block of statements based on the result of the expression being evaluated. The two conditional constructs in JavaScript are:

- The `if...else` construct
- The `switch...case` construct

The if...else Construct

The `if` statement in the `if...else` conditional construct is followed by a logical expression in parenthesis. This condition is evaluated and a decision is made on the basis of the result. The following statements depict the syntax of the `if...else` construct:

```
if (exp)
{
// Statements;
}
else
```

```
{
// Statements;
}
```

In the preceding syntax, the expression, `exp`, is evaluated. If the result is true, the statements inside the `if` construct are executed. If the result is false, the statements inside the `else` construct are executed.

Consider the example of a game where you need to validate the age of a player. If the age is greater than 12, the player is allowed to play the game. Otherwise, an appropriate message needs to be displayed. The following code snippet shows the usage of the `if...else` construct:

```
var age=5;
if (age<12)
{
alert('Sorry! This game is for children above 12 Years');
}
else
{
alert('Play the Game');
}
```

The preceding code snippet checks whether the age of the player is less than 12. The condition in the if statement checks the age of the player. If the condition evaluates to true, the message, **Sorry! This game is for children above 12 Years**, is displayed. If the condition evaluates to false, the message, **Play the Game**, is displayed.

The switch...case Construct

Another conditional construct available in JavaScript is the `switch...case` construct. It is used when you need to evaluate a variable for multiple values.

The following code depicts the syntax for the `switch...case` construct:

```
switch(VariableName)
{
case ConstantExpression_1:
// statements;
break;
case ConstantExpression_2:
// statements;
break;
case ConstantExpression_n:
// statements;
break;
default:
// statements;
break;
}
```

When the `switch` statement is executed, the variable passed as a parameter to the `switch` statement is evaluated and individually compared with each constant expression specified with each case statement. If one of the constant expressions is equal to the value of the variable given in the `switch` statement, the control is passed to the statement following the matched case statement. A `break` statement is used to exit the `switch` statement. This prevents execution of the remaining case structures by ending the execution of the `switch...case` construct. If none of the cases match, the statements under the default statement are executed.

Consider the following code where you want to display the name of the day of the week depending on the value of a variable:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var day="5";
switch(day)
{
case "1":
alert("Day is Monday");
break;
case "2":alert("Day is Tuesday");
break;
case "3":
alert("Day is Wednesday");
break;
case "4":
alert("Day is Thursday");
break;
case "5":
alert("Day is Friday");
break;
case "6":
alert("Day is Saturday");
break;
case "7":
alert("Day is Sunday");
break;
default:
alert("Not a valid number");
break;
}
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `switch` statement is used to evaluate and compare the value of the variable, `day`, with the case constants and display the name of the day.

Using Loop Constructs

Loop structures are used to repeatedly execute one or more lines of code. In JavaScript, the following loop structures can be used:

- The `while` loop
- The `do...while` loop
- The `for` loop

The while Loop

The `while` loop is used to repeatedly execute a block of statements till a condition evaluates to true. The `while` statement always checks the condition before executing the statements in the loop. The syntax for the `while` loop construct is:

```
while (expression)
{
statements;
}
```

In the preceding syntax, the expression is evaluated. If the result is true, the statements in the body of the loop are executed. Once all the statements in the block are executed, the control passes back to the loop and the expression is re-evaluated. The loop exits when the expression evaluates to false.

The following code illustrates the use of the `while` loop:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var num=0;
while(num<20)
{ num=num+1; alert(num);
}
</SCRIPT>
</BODY>
</HTML>
```

The preceding code defines the variable, `num`, and initializes the same to the value, `0`. The `while` statement checks whether the value of `num` is less than `20`. If the condition evaluates to true, the statements within the `while` loop are executed. This process continues till the value of `var` is less than `20`.

The do...while Loop

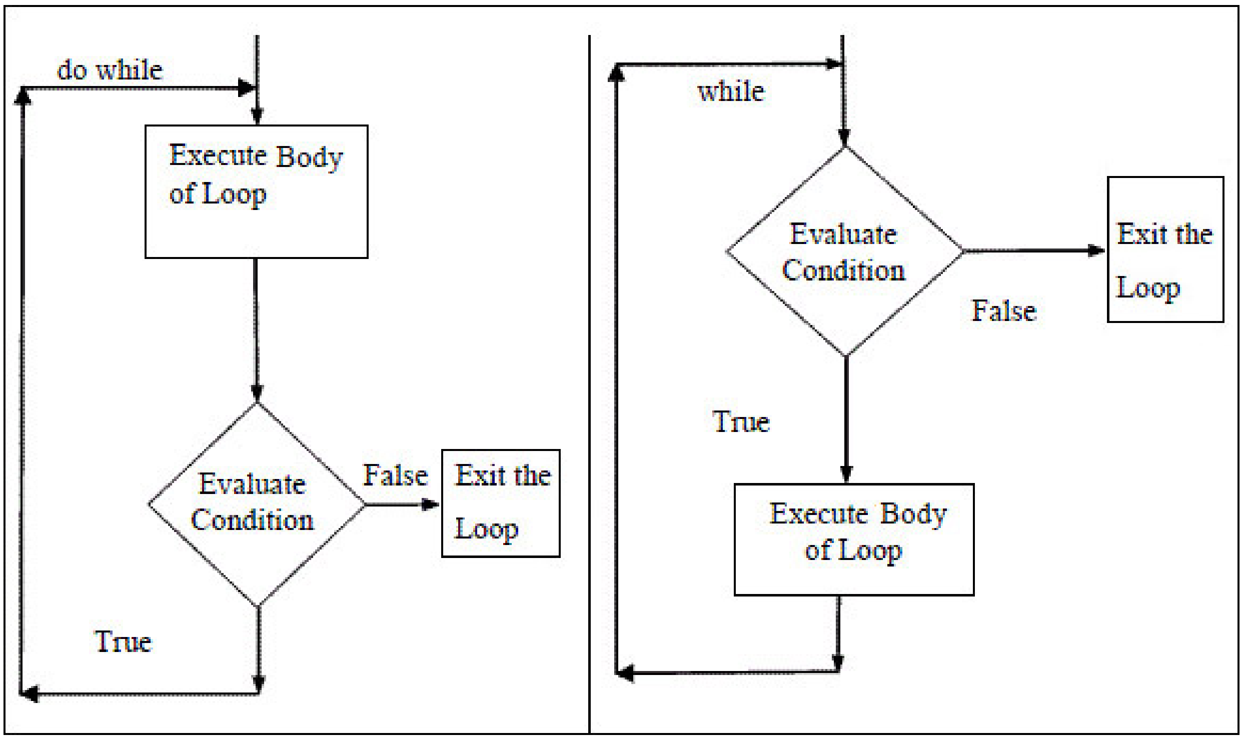
The `do...while` loop construct is similar to the `while` loop construct. However, the statements within the `do...while` loop are executed at least once, in comparison to the `while` loop, where the statements in the block are not executed when the condition evaluates to false. In addition, the statements within the `do-while` loop are executed before the condition is checked as compared to the `while` loop, where the statements within the block are executed after the condition is checked. The following syntax is used to declare the `do-while` loop:

```
do
{ Statements;
}
while(condition)
```

Consider the following code that illustrates the use of the `do-while` loop:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var num=0;
do
{
num=num+1;
alert(num);
}
while(num<20)
</SCRIPT>
</BODY>
</HTML>
```

The difference between the functionality of the `do...while` loop construct and that of the while loop construct is shown in the following figure.



The Functioning of the do...while and while Loops

The for Loop

The `for` loop allows the execution of a block of code depending on the result of the evaluation of the test condition. The following syntax is used to declare the `for` loop:

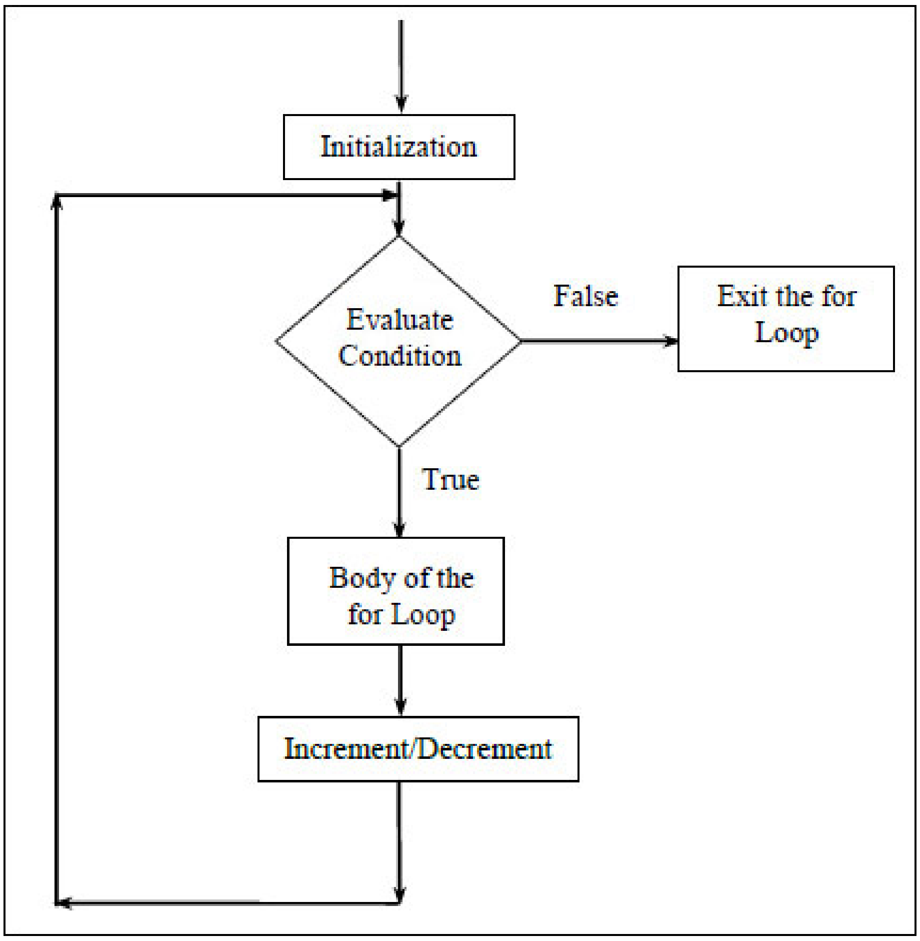
```
for (initialize variable; test condition; step value)
{
// code block
}
```

In the preceding syntax:

- `initialize variable`: Initializes the loop variable to a value.
- `test condition`: Specifies the condition to be checked before executing statements in the code block.
- `step value`: Indicates the increment or decrement to be performed for every iteration.

The execution of the `for` loop starts with the initialization of the loop variable. It, then, evaluates the test condition. If the test condition evaluates to true, the code block in the body of the loop is executed. If it evaluates to false, the `for` loop is exited. Once all the statements in the code block are executed, the variable is incremented or decremented as specified in the step value. Once the value is iterated, the test condition is re-evaluated. This process continues till the test condition evaluates to true.

The sequence of execution of the `for` loop is shown in the following figure.



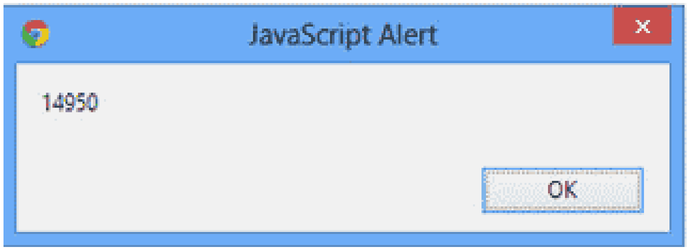
The Sequence of Execution of the for Loop

The following code illustrates the use of the `for` loop:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var num;
var sum=0;
for(num=100;num<200;num=num+1)
{
sum=sum+num;
}
alert(sum);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code creates variables named `num` and `sum`. In the `for` loop, the value, `100`, is assigned to the `num` variable. The condition inside the `for` loop checks whether the value of `num` is less than `200`. If the condition evaluates to true, the statements within the `for` loop are executed. This process continues till the value of the `num` variable is less than `200`.

The output of the preceding code will be displayed, as shown in the following figure.



```
14950
```

The Code Output

The working of the `while` loop is similar to that of the `for` loop. The main difference is that the variable is incremented within the code block of the `while` loop as compared to the `for` loop, where the variable is incremented after all the statements in the code block are executed.

Break and Continue Statements

In some situations, there may be a need to exit a loop before the loop condition is checked after iteration. The `break` statement is used to exit the loop. It prevents the execution of the remaining statements of the loop. The `break` statement is usually placed within an `if` construct inside the loop. The `continue` statement is used to skip all the subsequent instructions and take the control back to the beginning of the loop.

The following syntax is used to show the use of the `break` statement inside the `while` loop:

```
while(test condition)
{
//Statement1;
if (test condition1)
{
break;
}
//Statement2;
//Statement3;
}
```

In the preceding syntax, if `test condition` is true, the control enters the body of the loop. It executes `statement1`. Next, it checks test condition1. If it evaluates to true, the control is transferred outside the loop. As a result, `Statement2` and `Statement3` are not executed. If `test condition1` evaluates to false, the loop continues its normal course of execution. As a result, `Statement2` and `Statement3` are executed.

Consider the following code that illustrates the use of the `break` and `continue` statements:

```
<!DOCTYPE HTML>
<HTML>
<HEAD></HEAD>
```

```
<BODY>
<SCRIPT type="text/javascript">
var iNum = 0;
for (var i=1; i < 10; i++)
{
if (i % 3 == 0)
{
break;
}
iNum++;
}
alert(iNum);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code creates the variable, `iNum`, and assigns the value, `0`, to it. The `for` loop iterates the variable, `i`, from 1 to 10. The `if` statement in the `for` loop validates if the value of `i` is divisible by `3`. If `i` is divisible by `3`, the `break` statement is executed and the control comes out of the `for` loop. When the preceding code is executed, the value, `2`, is displayed.

However, if the preceding code is executed by replacing `break` with the `continue` statement, a different output is displayed. For example, consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT type="text/javascript">
var iNum = 0;
for (var i=1; i < 10; i++) {
if (i % 3 == 0) {
continue;
}
iNum++;
}
alert(iNum);
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `if` statement in the `for` loop validates if the value of `i` is divisible by `3`. If `i` is divisible by `3`, the `continue` statement is executed and the control comes back to the beginning of the `for` loop. When the preceding code is executed, the value, `6`, is displayed.

Implementing Functions

Consider the ShopForYou.com website. The company is required to calculate the tax levied on the products purchased by the customers. The value of tax varies from item to item. For example, the tax on certain products, such as medicine, is 0%, and the tax on mobile products is 12.5%. The code that calculates the tax on various categories of products needs to be used at various locations in the script of your page. If the code is repeated at all these locations, any change required in the code will need to be made at multiple locations. For example, if you want to change the tax rate from 12.5% to 10%, you will need to make this change at all the locations where this code is used. In addition, if any reused piece of code contains an error, the error needs to be corrected at multiple locations.

To overcome this problem, functions are introduced. You can write the code that needs to be reused inside a function. Now, instead of repeating the use of code, you can just make a call to the function that contains the required code. This enables you to write the code only once and use it whenever required. Thus, functions are used to optimize the performance by implementing the concept of reusability.

Introducing Functions

A function is a self-contained block of statements that has a name. Functions can be executed whenever the same code is required to be performed repeatedly at different locations in a script.

A function is created as a separate module with a name attached to it. Each function can have several statements within it. When functions are used, you can easily detect the error in the script by comparing the expected result of execution of each function with the actual result of execution of that function. If the results differ, it means that there is an error in the code. Thus, it becomes easier to debug the code.

In other words, functions make your code modular, simple, and reusable. In JavaScript, functions are of the following types:

- Built-in functions
- User-defined functions

Built-in Functions

Built-in functions are ready to use as they are already coded. Some of the built-in functions supported by JavaScript are:

- `isNaN()`
- `parseInt()`
- `parseFloat()`
- `eval()`
- `prompt()`
- `confirm()`

`isNaN()`

The `isNaN()` function determines whether a parameter is not a number. The function returns true if the parameter is not a number.

The following code illustrates the use of the `isNaN()` function:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
num1="Two thousand"; num2=8000; Result=isNaN(num1); alert(Result);
Res=isNaN(num2); alert(Res);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code creates the variable, `num1`, and assigns the value, `Two thousand`, to it. It also creates a variable, `num2`, and assigns the value, `8000`, to it. The `isNaN()` function evaluates whether the variables, `num1` and `num2`, contain numeric data or not. As the value in the `num1` variable is a string, the `isNaN(num1)` function will return true. However, the value in `num2` is numeric. Therefore, the `isNaN(num2)` function will return `false`.

NaN stands for Not a Number.

`parseInt()`

The `parseInt()` function parses the string parameter and returns the corresponding integer.

Consider the following code snippet:

```
x="5";
y=parseInt(x);
```

In the preceding code snippet, the variable, `x`, stores the string, `5`. The `parseInt()` function takes the parameter, `x`, parses it and assigns the integer value, `5`, to the variable, `y`.

`parseFloat()`

The `parseFloat()` function takes a string parameter and returns a floating point number. Consider the following code snippet:

```
x="6.2";
y=parseFloat(x);
```

In the preceding code snippet, the variable, `x`, stores the string, `6.2`. The `parseFloat()` function takes the parameter, `x`, parses it, and assigns the floating point number, `6.2`, to the variable, `y`.

`eval()`

The `eval()` function is used to evaluate or execute a parameter. If the parameter is an expression, the expression is evaluated. However, if the parameter is a JavaScript statement, the statement is executed.

Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var res1=0; res1=eval(5+10); alert(res1);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code evaluates the expression, `5+10`, and displays the result.

Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
eval("num1=5;num2=10;res1=num1+num2;alert(res1);");
</SCRIPT>
</BODY>
</HTML>
```

The preceding code evaluates the JavaScript statements written inside the `eval()` function and displays the result in the message box.

`prompt()`

The `prompt()` function is used to display a prompt dialog box, which allows a user to input a value. The prompt dialog box contains two buttons, **OK** and **Cancel**. If the user clicks the **OK** button, the `prompt()` function returns the value entered by the user. However, if the user clicks the **Cancel** button, a null value is returned.

The following code illustrates the use of the `prompt()` function:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var name=prompt("Please Enter Your Name","John");
alert(name);
</SCRIPT>
</BODY>
```

```
</HTML>
```

The preceding code prompts a user to enter the name by using the `prompt()` function and displays the name in a message box. The `prompt()` function contains two parameters. The first parameter, **Please Enter Your Name**, asks the user to provide the name and the second parameter specifies **John** as the default name. The first parameter displays a message in a dialog box, and it is a required parameter. On the other hand, the second parameter specifies the default text and it is an optional parameter.

The `prompt()` function always returns a string value. Therefore, if you need to work with data other than string, you need to cast or convert the string value that was returned by the `prompt()` function.

The following code shows how to convert the string value returned by `prompt()`:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT>
var a,b,c;
a=parseInt(prompt("Enter number 1:"));
b=parseInt(prompt("Enter number 2:"));
c=a+b;
document.write(c);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code prompts the user to enter two numbers, and then converts them by using the `parseInt()` function.

NOTE *The `document.write()` function is used in JavaScript to display a string output directly on the Web page. You will learn about it in detail in the coming chapters.*

confirm()

The `confirm()` function is used to display a dialog box that will enable a user to verify or accept a task. This dialog box contains two buttons, **OK** and **Cancel**. If a user clicks the **OK** button, the `confirm()` function returns true. If the user clicks the **Cancel** button, the `confirm()` function returns false.

The following code illustrates the use of the `confirm()` function:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
var response;
```

```
response=confirm("Do You Wish to Continue");
if(response==true){
alert("You can proceed further");
}
else
{
alert("You cannot proceed further");
}
</SCRIPT>
</BODY>
</HTML>
```

If the user clicks the **OK** button, the preceding code displays the message, **You can proceed further**. However, if the user clicks the **Cancel** button, the message, **You cannot proceed further**, is displayed.

User-defined Functions

JavaScript enables you to define your own functions according to your needs. For example, you need to calculate the average marks of students by accepting the marks from them. In this case, you can create a user-defined function, `Average()`, and call this function whenever you need to calculate the average marks. It enhances the modularity and efficiency of the code.

Creating Functions

Functions are created by using the keyword, `function`, followed by the function name and the parentheses, `()`. A function is normally defined in the head section of a Web page.

The following syntax is used to create functions:

```
function [functionName] (Variable1, Variable2)
{
//function statements
}
```

A user-defined function can optionally accept a list of parameters. The parameters that a function accepts are provided in parentheses and separated by commas. In the given code, `Variable1` and `Variable2` represent the parameters passed to the function. The function can use the values passed in these parameters to perform certain operations.

Consider the following code snippet to illustrate the creation of a function:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function tax(product_category,product_name,price)
```

```
{
if(product_category=="Mobile")
{
total_price=(12.5/100)*price+price;
alert("Total price of " +product_name +" is: $" +total_price);
}
if(product_category=="Medicine")
{
total_price=price;
alert("Total price of " +product_name +" is: $" +total_price);
}
}
</SCRIPT>
</HEAD>
<BODY>
. . . .
</BODY>
</HTML>
```

In the preceding code snippet, the `tax()` function is created that accepts three parameters, `product_category`, `product_name`, and `price`. The `tax()` function is used to calculate the total price of a product after adding the tax amount to the price of the product.

Accessing Functions

Once you have created a function, you can call the same from any portion of the code where you want to use the functionality provided by the function. A function can return a value of any data type to the calling code. Let us see how a function is called, how a value from a function is returned, and how the value returned by a function is retrieved.

Calling a Function

A function is called by using the function name. The following syntax is used for accessing a function:

```
functionName ();
```

In the preceding syntax, `functionName` is the name of the function.

While calling a function, you can also pass a variable or a value to it. The variable or the value passed to a function is known as a parameter. If parameters need to be provided, these are specified in parenthesis while calling the function. The following syntax is used for specifying parameters in a function:

```
functionName (parameter1, parameter 2...);
```

The following code illustrates how a function is accessed:

```
<!DOCTYPE HTML>
```

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
. . .
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript"> product_category="Medicine";
tax(product_category,"Paracetamol",8000); product_category="Mobile";
tax(product_category,"GV3",3000); tax(product_category,"XV5",5000);
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `tax()` function is called and `product_category`, `product_name`, and `price` are passed as parameters. The `tax()` function displays the total price after adding the tax amount to the price of the product. If `product_category` is `Medicine`, no tax amount is added to the price of the product. However, if `product_category` is `Mobile`, 12.5 % of the product price is added as the tax.

Returning Values from a Function

A function can return a value through the `return` statement. The following code illustrates how a value from a function is returned:

```
function functionName()
{
var variable=10;
return variable;
}
```

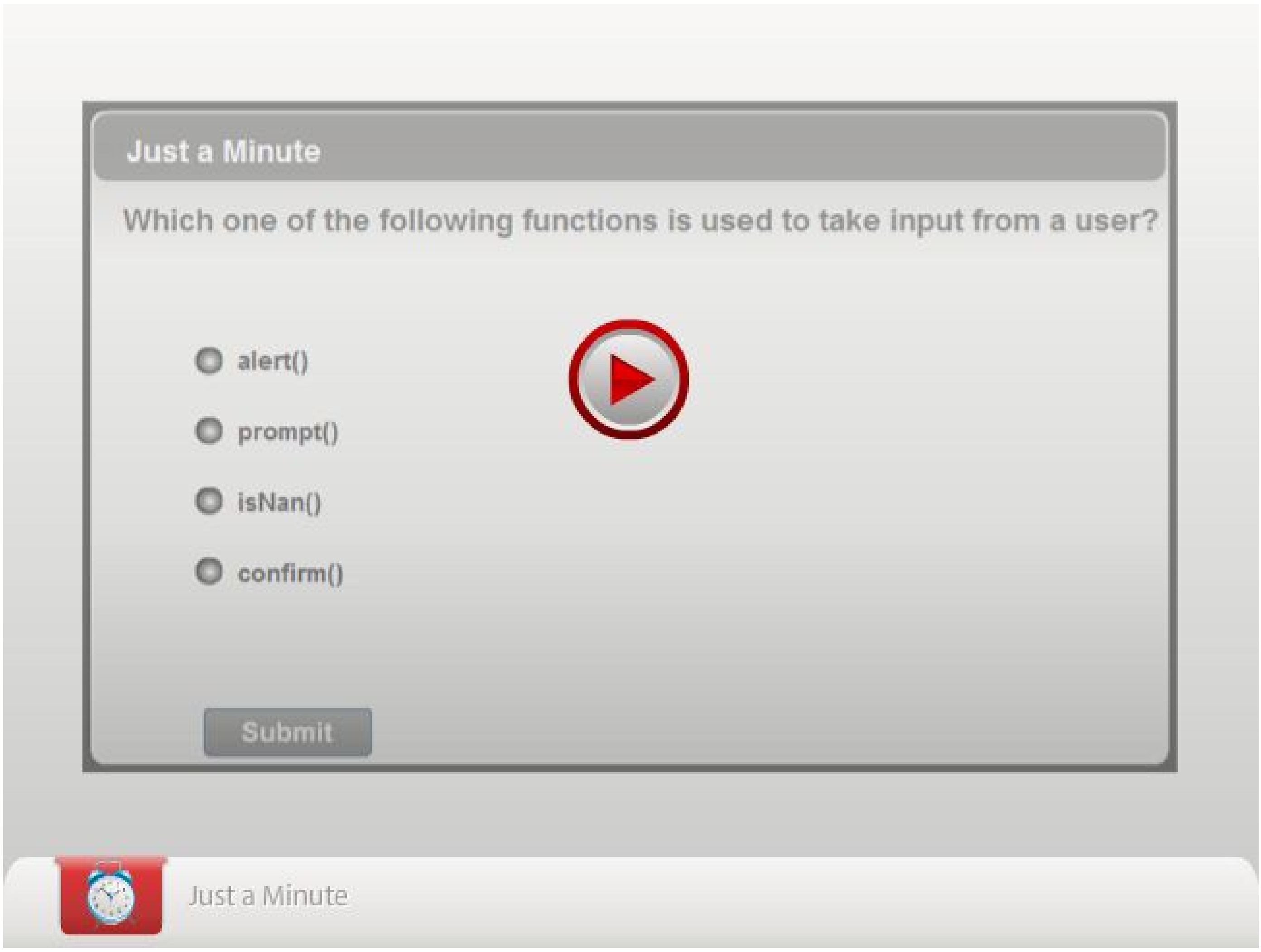
Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function sum(a,b)
{
return a+b;
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript"> total = sum (3, 6);
```

```
document.write(total);
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `sum()` function accepts two numbers, `3` and `6`, as parameters and returns their sum, 9.

> NOTE *A function can accept more than one parameter but can return only one value.*



Just a Minute

Which one of the following functions is used to take input from a user?

- alert()
- prompt()
- isNan()
- confirm()

Activity 4.2: Implementing Functions

Summary

In this chapter, you learned that:

- To create a dynamic and interactive Web page, you need to incorporate a block of code, which is known as script, in the Web page.
- Client-side scripting refers to the scripts that are executed at the client-side by the Web

browser, running on the user's computer.

- Server-side scripting refers to the scripts that are executed by the Web server on the basis of the user's request.
- A script can be embedded directly into a Web page by writing the JavaScript code inside the `<SCRIPT>` tag or by writing the entire JavaScript code in an external JavaScript (.js) file.
- The external JavaScript file is saved with the .js extension.
- Values can be assigned to a variable in the following ways:
    - Assigning a value to a variable after its declaration
    - Initializing a variable while declaring it
    - Initializing a variable without declaring it explicitly
- An operator is a set of one or more characters that is used for computations or comparisons.
- You can use the following categories of operators in JavaScript:
    - Arithmetic operators
    - Assignment operators
    - Arithmetic Assignment Operators
    - Comparison operators
    - Logical operators
- The two conditional constructs in JavaScript are:
    - The `if...else` construct
    - The `switch...case` construct
- In JavaScript, the following loop structures can be used:
    - The `while` loop
    - The `do...while` loop
    - The `for` loop
- A function is a self-contained block of statements that has a name.
- Some of the built-in functions supported by JavaScript are:
    - `isNaN()`
    - `parseInt()`
    - `parseFloat()`
    - `eval()`
    - `prompt()`
    - `confirm()`
- Functions are created by using the keyword, `function`, followed by the function name and the parentheses, `()`.

Reference Reading

Understanding Scripting

Reference Reading: Books	***Reference Reading: URLs***
JavaScript: the complete reference By Thomas A. Powell, Fritz Schneider	*http://www.w3schools.com/web/web_javascript.asp* *http://www.w3schools.com/web/web_scripting.asp*

Implementing JavaScript in Web Pages

Reference Reading: Books	***Reference Reading: URLs***
The Definitive Guide to HTML5 By Adam Freeman	*http://www.w3schools.com/js/default.asp*

Using Variables, Operators, and Control Structures

Reference Reading: Books	***Reference Reading: URLs***
The Definitive Guide to HTML5 By Adam Freeman	*http://softearth.tripod.com/Books/Using_Javascript/ch2.htm*

Implementing Functions

Reference Reading: Books	***Reference Reading: URLs***
The Definitive Guide to HTML5 By Adam Freeman	*http://www.w3schools.com/js/js_functions.asp*