```
                left:0px;
                top:0px;
                z-index:-1;
            }
        </STYLE>
```

Introducing JavaScript

JavaScript is a client and server-side object-based scripting language that is used to make interactive Web pages. A scripting language is a lightweight programming language with less complexity. In addition to JavaScript, some other examples of scripting languages are VBScript,. Among these, JavaScript is the most commonly used scripting language to add dynamism and interactivity to Web pages. This is because JavaScript, written on the client side, executes on a client browser, thereby reducing the load on the server.

JavaScript is an interpreted language, which implies that scripts written in JavaScript are processed line by line. These scripts are interpreted by the JavaScript interpreter, which is a built-in component of the Web browser. JavaScript can be written on the client side as well server side. Client-side JavaScript allows you to validate only those programs that execute and produce the result on the client-machine. In contrast, server-side JavaScript validates only those programs that execute on the server. JavaScript includes various built-in objects and features that can be used to make your HTML pages dynamic. Moreover, it is platform independent, which implies that you need to write the script once and can run it on any platform or browser without affecting the output of the script.

Using JavaScript in an HTML Document

You can insert JavaScript code in an HTML document by using the SCRIPT element. When an HTML document with the SCRIPT element is loaded in a Web browser, the browser processes the content enclosed within the SCRIPT element as JavaScript code. Either a scripting statements or a reference to an external script file is contained by the SCRIPT element. This element provides the src attribute, which allows you to add the reference of an external script file in an HTML document. The SCRIPT element contains five attributes: async, type, charset, defer, and src. Table 13 describes the attributes of the SCRIPT element:

Table 13: Attributes of the SCRIPT Element

Attributes	Values	Description
async	true false	Specifies whether the script should be executed asynchronously or not
type	text/ecmascript text/javascript application/ecmascript application/javascript text/vbscript	Specifies the Multipurpose Internet Mail Extensions (MIME) type of script
charset	charset	Specifies the character encoding used in the script
defer	true false	Specifies whether the browser can continue parsing the Web page or not
src	URL	Specifies the URL of a file that contains the script

Scanned by CamScanner

You can use the SCRIPT element in a Web page in the following three ways:

- In the HEAD element
- In the BODY element
- As an external script file

Let's discuss each of these in detail.

JavaScript in the HEAD Element

You can place the SCRIPT element inside the HEAD element of an HTML document. The script placed inside the HEAD element runs when you perform some action, such as click a link or the submit button. The following code snippet shows how to use the SCRIPT element inside the HEAD element of an HTML document:

```
<HEAD>
<SCRIPT type="text/ javascript">
Script code here
</SCRIPT>
</HEAD>
```

In the preceding code snippet, the SCRIPT element is used inside the HEAD element. In the SCRIPT element, the type attribute is used to specify the type of the script.

JavaScript in the BODY Element

You can also write the SCRIPT element inside the BODY element of an HTML document. The SCRIPT element written within the BODY element executes when the Web browser starts loading of the Web page. The following code snippet shows how to use the SCRIPT element inside the BODY element of an HTML document:

```
<BODY>
<SCRIPT type="text/ javascript">
Script code here
</SCRIPT>
</BODY>
```

JavaScript in an External File

When the JavaScript code created in an HTML document is very lengthy, it affects the readability of the HTML document. In addition, at times, you may need to use the same JavaScript code in several Web pages. In such cases, you can store the JavaScript code in an external file and save that file using the .js extension. Next, you need to link the external file with the HTML document by using the src attribute of the SCRIPT element to access the script file. The following code snippet shows how to link the external script file with the HTML document:

```
<HEAD>
<SCRIPT src="URL of the External file">
</SCRIPT>
</HEAD>
```

The preceding code snippet links the external script file with the HTML document by using the src attribute of the SCRIPT element, which specifies the URL or path of the external script file.

After discussing the use of JavaScript in an HTML document, let's discuss the programming fundamentals of JavaScript.

Exploring Programming Fundamentals of JavaScript

JavaScript helps you to create interactive Web pages and embed JavaScript statements directly in an HTML document. It also shares the fundamental features of all programming languages. For example, JavaScript can retrieve data from a source, process it, and display the output. The programming fundamentals of JavaScript include:

- Variables
- Operators
- Control flow statements
- Popup boxes

Let's discuss each of these in detail.

Exploring Variables

In JavaScript, data can be temporarily stored in variables, which are the named locations in the memory. A variable has a name, value, and memory address. The name of the variable uniquely identifies the variable, the value refers to the data that is stored in the variable, and the memory address refers to the memory location of the variable. The variable must be declared before storing any data in it. While declaring a variable, you need to provide a unique user-friendly name for it. The following syntax is used to declare a variable:

```
var variable_name;
```

In the preceding syntax, `var` is a keyword and `variable_name` represents the name of the variable. You can also declare multiple variables in the same statement by separating each variable with a comma (,), as shown in the following code snippet:

```
var variable1, variable2, variable3;
```

In the preceding code snippet, three variables named `variable1`, `variable2`, and `variable3` are declared by using the `var` keyword.

JavaScript allows you to store multiple types of values in a variable. For example, the same variable can store both a string and a number at different times in the script. This is why JavaScript is considered as a poorly typed language.

In JavaScript, you can assign values to a variable either at the time of declaration or after that. In addition, you can assign values to a variable as many times as you want according to your requirement. However, the variable stores only the most recent value assigned to it. After assigning a value to a variable, you can use it in the script. You can access the value of a variable by using its name in the script. The syntax to assign a value to a variable at the time of declaration is as follows:

```
var variable_name= value;
```

The following syntax shows how to declare and assign a value to a variable:

```
var variable_name;
variable_name= value;
```

In the preceding syntax, `variable_name` refers to the name of the variable, equal (=) refers to the assignment operator, and value refers to the value that is to be stored in the variable.

Exploring Operators

It is known that variables are used to temporarily store the data that you want to use in a script. You can modify or change the data by manipulating the respective variables. For example, suppose you have declared two variables named marksMaths and marksScience and wanted to add the values of these two variables. For this, you need to manipulate these variables to get the desired result. One of

Scanned by CamScanner

the most common ways of manipulating variables is by using operators. An operator is a symbol or word that is reserved for a special task or action. Every operator works on one or more operands, that is, an operator can take the values of more than one operand. After taking the values, it performs an action on the operands and returns the result of that action.

In JavaScript, there is a whole gamut of operators that you can use as per your requirements. Some of the operators work on a single operand, while some work on two or more operands. Moreover, some operators work on numbers, while others work on strings and Boolean values. Table 14 describes a list of operators:

Table 14: Operators Available in JavaScript

Operator	Description	Example
Arithmetic Operators		
+	Adds two numbers or joins two strings. This operator also represents a positive number when it is prefixed to a number.	45+10 returns 55 "My " + "Name" returns "My Name"
-	Subtracts two numbers or represents a negative number.	-45+10 returns -35
*	Multiplies two numbers.	45*10 returns 450
/	Divides two numbers and returns the quotient.	45/10 returns 4.5
%	Divides two numbers and returns the remainder.	45%10 returns 5
++	Increments the value of a number by one. It can be prefixed or suffixed to a number. When prefixed, the value is incremented in the current statement, and when suffixed, the value is incremented after the current statement.	myVar1=45 myVar2=++myVar1 assigns 46 to myVar2 myVar2=myVar1++ assigns 45 to myVar2
--	Decrements the value of a number by one. It can be prefixed or suffixed to a number. When prefixed, the value is decremented in the current statement, and when suffixed, the value is decremented after the current statement.	myVar1=45 myVar2=--myVar1 assigns 44 to myVar2 myVar2=myVar1-- assigns 45 to myVar2
Assignment Operators		
=	Assigns the value to the left-hand side variable.	myVar=90
+=	Adds the right-hand side operand to the left-hand side operand and assigns the result to the left-hand side operand.	myVar1=45, myVar2=10 myVar1+=myVar2 assigns 55 to myVar1
-=	Subtracts the right-hand side operand from the left-hand side operand and assigns the result to the left-hand side operand.	myVar1=45, myVar2=10 myVar1-=myVar2 assigns 35 to myVar1
*=	Multiplies the right-hand side operand with the left-hand side operand and assigns the result to the left-hand side operand.	myVar1=45, myVar2=10 myVar1*=myVar2 assigns 450 to myVar1

Table 14: Operators Available in JavaScript

Operator	Description	Example
/=	Divides the left-hand side operand by the right-hand side operand and assigns the quotient to the left-hand side operand.	myVar1=45, myVar2=10 myVar1/=myVar2 assigns 4.5 to myVar1
%=	Divides the left-hand side operand by the right-hand side operand and assigns the remainder to the left-hand side operand.	myVar1=45, myVar2=10 myVar1%=myVar2 assigns 5 to myVar1
Comparison Operators		
==	Returns true if both the operands are equal; otherwise, it returns false.	45==10 returns false
!=	Returns true if both the operands are not equal; otherwise, it returns false.	45!=10 returns true
>	Returns true if the left-hand side operand is greater than the right-hand side operand; otherwise, it returns false.	45>10 returns true
>=	Returns true if the left-hand side operand is greater than or equal to the right-hand side operand; otherwise, it returns false.	45>=10 returns true
<	Returns true if the left-hand side operand is less than the right-hand side operand; otherwise, it returns false.	45<10 returns false
<=	Returns true if the left-hand side operand is less than or equal to the right-hand side operand; otherwise, it returns false.	45<=10 returns false
Logical Operators		
&&	Returns true only if both the operands are true; otherwise, it returns false.	true&&false returns false
||	Returns true only if either of the operands is true. It returns false when both the operands are false.	true||false returns true
!	Negates the operand, that is, returns true if the operand is false and returns false if the operand is true.	!true returns false
Conditional Operator		
?:	Returns the second operand if the first operand is true. However, if the first operand is false, it returns the third operand.	myVar1=45, myVar2=10 myResult=(myVar1<myVar2)?myVar1:myVar2 returns 10

You can use operators to perform various functions, such as calculating numbers, modifying strings, and making decisions. The combination of operators and their operands form expressions. In JavaScript and other programming languages, expressions are evaluated in a particular order. Operators are evaluated in the order of high to low precedence.

Scanned by CamScanner

Exploring Control Flow Statements

Scripts written in JavaScript are executed sequentially, which means that the first statement in the script is the first to be executed and the last statement in the script is the last to be executed. This is the simplest and most straightforward way to execute scripts. However, you can change the sequence in which the script statements are executed by using control flow statements.

As the name suggests, control flow statements are special statements that control or alter the sequence in which the script statements are executed. The decision regarding the execution of a statement is based on the value of a condition, which is a JavaScript expression that evaluates to a Boolean value (true or false). When the condition evaluates to true, a particular group of statements is executed, and when the condition evaluates to false, another group of statements is executed.

In JavaScript, the control flow statements are divided into the following categories:

- **Selection statements** – Allow the execution of a group of statements from multiple groups of statements
- **Loops** – Allow repeated execution of a group of statements
- **Jump statements** – Allow the execution to skip or jump over certain statements

Let's discuss each of these categories in detail.

Understanding the Selection Statements

Suppose there is a Web page in your shopping website, which accepts a price range from the users. When the users provide a price range, the items or products that lie within the specified price range are displayed. Such functionality and logic can be incorporated by using selection statements in the script.

Selection statements use a condition to select or determine the statements that are to be executed. These statements help you in making decisions and changing the flow of execution of the statements. In JavaScript, there are three selection statements, which are as follows:

- if
- if...else
- switch

Let's discuss these selection statements in detail.

The if Statement

The if statement is one of the most basic and simplest control flow statements. You can use the if statement when you want to execute a group of one or more script statements only after a particular condition is met. The syntax of the if statement is as follows:

```
if(condition)
{
  statement1
}
```

In the preceding syntax, if is a JavaScript keyword that signifies an if statement and contains a condition which needs to be evaluated. If the condition evaluates to true, then the script statement represented by statement1, enclosed within the curly braces is executed. If the condition evaluates to false, then the statement enclosed within the curly braces is skipped and the statement immediately after the closing curly brace (}) is executed. Note that the condition for the if statement is enclosed within parentheses immediately after the if keyword.

The if...else Statement

It has been mentioned earlier that the if statement allows you to execute a set of statements only after a particular condition is true. However, if you want to execute another set of statements when the condition is false, then you can use the if...else statement. The syntax of the if...else statement is as follows:

```
if (condition)
{
statement1
}
else
{
statement2
}
```

In the preceding syntax, if and else are keywords that signify the if...else statement. The condition of the if...else statement is enclosed within parentheses. If the condition is true, then the group of statements, represented by statement1, enclosed within the first curly braces is executed. If the condition is false, then the statement1 is skipped and the group of statements, represented by statement2, of the else block is executed.

The switch Statement

A switch statement is used to select a particular group of statements to be executed among several other groups of statements. The group of statements that is to be executed is selected on the basis of a numeric or string expression. The syntax of the switch statement is as follows:

```
switch(expression)
{
case  value1:statement1
        break;
case value2:statement2
        break;
case value3:statement3
        break;
default:statement_default
        break;
}
```

In the preceding syntax, switch, case, and break are JavaScript keywords. The switch keyword indicates the switch statement. In a switch statement, the expression that is to be evaluated is specified within parentheses. This expression is checked against each of the case values specified in the case statements (indicated by the case keyword). If any of the case values match the value of the expression, the group of statements (statement1, statement2, or statement3) specified in the respective case statement is executed. If none of the case values matches the value of the expression, then the default statement, specified by the default keyword, is executed. The default statement is generally placed at the end of the switch statement; however, you can place it anywhere within the switch statement.

Every group of statements specified by the case statements has a break statement indicated by the break keyword. The break statement prevents the execution from passing on to the next case or default statement after a match is found.

Understanding Loops

Loops or iteration statements are control flow statements that allow you to execute a particular group of statements repeatedly. The number of times a group of statements is executed depends on a

Scanned by CamScanner

particular condition that is a boolean expression. If the boolean expression is true, then the group of statements is executed depending on the condition. In this way, the loop keeps on executing until the condition becomes false. When the condition becomes false, the execution of the loop halts, the current group of statements is skipped, and the statement immediately following the loop is executed.

In JavaScript, you can use any of the following three loops:

- while loop
- do...while loop
- for loop

The condition is placed either at the start or at the end of the loop, depending on the type of loop. Let's discuss each loop in detail.

The while Loop

You can use the while loop when you want to check the condition at the start of the loop. The group of statements that is to be executed is specified after the condition. This implies that if the condition is false in the first iteration itself, the group of statements in the while loop is never executed. On the other hand, the group of statements keeps on executing until the condition becomes false. Therefore, in a while loop, there can be zero or more iterations.

The syntax of the while loop is as follows:

```
while(condition)
{
statement1
}
```

In the preceding syntax, while is a JavaScript keyword. The condition for the while loop is specified in parentheses and the group of statements (statement1) is specified within the curly braces.

The do...while Loop

You have learned that in the while loop, if the condition becomes false in the very first iteration, the group of statements specified in the while loop is never executed. However, if you want the group of statements to be executed at least once, even if the condition is false, then you can use the do...while loop. This is possible because the condition of the do...while loop is placed at the end of the loop.

The syntax of the do...while loop is as follows:

```
do
{
  statement1
}
while (condition);
```

In the preceding syntax, do and while are JavaScript keywords. The group of statements, represented by statement1, is enclosed within curly braces. The condition is enclosed within parentheses immediately after the while keyword. Note that there is a semicolon after the closing parenthesis.

The for Loop

The for loop is one of the most common loops in JavaScript and other programming languages. It allows you to execute a group of statements for a predetermined number of times, that is, the number of iterations of a for loop is known beforehand. The condition of the for loop is placed at the beginning of the loop.

The syntax of the for loop is as follows:


```
for(initialization_statement; condition; updation_statement)
{
  statement1
}
```

In the preceding syntax, for is a JavaScript keyword that is immediately followed by a pair of parentheses that encloses `initialization_statement`, `condition`, and `updation_statement`, each of which is separated by a semicolon. The `initialization_statement` refers to the statement in which the loop control variable is declared or assigned an initial value. A loop control variable, as the name suggests, is a variable that is used in `condition` and `updation_statement` to control the execution of the for loop. The `updation_statement` refers to the statement that updates the value of the loop control variable.

Note that in the first iteration of the for loop, `initialization_statement` is the first statement to be executed, followed by the evaluation of the condition, which, if true, results in the execution of the group of statements (statement1) inside the for loop. After the group of statements is executed, the `updation_statement` is executed. The condition of the for loop is again evaluated for the second iteration and so on.

Understanding Jump Statement

As the name suggests, jump statements allow you to jump over or skip certain statements in a script and execute some other statement. You can use the jump statements to exit or break a loop. In JavaScript, there are two jump statements, which are discussed as follows:

- **Break statement**—It is known that the switch statement uses the break statement to prevent the execution of the subsequent case statements. The break statement also allows you to break or exit a loop. When used inside a loop, the break statement stops executing the loop and causes the loop to be immediately exited. If the loop has statements after the break statement, those statements are not executed.
- **Continue statement**—Similar to the break statement, the continue statement can also be used to stop the execution of a loop. However, the continue statement does not exit the loop; it executes the condition for the next iteration of the loop. If the loop has any statements after the continue statement, then those statements are skipped.

Exploring Popup Boxes

A popup box is a window that displays a message along with an OK button. It may also contain a Cancel button. In addition, it can prompt users to enter some text. JavaScript supports the following three types of popup boxes:

- **The alert box**—The alert box is generally used to display an alert message while executing the JavaScript code. In addition, an alert box is used to display error messages after you validate a form. It contains an OK button, which the user has to click to continue with the execution of the code.
- **The confirm box**—The confirm box is an advanced form of the alert box. It is used to display a message as well as return a true or false value. It displays a dialog box along with two buttons. The two buttons are OK and Cancel. When you click the OK button, the confirm box returns a true value, and when you click the Cancel button, the confirm box returns a false value. In other words, the confirm box enables a user to interrupt the processing of JavaScript by asking a question to the user and, as per the button clicked, continues the processing.

Scanned by CamScanner

- **The prompt box**—The prompt box is used to input a value from a user. It contains a text box and OK and Cancel buttons. The user has to click either the OK or Cancel button to continue the execution of the code after entering an input value. If the user clicks the OK button, the input value is returned. Otherwise, the box returns null value.

You have learned about JavaScript language construct. Let's learn about objects in JavaScript.

Exploring Objects in JavaScript

It has been discussed earlier that the JavaScript language is based on objects. In JavaScript, you can create an object in two ways either by creating a direct instance or by creating an object using a function template. A direct instance of an object is created by using the `new` keyword. The following code snippet shows the syntax of creating the direct instance of an object:

```
Obj=new object();
```

You can also add properties and methods to an object by using a period (.) followed by a property or method name, as shown in the following code snippet:

```
Obj.name="Robert";
Obj.rollnumber=32;
Obj.getValue();
```

In the preceding code snippet, `Obj` is the newly created object, while `name` and `rollnumber` are its properties and `getValue()` is its method.

Another way to create an instance of an object is by creating a function template of the object. After defining the function template for an object, you need to create an instance of the object by using the `new` keyword. A function template for an object is created by using the `function` keyword. You can also add properties to the function template by using the `this` keyword. The following code snippet shows an example of creating a function template:

```
function bike(speed,engine,color)
{
  this.speed=speed;
  this.engine=engine;
  this.color=color;
}
```

In the preceding code snippet, a function named `bike` is created and three parameters named `speed`, `engine`, and `color` are added to it. The values of the parameters are assigned with the properties that you want the bike object to have. The `this` keyword is used to represent the current object that is in use.

Next, you have to create the instance of the object bike by using the `new` keyword, as shown in the following code snippet:

```
var my_bike= new bike("120kph","V-6","red");
```

Now, you can access the `my_bike` instance of the `bike` object, as shown in the following code snippet:

```
var engine_type= my_bike.engine;
```

The next section discusses the properties of an object.

Properties of an Object

Property, also known as attribute, is the characteristic of an object. For example, vehicle is an object and color and speed are its properties. You can add properties to an object by using the `this` keyword followed by the dot (.) operator. Now, let's extend an object function `myobject` to contain two properties, each containing a string of text, as shown in the following code snippet:


```
function myobject(parameter){
this.property1=parameter
this.property2="Hello world"
}
```

The properties defined in the preceding code snippet can be accessed as shown in the following code snippet:

```
<SCRIPT>
var object=new myobject("Hello all")
//alerts "Hello all"
alert(object.property1)
//writes "Hello world"
document.write(object.property2)
</SCRIPT>
```

Methods of an Object

A method is a set of one or more statements that are executed by referring the name of the method. In other words, a method can be defined as an action performed by an object. In real-world scenario, let's consider bike as an object, which performs actions, such as blowing horn or accelerating speed. A complex application is divided into methods; as a result, code becomes more flexible, easy to maintain, and easy to debug. Dividing an application into discrete logical units makes it more readable. Methods also increase the reusability of code as you can execute a code for n number of times by just calling the method name. To add methods to a user-defined object, you need to perform the following steps:

- Declaring and defining a function for each method
- Associating a function with an object

Let's assume a triangle as an object and create a function `computearea()`, as shown in the following code snippet:

```
// method function
function computearea()
{
var area=this.base*this.altitude*0.5
return area
}
```

In the preceding code snippet, the `computearea()` function calculates the area of a triangle. Now, associate the function `computearea()` to the object triangle, as shown in the following code snippet:

```
<script type="text/javascript">
function triangle(b,a){
this.base=b
this.altitude=a
this.area=computearea
}
</script>
```

Now, call the method by instantiating the object, as shown in the following code snippet:

```
<script type="text/javascript">
var mytriangle=new triangle (20,10)
alert("area="+mytriangle.area())
</script>
```

Scanned by CamScanner

Exploring the Standard/Built-in JavaScript Objects

JavaScript supports some of the built-in objects such as String, RegExp, Boolean, Number, Array, Math, and Date. These objects are also called core language objects. Figure 32 shows the built-in objects hierarchy:

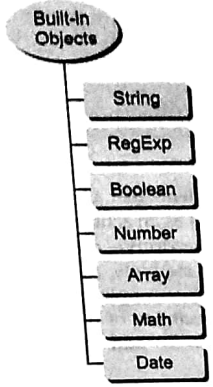

Figure 32: Showing the Built-in Objects Hierarchy

Let's discuss each of these objects in detail.

The String Object

A series of characters is known as String. In JavaScript, all strings are represented as instances of the String object. The purpose of the String object in JavaScript is to perform operations on the stored text, such as finding the length of the text or string. The String object is a wrapper class and a member of the global objects.

Table 15 lists the properties of the String object:

Table 15: Properties of the String Object	
Property	**Description**
`constructor`	Returns the function, which creates the prototype of a String object
`length`	Returns the length of a string
`prototype`	Adds properties and methods to an object

Table 16 lists the methods of the String object:

Table 16: Methods of the String Object	
Method	**Description**
`charAt()`	Returns the character in the specified index
`charCodeAt()`	Returns the Unicode equivalence of the character in the specified index
`concat()`	Joins two strings and returns the joint string
`fromCharCode()`	Converts Unicode to character

Table 16: Methods of the String Object	
`indexOf()`	Returns the position of the first occurrence of the specified character in a string
`lastIndexOf()`	Returns the position of the last occurrence of a specified value in a string
`match()`	Looks for the match between a regular expression and a string, and returns the matches
`quote()`	Writes quotes in JavaScrip
`replace()`	Searches for the match between a regular expression and a string, and replaces the matched substring with a new substring
`search()`	Searches for a match between a regular expression and a string, and returns the position of the match
`slice()`	Returns a part of a string as new string
`split()`	Splits a string into substrings
`substr()`	Extracts characters from a string beginning at the specified start index for the specified number of characters
`substring()`	Returns characters in a string between two specified indices
`toLowerCase()`	Converts a string value into a lowercase letter
`toSource()`	Returns an object literal representing the specified object
`toString()`	Returns a string representing the specified object
`toUpperCase()`	Converts a string into a uppercase letter
`valueOf()`	Returns the primitive value of a String object

A string is created by using literals. A string literal is either enclosed within single quotes (' ') or double quotes (" "). The following code snippet displays how to use string literals:

```
var string1=" ";
//string1 has null value
var string2="Petter";
//Petter is the value of string2
var string3='Don\'t do it';
//Don't do it is the value of string3
```

You can also compare two strings using the comparison operators (== or !=) given in Table 14.

Now, let's discuss the RegExp object.

The RegExp Object

A regular expression (RegExp) is an object that helps you to validate the pattern of characters. For example, you can validate whether the syntax of an e-mail address entered in a form is correct or not by providing the pattern for e-mail address in the RegExp object.

The two ways of defining regular expressions in JavaScript are as follows:

Using RegExp object's constructor function

Using literal

The following syntax shows the use of the RegExp constructor:

```
var RegularExpression = new RegExp("pattern","flag");
```


Scanned by CamScanner

In the preceding code snippet, pattern is the text of the regular expression and flag makes the regular expression search for a specific pattern.

The following code snippet is used to match all strings that look like the xxx.xxx.xxx.xxx IP address:

```
var regularexp=new RegExp("\\b\\d{1,3}\\. \\d{1,3}\\. \\d{1,3}\\.
  \\d{1,3}\\b","g");
```

The following code snippet shows the use of a literal:

```
var RegularExpression = /pattern/flag;
```

For example, to match all strings that look like IP address: xxx.xxx.xxx.xxx, the following code snippet is used:

```
var regularexp= /\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b/g;
```

The following are the three types of flags in JavaScript:

- g – Finds all the matches globally
- i – Ignores the case in searches, i.e., text written in the lower case is same as the text written in the upper case
- m – Represents multiline matching, i.e., the symbols ^(caret) and $(dollar) match the beginning and end of lines as well as the entire string

Table 17 lists the different patterns used in the RegExp object:

Table 17: Patterns Used in the RegExp Object

Pattern	Description
Pattern delimiters	
(pattern), (?:pattern)	Matches the entire contained pattern; (pattern) retains the match; (?:pattern) does not retain the match.
Lookaheads	
(?=pattern), (?!pattern)	(?=pattern) matches only if the given condition is satisfied; (?!pattern) matches only if the given condition is not satisfied.
Character Escapes	
\f, \r, \n, \t, \v, \0, [\b], \s, \S, \w, \W, \d, \D, \b, \B, \cX, \xhh, \uhhhh	\f matches form-feed; \r matches carriage return; \n matches linefeed; \t matches horizontal tab; \v matches vertical tab; \0 matches NULL character; [\b] matches backspace; \s matches whitespace; \S matches anything but a whitespace; \w matches any alphanumerical character (word characters) including underscore; \W matches any non-word characters; \d matches any digit; \D matches any non-digit; \b matches a word boundary; \B matches a non-word boundary; \cX matches a control character; example g; \cd matches control-d;

Pattern	Description
	\xhh matches the character with two characters of hexadecimal code hh; \uhhhh matches the Unicode character with four characters of hexadecimal code hhhh.
Character Sets	
[characters], [^characters]	[characters] matches all the contained characters; [^characters] matches all but the contained characters.

Table 18 lists the quantifiers of the RegExp object:

Table 18: List of Quantifiers of the RegExp Object

Quantifier	Description
n+	Specifies a string that contains at least one n
n*	Specifies a string that contains zero or more occurrences of n
n?	Specifies a string that contains zero or one occurrences of n
n{a}	Specifies a string that contains n, a number of times
n{a,b}	Specifies a string that contains n, a to b number of times
n{a,}	Specifies a string that contains n, a or more number of times
n$	Specifies a string that ends with n
^n	Specifies a string that begins with n
?=n	Specifies a string that is followed by a specific string n
?!n	Specifies a string that is not followed by a specific string n

Table 19 lists the properties of the RegExp object:

Table 19: List of Properties of the RegExp Object

Property	Description
`global`	Signifies that the g modifier is set
`ignoreCase`	Signifies that the i modifier is set
`lastIndex`	Signifies the index to start the next match
`multiline`	Signifies that the m modifier is set
`source`	Signifies the text of the RegExp pattern

Table 20 lists the methods of the RegExp Object:

Table 20: List of Methods of the RegExp Object

Method	Description
`compile()`	Compiles the RegExp object
`exec()`	Tests for a match in a string and returns a result array
`test()`	Tests for a match in a string and returns true or false

The Boolean Object

A Boolean object is a wrapper class and a member of global objects. It is used to convert the non-boolean values into boolean values and has two values, that is, true and false. The Boolean object

Scanned by CamScanner

returns false when the object is passed with values such as 0 (zero), " " (empty string), false, null, undefined, and Not a Number (NaN) values. The Boolean object can be created in the following ways:

- Using boolean literal notation
- Using the Boolean object as function
- Using the testable expressions

Using Boolean Literals

Boolean literals make use of two keywords, `true` and `false`, to automatically inherit the members of the Boolean object, as shown in the following code snippet:

```
var bool=true;
```

In the preceding code snippet, the value of the bool variable has been set to true. The following code snippet shows how to print the value of bool:

```
document.write(bool + " ");
```

After converting the value of bool to a string by using the `toString()` method, the result comes out to be true, as shown in the following code snippet:

```
document.write(bool.toString() + " ");
```

The syntax shown in the following code snippet is used to print the value of bool:

```
document.write(true + " ");
```

You can convert the true value in a string format by using the following code snippet:

```
document.write(true.toString() + " ");
```

Using Boolean Object as Function

Boolean objects are used to pass the desired initial value as an argument, as shown in the following code snippet:

```
var bool=Boolean(false);
```

The preceding code snippet returns the false value. The syntax to return the true value is shown in the following code snippet:

```
var num=10;
var bool=Boolean(num>5);
```

Using Testable Expressions

Testable expressions are used to evaluate the output in boolean values. For example, let's consider the following code snippet:

```
If (15>10)
{
…
}
```

The code shown in the preceding code snippet can also be written as follows:

```
If (true)
{…
}
```


Table 21 lists the properties of the Boolean object:

Table 21: Properties of the Boolean Object	
Property	**Description**
`constructor`	Returns the function that has created the prototype of the Boolean object
`prototype`	Allows you to add properties and methods to an object

Table 22 lists the methods of the Boolean object:

Table 22: Methods of the Boolean Object	
Method	**Description**
`toString()`	Converts the boolean value into string and returns the string
`valueOf()`	Returns the primitive value of a Boolean object

Now, let's discuss the Number object.

The Number Object

In javascript, all numbers are 64 bit (8 bytes) floating point number. These numbers ranges from 5e-324 (negative) to 1.7976931348623157e+308 (positive).

Unlike C or C++, there are no data types, such as integer or floating-point types, to define numbers in JavaScript. The syntax for creating the Number object is shown in the following code snippet:

```
var num = new Number(value);
```

Let's create the Number object named num with the value 13.12, as shown in the following code snippet:

```
var num = new Number('12.12');
```

Table 23 lists the properties of the Number object:

Table 23: Properties of the Number Object	
Property	**Description**
`constructor`	Holds the value of the constructor function that has created the object
`MAX VALUE`	Returns the maximum numerical value possible in JavaScript
`MIN VALUE`	Returns the minimum numerical value possible in JavaScript
`NEGETIVE INFINITY`	Represents the value of negative infinity
`POSITIVE INFINITY`	Represents the value of infinity
`prototype`	Adds properties and methods to the Number object

Table 24 lists the methods of the Number object:

Table 24: Methods of the Number Object	
Method	**Description**
`toExponential(x)`	Converts a number into an exponential notation
`toFixed(x)`	Rounds up a number to x digits after the decimal
`toPrecision(x)`	Rounds up a number to a length of x digits
`toString()`	Returns a string value for the Number object
`valueOf()`	Returns a primitive value for the Number object

Scanned by CamScanner

Now, let's discuss the Array object.

The Array Object

Multiple values are stored in a single variable using the Array object. In JavaScript, an array can hold different types of data types in a single array slot, which implies that an array can have a string, a number, or an object in a single slot. An Array object can be created in the following ways:

- Using the Array constructor
- Using the array literal notation

Using the Array Constructor

An empty array is created in cases where you do not know the exact number of elements to be inserted in an array. You can create an empty array by using an Array constructor, as shown in the following code snippet:

```
var myarray = new Array();
```

The following code snippet is used to create an array of any given size:

```
var myarray= new Array(size);
var myarray = new Array(20)
```

In the preceding code snippet, an array with 20 items is created.

You can also create an array with given elements, as shown in the following code snippet:

```
var myarray=new Array("element 1", "element 2", "element 3",.., "element n");
var myarray=new Array("sunday", "monday", "tuesday");
```

In the preceding code snippet, an array with elements sunday, monday, and tuesday is created.

Using the Array Literal Notation

It has been aforementioned that an array can be created by using array literal notations. Array literal notations are comma-separated list of items enclosed by square brackets.

The syntax to create an empty array by using the array literal notation is shown in the following code snippet:

```
var myarray= []
```

The following code snippet shows how to create an array when elements are given:

```
var myarray=["element 1", "element 2", "element 3", .., "element n"]
var myarray = [6, "hello", false, true];
```

In the preceding code snippet, an array containing different values, such as number 6, string hello, boolean value true, and boolean value false, is created.

Table 25 lists the properties of the Array object:

Table 25: List of Properties of the Array Object

Property	Description
constructor	Holds the value of the constructor function that has created the object
length	Holds the number of elements in an array
prototype	Adds properties and methods to the Array object

Table 26 lists the methods of the Array object:

Table 26: List of Methods of the Array Object

Method	Description
concat()	Joins two or more arrays and returns the joined array
join()	Joins all the elements of an array into a string
pop()	Removes the last element of an array and returns that element
push()	Adds new element as the last element and returns the length of the new array
reverse()	Reverses the order of list of element in an array
shift()	Removes the first element of an array and returns that element
slice()	Selects a part of an array and returns that part as a new array
sort()	Sorts the elements of an array
splice()	Adds or removes the elements of an array
toString()	Converts an array into a string and returns the string
unshift()	Adds new elements to an array and returns the new length
valueOf()	Returns the primitive value of an Array object

Now, let's discuss the Math object.

The Math Object

The Math object is used to perform simple and complex arithmetic operations. JavaScript Math object is used to perform mathematical tasks.

Table 27 lists the properties of the Math object:

Table 27: Properties of the Math Object

Property	Description
E	Holds Euler's number whose approximate value is 2.718
LN2	Holds natural logarithm of 2 whose approximate value is 0.693
LN10	Holds natural logarithm of 10 whose approximate value is 2.302
LOG2E	Holds the base-2 logarithm of E whose approximate value is 1.442
LOG10E	Holds the base-10 logarithm of E whose approximate value is 0.434
PI	Holds the numerical value of PI whose approximate value is 3.142
SQRT1_2	Holds the square root of ½ whose approximate value is 0.707
SQRT2	Holds the square root of 2 whose approximate value is 1.414

Table 28 lists the methods of the Math object:

Table 28: Methods of the Math Object

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns arccosine (in radian) of x
asin(x)	Returns arcsine (in radian) of x
atan(x)	Returns the arctangent of x with numeric value between –PI/2 and PI/2 radians
atan2(y,x)	Returns the arctangent of the quotient on dividing y and x

Scanned by CamScanner

Table 28: Methods of the Math Object	
Method	**Description**
ceil(x)	Rounds up x to the nearest larger integer
cos(x)	Returns cosine value of x
exp(x)	Returns the value of ex
floor(x)	Rounds up x to the nearest smaller integer
log(x)	Returns the natural logarithmic value of x
max(x,y,z,...,n)	Returns the highest number from the given list
min(x,y,z,...,n)	Returns the lowest number from the given list
pow(x,y)	Returns x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds up x to the nearest integer
sin(x)	Returns the sine value of x
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent value of x

The Date Object

The Date object is used to display a date on a Web page or a timestamp in numerical or mathematical calculations. In JavaScript, you can either define the Date object by providing the date of your choice or let JavaScript define the Date object based on the visitor's system clock. The Date object stores dates as milliseconds that have passed since midnight on January 1, 1970 Coordinated Universal Time (UTC). The Date object can represent dates 285,616 years before or after January 1, 1970. It can be created with the help of the constructor function. You can create the Date object as shown in the code snippet:

```
var date1 = new Date();
      OR
var date1 = new Date(milliseconds);
      OR
var date1= new Date (yyyy, mm, dd [, hr, min, sec, millisec]);
      OR
var date1= new Date("mm dd, yyyy");
      OR
var date1= new Date("mm dd, yyyy hr:min:sec");
```

Table 29 lists the properties of the Date object:

Table 29: Properties of the Date Object	
Property	**Description**
constructor	Holds the value of the constructor function that has created the object
prototype	Adds properties and methods to the Date object

Table 30 lists the methods of the Date object:

Table 30: Methods of the Date Object	
Method	**Description**
getDate()	Returns the day of the month that ranges from 1 to 31
getDay()	Returns the numerical equivalence of the day of a week that ranges from 0 to 6, where 0 stands for Monday and 6 stands for Sunday
getFullYear()	Returns the numerical equivalence of the year in four digits
getHours()	Returns the hours that ranges from 0 to 23
getMilliseconds()	Returns the milliseconds that ranges from 0 to 999
getMinutes()	Returns minutes that ranges from 0 to 59
getMonth()	Returns the numerical equivalence of month that ranges from 0 to 11
getSeconds()	Returns the seconds that ranges from 0 to 59
getTime()	Returns the number of milliseconds since midnight Jan 1, 1970
getTimezoneOffset()	Returns the difference of time (in minutes) between the Coordinated Universal Time (UTC) and the local time
getUTCDate()	Returns the day of the month (from 1 to 31) as per the universal time
getUTCDay()	Returns the numerical equivalence of the day of the week (from 0 to 6) as per the universal time
getUTCFullYear()	Returns the year in four digits as per the universal time
getUTCHours()	Returns the hours (that ranges from 0 to 23) as per the universal time
getUTCMilliseconds()	Returns the milliseconds (that ranges from 0 to 999) as per the universal time
getUTCMinutes()	Returns the minutes (that ranges from 0 to 59) as per the universal time
getUTCMonth()	Returns the numerical equivalence of month (that ranges from 1 to 31) as per the universal time
getUTCSeconds()	Returns the seconds (that ranges from 0 to 59) as per the universal time
Parse()	Parses a date string and returns the number of millisecond since the midnight of January 1, 1970
setDate()	Sets the day of a month that ranges from 1 to 31
setFullYear()	Sets the year in four digits
setHours()	Sets the hours that ranges from 0 to 23
setMilliseconds()	Sets the milliseconds that range from 0 to 999
setMinutes()	Sets the minutes that range from 0 to 59
setMonth()	Sets the numerical equivalence of month that ranges from 0 to 11
setSeconds()	Sets the seconds that range from 0 to 59

Scanned by CamScanner

Table 30: Methods of the Date Object

Method	Description
`setTime()`	Sets a date and time by adding or subtracting specified milliseconds to/from midnight January 1, 1970
`setUTCDate()`	Sets the day of the month that ranges from 1 to 31 as per the universal time
`setUTCFullYear()`	Sets the year in four digits as per the universal time
`setUTCHours()`	Sets the hours that range from 0 to 23 as per the universal time
`setUTCMilliseconds()`	Sets the milliseconds that range from 0 to 999 as per the universal time
`setUTCMinutes()`	Sets the minutes that range from 0 to 59 as per the universal time
`setUTCMonth()`	Sets the numerical equivalence of month that ranges from 0 to 11 as per the universal time
`setUTCSeconds()`	Sets the seconds that range from 0 to 59 as per the universal time
`toDateString()`	Converts date into a string
`toLocalDateString()`	Converts date into string as per local conventions
`toLocalTimeString()`	Converts time into string as per local convention
`toLocalString()`	Converts the Date object into a string as per local convention
`toString()`	Converts the Date object into a string
`toTimeString()`	Converts time into a string
`toUTCString()`	Converts the Date object into string as per the universal time
`UTC()`	Holds the number of millisecond since the midnight of January 1, 1970, as per the universal time
`valueOf()`	Returns the primitive value of the Date object

In JavaScript, there are two kinds of objects—language objects and browser objects. Let's learn about JavaScript browser objects.

Exploring Browser Objects in JavaScript

Browser objects are automatically created by a browser at the time of loading a Web page. When an HTML document is opened in a browser window, the browser interprets the document as a collection of hierarchical objects and accordingly displays the data contained in these objects. The browser parses the document and creates a collection of objects, which defines the document and its details. Parsing is a process of analysing a document for correct syntax and building a data structure on the basis of the specified syntax.

Browser objects are of various types, including window, navigator, document, screen, history, and location. The window object is used to open a window in a browser. The navigator object acts as a storehouse of all the data and information about the browser. The document object is an HTML document that is loaded into a browser. The history object stores URLs visited by a user in the browser. The screen object stores the information of the visitor's screen. Finally, the location object stores the information of the currently visited URL. The collection of browser objects is also known as the Browser Object Model (BOM).

Let's start the discussion with the window object.

Understanding the Window Object

The Window object is used to open a window in a browser to display the Web page. It is a global object, which means that it provides global access to the associated variables and functions. Whenever you open a new window, a Window object is created.

Figure 33 shows the Window object in the hierarchy of browser objects:

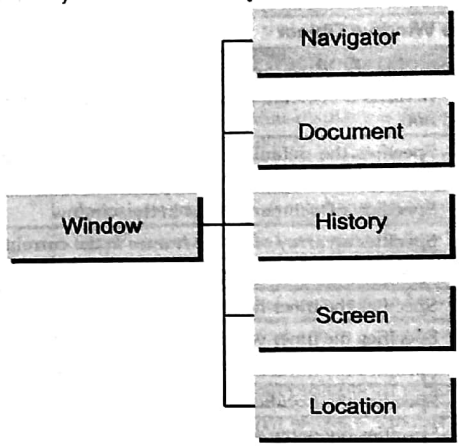

Figure 33: Showing the Window Object in the Hierarchy of Browser Objects

The properties of the Window object are used to retrieve the information about the window that is currently open on the Web browser, whereas the methods are used to perform specific tasks, such as opening, maximizing, or minimizing the window. The Window object is the top level or parent object in the hierarchy of browser objects and is automatically defined by the browser, as shown in the following code snippet:

```
document.write ("JavaScript")
```

In the preceding code snippet, JavaScript specifies the current Window object `document.write()` and executes it.

The Window object in JavaScript has the following features:

- Window object collection
- Window object properties
- Window object methods

Let's now discuss each of these features one by one.

Window Object Collection

The Window object collection is a set of all the Window objects available in an HTML document. If an HTML document contains the `FRAME` or `IFRAME` element, then its collection should also contain Window objects for these elements. The Window object contains a collection object called frame, which is used to retrieve a collection of Window objects defined in the specified document.

Now, let's discuss the window object properties.

Scanned by CamScanner

Window Object Properties

Properties refer to the variables created inside the Window object. In JavaScript, the available data is attached to the Window object as properties. The syntax used to access Window object properties is as follows:

```
window.propertyname
```

In the preceding syntax, propertyname is the name of the property.

Table 31 shows the different properties available in the window object:

Table 31: Properties of the Window Object

Property	Description
`closed`	Returns a boolean value that specifies whether a window has been closed or not
`defaultStatus`	Specifies the default message that has to be appeared in the status bar of a window
`document`	Specifies a Document object in the window
`frames`	Specifies an array of all the frames in the current window
`history`	Specifies the History object for the window
`innerHeight`	Specifies the inner height of a window's content area
`innerWidth`	Specifies the inner width of a window's content area
`length`	Specifies the number of frames contained in a window
`location`	Specifies the Location object for the window
`name`	Specifies the name of a window
`outerHeight`	Specifies the height of the outside boundary of a window in pixels
`outerWidth`	Specifies the width of the outside boundary of a window in pixels
`parent`	Returns the parent frame or window of the current window
`screenLeft`	Specifies the x coordinate of the window relative to a user's monitor screen
`screenTop`	Specifies the y coordinate of the window relative to a user's monitor screen
`screenX`	Specifies the x coordinate of the window relative to a user's monitor screen
`screenY`	Specifies the y coordinate of the window relative to a user's monitor screen
`self`	Returns the reference of the current active frame or window
`status`	Specifies the message that is displayed in the status bar of a window when an activity is performed on the window
`top`	Specifies the reference of the topmost browser window

Now, let's discuss the window object methods.

Window Object Methods

Methods refer to the functions created inside the Window object. The methods associated with the Window object specify actions, such as how it displays a message or gets input from the user. The syntax used to access the Window object methods is shown in the following code snippet:

```
window.alert("Demo of an alert box");
```

In the preceding code snippet, the `alert()` method displays an alert message box.

Table 32 shows the methods of the Window object:

Table 32: Methods of the Window Object

Method	Description
`alert()`	Specifies a method that displays an alert box with a message and the OK button
`blur()`	Specifies a method that removes the focus from the current window
`clearInterval()`	Specifies a method that clears the timer, which is set by using the `setInterval()` method
`clearTimeout()`	Specifies a method that clears the timer, which is set by using the `setTimeout()` method
`close()`	Specifies a method that closes the current window
`confirm()`	Specifies a method that displays a dialog box with a message and two buttons, that is, OK and `Cancel`
`createPopup()`	Specifies a method that creates a pop-up window
`focus()`	Specifies a method that sets focus on the current window
`moveBy()`	Specifies a method that moves a window relative to its current position
`moveTo()`	Specifies a method that moves a window to an specified position
`open()`	Specifies a method that opens a new browser window
`print()`	Specifies a method that sends a print command to print the content of the current window
`prompt()`	Specifies a method that prompts for input
`resizeBy()`	Specifies a method that resizes a window with the specified pixels
`resizeTo()`	Specifies a method that resizes a window with the specified width and height
`scrollBy()`	Specifies a method that scrolls the content of a window by the specified number of pixels
`scrollTo()`	Specifies a method that scrolls the content of a window up to the specified coordinates
`setInterval()`	Specifies a method that evaluates an expression at specified time intervals (in milliseconds)
`setTimeout()`	Specifies a method that evaluates an expression after a specified number of milliseconds

Understanding the Navigator Object

The navigator object is used to display information about the version and type of the browser. In JavaScript, the navigator object is associated with some properties and methods, which help you navigate from one element to another. The navigator object has the following features:

- Collections
- Properties
- Methods

Let's start with the discussion of navigator object collections.

Scanned by CamScanner

Navigator Object Collections

Javascript run-time system creates the navigator object automatically. This object contains all the information about the client browser. The navigator object provides the following collection objects:

- **plugins[]**—Returns a reference to all the embedded objects in the document
- **mimeTypes**—Returns a collection of Multipart Internet Mail Extension (MIME) types supported by the client browser

Now, let's discuss navigator object properties.

Navigator Object Properties

The properties of the navigator object are the variables created inside the navigator object. The syntax used for accessing the navigator object property is shown in the following code snippet:

```
navigator.propertyname
```

In the preceding syntax, property name is the name of the property.

Table 33 shows the properties available in the navigator object:

Table 33: Properties of the Navigator Object

Properties	Description
appcodename	Specifies the code name of the browser
appname	Specifies the name of the browser
appversion	Specifies the version of the browser being used
cookieenabled	Specifies whether the cookies are enabled or not in the browser
platform	Contains a string indicating the machine type for which the browser was compiled
useragent	Contains a string representing the value of the user-agent header sent by the client to the server in the http protocol

Navigator Object Methods

The methods in the navigator object refer to the functions created inside the navigator object. These methods also help to check whether or not a browser supports Java code.

Table 34 shows the methods available in the navigator object:

Table 34: Methods of Navigator Object

Methods	Description
javaEnabled()	Tests whether or not Java is enabled.
taintEnabled()	Determines whether or not data tainting is enabled. Tainting helps in preventing the transfer of secure and private information such as directory structures or user session history.

Understanding the History Object

The history object is a predefined JavaScript object that consists of an array of URLs, which are visited by a user in the browser. This object is returned by using the history property of a window object. The history object has the following features:

- *Properties*
- *Methods*

Let's start with the discussion of history object properties.

History Object Properties

The properties of the history object are the variables created inside the history object. The syntax used for accessing the history object property is as follows:

```
history.propertyname
```

In the preceding syntax, property name is the name of the property.

Table 35 shows the properties available in the history object:

Table 35: Properties of the History Object

Properties	Description
length	Specifies the number of elements contained in the object.
current	Specifies the URL of the current entry in the object.
next	Specifies the URL of the next element in the History list. It is equivalent to the URL which is visited by using the Forward button in the Go menu.
previous	Specifies the URL of the previous element in the History list. It is equivalent to the URL which is visited by using the Back button in the Go menu.

History Object Methods

The methods in the history object refer to the functions created inside the history object. Methods also specify the actions related to the URLs visited in the browser.

Table 36 shows the methods available in the history object:

Table 36: Methods of the History Object

Methods	Description
back()	Specifies a method that loads the previous URL from the history list
forward()	Specifies a method that loads the next URL from the history list
go()	Specifies a method that loads a specific URL from the history list

Understanding the Screen Object

The screen object is used to determine the properties of the display screen of a browser. In other words, the screen object provides the information about the dimensions of a display screen, such as its height, width, and color bits. You can use the screen object with or without the window object, as shown in the following syntaxes:

With window object:

```
window.screen
```

Without window object:

```
screen
```

Using the screen object along with the window object represents the browser window and its screen object. However, using the screen object, without using the window object, automatically detects the browser window, which is created by the JavaScript runtime engine.

The properties of the screen object are variables created inside the screen object. The syntax used for accessing the screen object property is as follows:

```
screen .propertyname
```


Scanned by CamScanner

In the preceding syntax, property name is the name of the property.

Table 37 shows the properties available in the screen object:

Table 37: Properties of the Screen Object

Property	Description
availHeight	Specifies the height of the screen, excluding the Windows Taskbar
availWidth	Specifies the width of the screen, excluding the Windows Taskbar
colorDepth	Specifies the depth of the color palette, in bits, to display images
height	Specifies the total height of the screen
pixelDepth	Specifies the color resolution, in bits per pixel, of the screen
width	Specifies the total width of the screen

NOTE

The screen object provides the watch() and unwatch() methods, which are inherited from the Object object. Object is the primitive object for all the objects that are derived from it. The watch() method adds a watchpoint to a property of the screen object. The unwatch() method removes a watchpoint from the screen object.

Understanding the Location Object

In JavaScript, location is a child object of the window object. It is used to store information of the current URL of the window object. The location object allows you to automatically navigate to another Web page. For instance, by using the following code snippet, you can navigate to the home page of Google:

```
<script type="text/javascript">
window.location="http://www.google.com"
</script>
```

The location object has the following features:

- Properties
- Methods

Location Object Properties

The properties of the location object are the variables created inside the location object. The syntax used for accessing the location object property is as follows:

```
location.propertyname
```

In the preceding syntax, property name is the name of the property.

The following string representation shows the different parts of a URL:

```
<protocol>//<host>[:<port>]/<pathname>[<hash>][<search>]
```

An example of a URL is as follows:

```
http://www.javascriptblackbooks.com:80/order.cgi?batch=1#intro
```

The properties of the location object show different parts of a URL. Table 38 shows the properties available in the location object:

Table 38: Properties of the Location Object

Property	Description
href	Represents a string specifying the entire URL; for instance, `http://www.javascriptblackbooks.com:80/order.cgi?batch=1#intro`

Table 38: Properties of the Location Object

Property	Description
protocol	Represents a string at the beginning of a URL up to the first colon (:), which specifies the method of access to the URL; for instance, `http:` or `https:`
host	Represents a string consisting of the hostname and port strings; for instance, `www.javascriptblackbooks.com:80`
hostname	Represents the server name, subdomain, and domain name (or IP address) of a URL; for instance, `www. javascriptblackbooks.com`
port	Represents a string specifying the communications port that the server uses; for instance, `80`
pathname	Represents a string portion of a URL specifying how a particular resource can be accessed; for instance, `order.cgi`
search	Represents a string beginning with a question mark that specifies any query information in an HTTP URL; for instance, `batch=1`
hash	Represents a string beginning with a hash (#), which specifies an anchor name in an HTTP URL; for instance, `#intro`

Let's now explore the methods of the location object.

Location Object Methods

The methods in the location object refer to the functions created inside the location object. Table 39 shows the methods available in the location object:

Table 39: Methods of the Location Object

Method	Description
assign()	Loads a new document in the browser.
reload()	Reloads the current document that is contained in the `location.href` property.
replace()	Replaces the current document with the specified new one. In addition, you cannot navigate back to the previous document using the browser's `Back` button.

The following section discusses DOM.

Document Object Model

DOM is designed to work independent of any programming language; it is often used with JavaScript to design dynamic and interactive HTML pages. DOM is a platform and language independent interface. It permits programs and scripts to access and update the content, structure and style of HTML or XML document dynamically.. Every element in an HTML document represents a DOM node and is related to each other through the parent-child relationship.

The properties, methods and events of DOM that are used in manipulating and creating Web pages are organized into objects and these objects are known as DOM objects. Depending upon the type of objects that they contain, DOM is divided into three parts— Core, XML, and HTML. The Core DOM includes the objects that are present in both XML and HTML documents, whereas the XML and HTML DOMs contain only XML and HTML objects, respectively.

Scanned by CamScanner

Understanding DOM Nodes

Every element in an HTML page represents a DOM node. These elements are related to each other through the parent-child relationship. All the nodes or elements used within an HTML document compose a DOM tree. The DOM tree describes the relationship among the elements. An element inside another element is known as the child element or child node, and the element that contains the element within it is known as the parent element or parent node. A node has only one parent node but it may have multiple children. The nodes having same parent nodes are known as siblings. Let's consider the following code snippet to understand this concept:

```
<P>Some Text</P>
```

In the preceding code snippet, there are two nodes: P and a text node. The combination of both these nodes represents a paragraph. The text node is inside the P node; therefore, the text node is the child node and the P node is the parent node. In the preceding code snippet, the P node has only one child. Now, consider the following code snippet:

```
<P>This is <B>My DOM</B></P>
```

In the preceding code snippet, you have created one parent node (P) and two child nodes (text and B node). Further, B node also contains a child node (text).

Figure 34 shows how the elements of the preceding code snippet can be represented in the form of a tree:

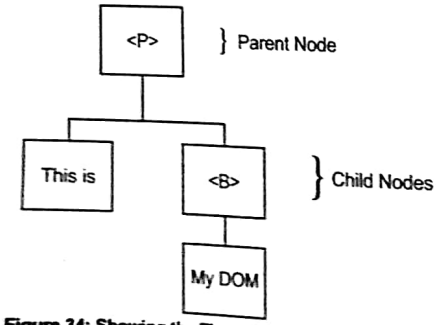

Figure 34: Showing the Elements in a Tree Presentation

In the representation shown in Figure 34, you can easily observe that the P node has two child nodes and B node has one child node. As discussed, a document can be viewed as a node tree. In the node tree view, a document is a collection of nodes. The nodes symbolize the branches and leaves on the document tree. There are several types of nodes, but the three main types are – Element nodes, Text nodes, and Attribute nodes.

Figure 35 shows a document in the form of a node tree:

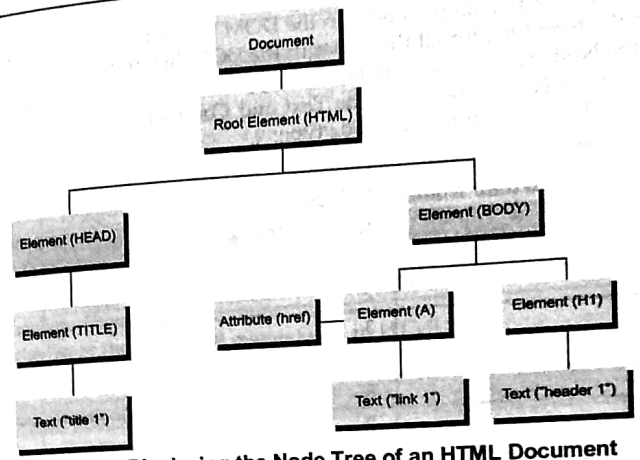

Figure 35: Displaying the Node Tree of an HTML Document

The key components of the node structure shown in Figure 35 are explained as follows:

- **Element nodes**—Represent the basic building blocks of documents known as elements. These elements can contain other elements, such as HTML, HEAD, BODY, A, and H1.
- **Text node**—Represents the content contained in the Element nodes, such as title 1, link 1, and header 1.
- **Attribute nodes**—Provide more information about elements. Attribute nodes are always contained in the Element nodes. For example, href is an Attribute node, which is contained within the BODY element. For example, the following code presents href as attribute node:

```
<BODY> <A href= "newfile.html">click here to go to the new file.</A> </BODY>
```

Each of these nodes has some properties that contain specific information about the node. The properties of a node are as follows:

- **nodeName**—Contains the name of the node. The nodeName property is used for the **Element node, Attribute node, and Text node** to represent the name of element, attribute, and text, respectively.
- **nodeValue**—Represents the value of the node.
- **nodeType**—Returns the type of the node.

The DOM nodes are also placed according to an inheritance hierarchy, which is shown in Figure 36:

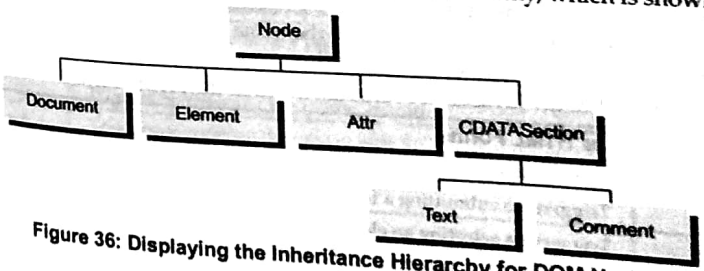

Figure 36: Displaying the Inheritance Hierarchy for DOM Nodes

Scanned by CamScanner

It can be seen in Figure 36 that all the nodes in the DOM hierarchy inherit the Node type. Therefore, the Node type is the base type for all the nodes in the DOM hierarchy. All nodes can use properties and methods of the Node type. Similar to the Node type, the CDATASection type is the base type for the Text and Comment types. Therefore, the Text and Comment types can use the properties and methods of the CDATASection type and Node type.

Table 40 lists the nodes and their numeric values:

Table 40: DOM Nodes and their Numeric Values

Node Type	Numeric Value
Element	1
Attr	2
Text	3
CDATASection	4
Comment	8
Document	9

Note that the Node type does not have numeric value because it is the base type and can only provide methods for child nodes. The following are some of the rules defined by DOM for the child nodes:

- Ensure that a Document node has only one Element node as a child node
- Ensure that an Element node has the Element, Text, and Comment nodes as child nodes
- Ensure that an Attr node is attached to the Element node
- Ensure that an Attr node has a Text node as a child node
- Ensure that the Text and Comment nodes do not have child nodes

You have learned about DOM, let's learn about events in JavaScript.

Exploring Events

Events refer to actions that are detected by a programming language when you perform a particular task. For example, the `onclick` event is detected by the programming language when you click the mouse button. Events are generally used in combination with functions, implying that when an event occurs, it calls a specified function. An event commonly occurs when a user clicks the mouse button, Web page is loaded, or form field is changed. Events are handled by a special function known as event handler, which handles a particular event when the event is triggered. The following code snippet shows how to create an event handler:

```
onEvent = "Code to handle the event"
```

In the preceding code snippet, onEvent refers to the name of an event followed by an equal sign and the code that executes when the event is triggered.

Table 41 shows the list of the JavaScript events used within the HTML forms:

Table 41: Events of the HTML Forms

Event	Description	New in HTML5
onsubmit	Triggers on submitting a form	No
onselect	Triggers on selecting an element	No
oninvalid	Triggers when an element is invalid	Yes

Table 41: Events of the HTML Forms

Event	Description	New in HTML5
oninput	Triggers when input is provided for an element	Yes
onforminput	Triggers when input is provided on a form	Yes
onformchange	Triggers when a form changes	Yes
onfocus	Triggers when a window gets focus	No
oncontextmenu	Triggers when the context menu is used	Yes
onchange	Triggers when an element changes	No
onblur	Triggers when a window loses focus	No

Table 42 shows the list of the keyboard events used in HTML:

Table 42: Keyboard Events Used in HTML

Event	Description	New in HTML5
onkeydown	Triggers on pressing a key	No
onkeypress	Triggers when a key is pressed	No
onkeyup	Triggers on releasing a key	No

Table 43 shows the list of the JavaScript mouse events used in HTML:

Table 43: Mouse Events of the HTML Forms

Event	Description	New in HTML5
onclick	Triggers on clicking the mouse button	No
ondblclick	Triggers on double clicking the mouse button	No
ondrag	Triggers on dragging an element	Yes
ondragend	Triggers on ending the drag operation	Yes
ondragenter	Triggers on dropping the dragged element on a valid target	Yes
ondragleave	Triggers on leaving the valid target while dragging an element	Yes
ondragover	Triggers on dragging the element over the valid drop target	Yes
ondragstart	Triggers on starting the drag operation	Yes
ondrop	Triggers on dropping the dragged element	Yes
onmousedown	Triggers on pressing the mouse button	No
onmousemove	Triggers on moving the mouse pointer	No
onmouseout	Triggers when the mouse pointer leaves an element	No
onmouseover	Triggers when the mouse pointer moves over an element	No
onmouseup	Triggers on releasing the mouse button	No
onmousewheel	Triggers on rotating the mouse wheel	Yes
onscroll	Triggers on scrolling the scroll bar of an element	Yes

Validating an HTML Form Using JavaScript

Form validation is the process of verifying whether a form has been filled properly before getting processed. When a user enters data into a form field, it is possible that the user can make a mistake

Scanned by CamScanner

or enter the incorrect data in the fields. You can check these mistakes or incorrect data input by validating those fields. For example, if your form has text boxes for the user id, password, and email address and you want the user id to start with a number, the password must be minimum of six characters and maximum of 20 characters. In addition, the email addresses must be entered in the correct format, then the best option is to write code to validate these fields. You can validate a form by using the following validation methods:

- **Server-side validation**—Uses Common Gateway Interface (CGI) scripts, Servlet, Java Server Pages (JSP), and Active Server Pages (ASP) to validate a form
- **Client-side validation**—Uses JavaScript or VBScript to validate a form

The server-side validation is more secure, but it is more complex to code and requires a server connection to validate the form. On the other hand, the client-side validation is easier and quicker because it does not require a server connection to validate the form, so the users find out instantly if they have made mistakes or entered incorrect data before submitting the form. The advantages of the client-side validation are as follows:

- Validates form elements before the form data is submitted to the server
- Does not require a connection with a server
- Reduces the load on the server
- Saves time

In JavaScript, you can restrict a user from filling up the form fields. If the user does not fill up the fields, then JavaScript generates an error message before submitting the form. This whole process is known as the required field validation. To validate required fields, you have to write the code that checks the value of these fields; if the fields have values, then the form is submitted. However, if the fields are blank or have null value, then an error message should be displayed to fill up the fields. Let's create a Web page named ValReqFields.html which shows an example of required field validation.

Listing 29 shows the code of ValReqFields.html:

Listing 29: Creating the ValReqFields.html File

```
<!DOCTYPE HTML>
<HTML>
<HEAD><TITLE>Validating required fields</TITLE>
<SCRIPT type="text/javascript">
function validateReqFields(thisform)
{
  var userName=thisform.uname.value;
  var password=thisform.pw.value;
  if(userName==null || userName=="")
  {
        alert("Enter the User name");
        thisform.uname.focus();
        return false;
  }
  if(password==null || password=="")
  {
        alert("Enter the password");
        thisform.pw.focus();
        return false;
  }
  return true;
}
</SCRIPT>
</HEAD>
<BODY>
<FORM action="Submit.html" onsubmit="return validateReqFields(this)"
 method="Post">
User Name:<input type= "text" name="uname"/>*<br/>
Password: <input type= "password" name="pw"/>*<br/>
<INPUT type="submit" value="submit"/>
</FORM>
</BODY>
</HTML>
```


In Listing 29, if the value of username is null, then the code alerts the user with the message Enter the user name, and the `focus()` function prompts the cursor in the `uname` text box and returns false. The same process happens with the password variable. If the form fields have values, the `ValidateReqFields ()` function returns true and the form action event triggers that call the Submit.html file. Listing 30 shows the code to create a Web page named Submit.html:

Listing 30: Creating the Submit.html File

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>submit form</TITLE>
</HEAD>
<BODY>
<H2>You have filled all the required fields </H2>
</BODY>
</HTML>
```

In Listing 30, we have created a Web page named Submit.html that displays a message when a user fills up all the required fields and clicks the submit button in the ValReqFields.html Web page. Figure 37 shows the output of Listing 30:

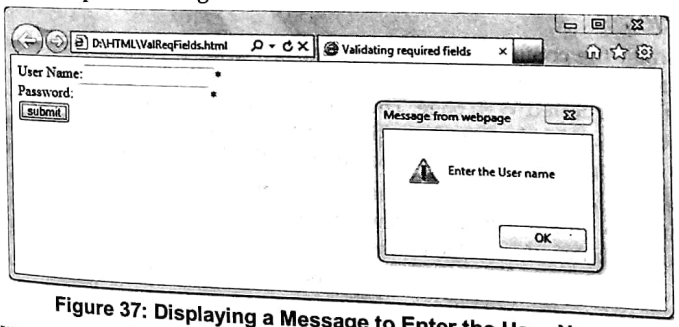

Figure 37: Displaying a Message to Enter the User Name

Figure 37 shows a message when a user attempts to submit a form without entering any value in the `User Name` field. Figure 38 shows a message when the user enters a value in the `User Name` field, but does not enter a value in the `Password` field:

Scanned by CamScanner

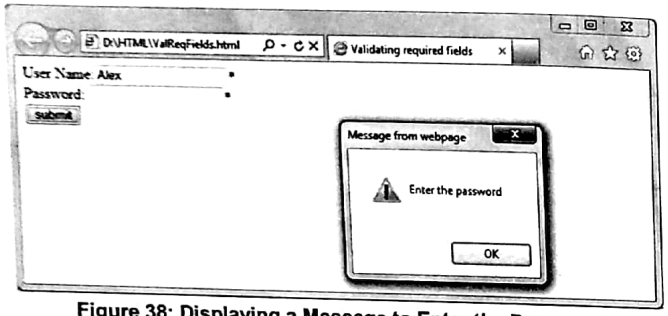

Figure 38: Displaying a Message to Enter the Password

When the user enters a value in both the fields and submits the form, the form calls the Submit.html file, as shown in Figure 39:

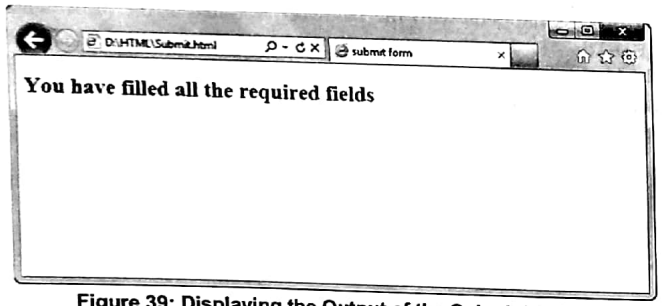

Figure 39: Displaying the Output of the Submit.html File

Figure 39 shows a message when the user fills all the required fields. After validating required fields, let's learn about cookies.

Exploring Cookies

A cookie is a small file containing information, which a server embeds on a user's computer. Each time the same computer requests for a Web page from a server, the server refers to the created cookie for displaying the requested Web page. The size of a cookie is dependent on a browser. In general, it should not exceed 1K (1024 bytes). A cookie is generally used to store the user name and password information on a computer so that you need not enter this information each time you visit a website.

A cookie has several attributes, which contain vital information about the cookie, such as its name, the domain name to which it belongs, and the address of the valid path within a domain. Some commonly used attributes of a cookie that we can use with JavaScript are as follows:

- **The name attribute**—Represents a variable name and the corresponding value to be stored in a cookie.
- **The expires** attribute—Specifies the time when a cookie is deleted. It is expressed as a Unix timestamp.
- **The domain attribute**—Represents a domain name (partial or complete) to which a cookie is sent. For example, if the value for the Valid domain attribute is www.gmail.com, a client sends the cookie information to the Web browser every time the client visits the www.gmail.com website.
- **The path attribute**—Identifies sites among various paths in the same domain. Setting this attribute to the server root (/) allows the entire domain to access the information stored in a cookie.
- **The secure attribute**—Restricts a browser from sending cookie information over unsecured connections. It allows the cookie to be sent over any type of HTTP connection. The default value of the Security Flag attribute is 0. Setting the value to 1 allows the cookie to be sent over a secure HTTP connection.

Let's create a Web page named cookie.html to learn how to create cookies. Listing 31 shows the code for the cookie.html file:

Listing 31: Showing the Code for the cookie.html File

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
        <SCRIPT type="text/javascript">

function getCookie(cname)
{
var cvalue = document.cookie;
var cstart = cvalue.indexOf(" " + cname + "=");
if (cstart == -1)
  {
  cstart = cvalue.indexOf(cname + "=");
  }
if (cstart == -1)
  {
  cvalue = null;
  }
else
  {
  cstart = cvalue.indexOf("=", cstart) + 1;
  var cend = cvalue.indexOf(";", cstart);
  if (cend == -1)
  {
cend = cvalue.length;
}
cvalue = unescape(cvalue.substring(cstart,cend));
}
return cvalue;
}

        function setCookie(cname,value,expiredays)
                {
                        var exdate=new Date();
                        exdate.setDate(exdate.getDate()+expiredays);
                        document.cookie=cname+ "="
                        +escape(value)+((expiredays==null) ?
                        "" : ";expires="+exdate.toUTCString());
                }
                function checkCookie()
                {
                        mycookie=getCookie('mycookie');
```


Scanned by CamScanner

```
                if (mycookie!=null && mycookie!="")
                {
                alert('Welcome again '+mycookie+'! \n The cookies has
                already
                been created.');
                }
                else
                {
                        mycookie=prompt('Please enter your name:',"");
                        if (mycookie!=null && mycookie!="")
                        {
                                setCookie('mycookie',mycookie,30);
                        }
                }
        }
    </SCRIPT>
</HEAD>
<BODY>
    <H1>Creating and Reading Cookies</H1>
    <P>Click on the button to create the cookie: </P>
    <INPUT type="button" onclick="checkCookie()" value="Create Cookie">
</BODY>
</HTML>
```

In Listing 31, the `document.cookie` statement is used in the `setCookie()` method to create a cookie for a Web page. Next, we have used the `getCookie()` method to retrieve the value of the existing cookie. The output of Listing 31 is shown in Figure 40:

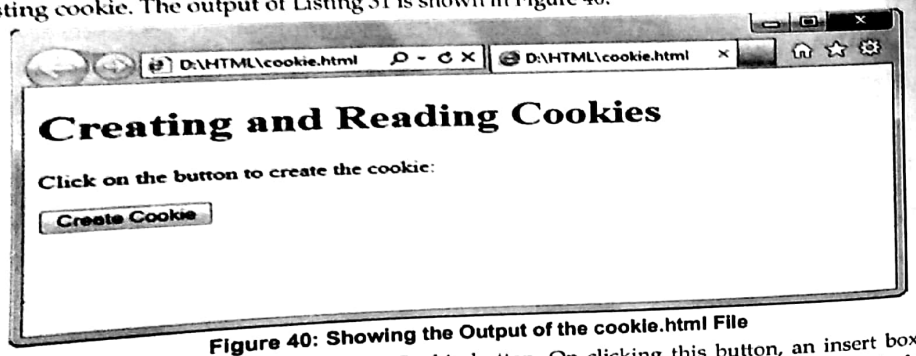

Figure 40: Showing the Output of the cookie.html File

In Figure 40, you can see the Create Cookie button. On clicking this button, an insert box opens in which you need to enter the value that you want to save in the cookie. The insert box is shown in Figure 41:

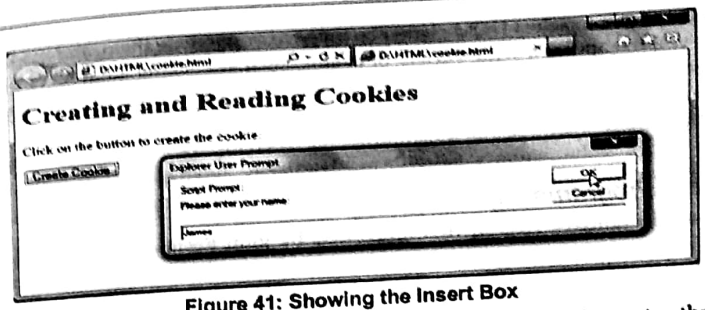

Figure 41: Showing the Insert Box

Enter you name in the text box and click the OK button (Figure 41). This creates the cookie of your Web page. Now, when you click the Create Cookie button again, an alert box appears with a message that the cookie has already been created, as shown in Figure 42:

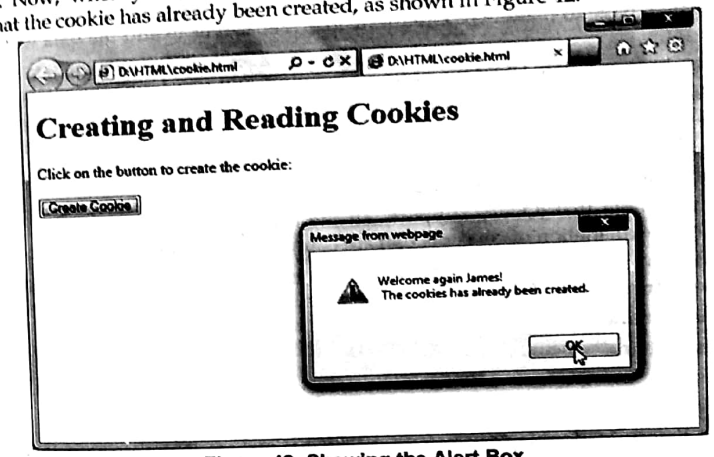

Figure 42: Showing the Alert Box

Let's learn about the basics of JQuery programming.

Introducing JQuery

jQuery is a concise JavaScript library that enables you to quickly and easily create Web pages and Web applications. It is a lightweight JavaScript and Asynchronous JavaScript and XML (AJAX) library, which supports multiple browsers and emphasizes on interaction between JavaScript and HTML. It enables you to create Web applications that can contain animations, communicate with server to send requests or get response, and handle events. jQuery is itself written in the JavaScript language and available in the form of the.js file. One of the important features of jQuery is that it supports and works on many different browsers, such as Internet Explorer, Firefox, and Safari. This means that you do not need to write different chunks of JavaScript code for each browser individually.

Scanned by CamScanner

jQuery also allows you to select DOM elements, traverse or navigate them, and modify their content. It also allows you to capture a wide variety of events, such as clicking a button or pressing a key on your keyboard. jQuery also provides built-in animation effects that you can use in your websites to make them more attractive and lively. Moreover, jQuery can be integrated with AJAX to create websites that are more spontaneous in sending and receiving the requests and responses from the server.

Basics of jQuery Programming

jQuery, released in January 2006 at BarCamp NYC by John Resig, is a framework that is created using JavaScript capabilities. It simplifies the creation of animations, communication between a Web client and a Web server, document traversing, and event handling. Some advantages of using jQuery are as follows:

- Allows you to add animated effects, such as fading in or out and sliding in or out, to HTML elements
- Enables a Web page to make AJAX requests to a Web server to add the data without reloading the page
- Allows you to manipulate DOM by adding, removing, and reordering the content of a Web page
- Allows you to create animated image slideshows, animated multi-level dropdown menus, and drag-and-drop interfaces
- Allows you to create complex forms with client-side validations and auto-complete AJAX text fields that retrieve data from a server-side database

It is important to note that jQuery supports all the concepts and functions that are available in JavaScript. A brief explanation of concepts that are common in JavaScript and jQuery is as follows:

- **String**—Refers to an immutable object that contains one or more or none characters. Some valid examples of strings are as follows:
  - "This is first line"
  - 'This is first line'
  - 'This is "my" first line'
  - "This is 'my' first line"
- **Number**—Refers to double-precision 64-bit format values. Numbers are immutable just as strings. Some valid examples of numbers are as follows:
  - 5350
  - 120.27
  - 0.26
- **Boolean**—Refers to a value that can be either true or false.
- **Object**—Refers to an instance of an entity. jQuery allows you to create objects, as shown in the following code snippet :

```
var emp =
{
   name: "Nirmal",
   age: 28
};
```

You can also write and read properties of an object by using the dot notation.

- **Array**—Refers to a collection of variables having a single name. In jQuery, you can define arrays, as shown in the following code snippet:

```
var x = [];
var y = [1, 2, 3, 4, 5];
```

- **Function**—Represents a block of code that is defined using the `function` keyword. A function can be named and anonymous. A named function has a name, while an anonymous function does not have a name. The following code snippet shows the declaration of named and anonymous functions:

```
function namedfunc()
{
...
...}

var anonymousfunc = function()
{
...
...}
```

- **Arguments variable**—Refers to a special variable that is always available inside a jQuery function. Similar to an array, this arguments variable also has the length property, but not the built-in methods of arrays. The following code snippet shows an example of the arguments variable:

```
function mymethod(x)
{
  console.log(typeof x, arguments.length);
}
mymethod ();                    //undefined 0
mymethod (1);                   //number 1
mymethod ("1", "2", "3");       //string 3
```

Another property of the arguments variable is callee, which provides the reference of the current function, as shown in the following code snippet:

```
function mymethod ()
{
   return arguments.callee;
}
mymethod ();                    //calling mymethod
```

- **The `this` keyword**—Refers to the current context of a function. The default context of the this keyword is the window object. If the this keyword is used in a function, the current context depends on how the function is called. The following code snippet shows the contexts of the this keyword:

```
$(document).ready(function()
{
  // Here the this keyword provides the reference of window object
});
$("a").click(function()
{
  // Here the this keyword provides the reference of an anchor DOM element
});
```

- **Variable scope**—Refers to the region in a program where a variable can be accessed. A variable can have either local scope or global scope. A variable having global scope can be accessed anywhere within a program, while the variable having the local scope can be accessed within the function in which it is defined.

Scanned by CamScanner

- **DOM**—Refers to a tree structure of HTML elements. jQuery provides various selectors to select DOM elements from a DOM document. Some examples of these selectors are :contains() and :file(). The :contains() selector selects all elements that contain the given text and the :file selector selects all elements that are of file type. jQuery also provides the following methods to access the DOM elements:
  - **.get()**—Returns the DOM elements that are matched with the specified jQuery object
  - **.index()**—Searches a specified element within a list of matched elements
  - **.size()**—Returns the number of DOM elements that are matched with the specified jQuery object
  - **.toArray()**—Returns the DOM elements that are matched with the specified jQuery object as array elements

Let's now learn how to load and use jQuery in a Web page.

Loading and Using jQuery

jQuery is available in the form of a single file that has the extension of .js. The latest version of jQuery is 2.0.3 and available in two formats, regular (uncompressed) and compact (or minified), in the form of the jquery-2.0.3.js and jquery-2.0.3.min.js files, respectively. You can download these files from the http://jquery.com/ link. The regular and compact formats of jQuery library provide the same functionality, but the compact format (the .min.js file) is the compressed version of the regular format. If you open both files, you can see that the .min.js file contains short names for all the variables and does not contain whitespaces and comments. The following code snippet shows how to use or load jQuery:

```
<SCRIPT type="text/javascript" language="Javascript" src=" jquery-1.5.js
  "></SCRIPT>
```

OR

```
<SCRIPT type="text/javascript" language="Javascript" src=" jquery-1.5.min.js
  "></SCRIPT>
```

You can also load a jQuery library by providing a Content Delivery Network (CDN) source in the SCRIPT element. CDN is a group of computers that are placed at different locations and connected with network. These computers contain copies of data files, such as jQuery library, and allow you to quickly access the data. Two common CDNs that host jQuery files are Microsoft and Google AJAX API. The following code snippet shows how to use Microsoft CDN to load a jQuery file:

```
<SCRIPT type="text/javascript" language="Javascript" src="
  http://ajax.microsoft.com/ajax/jquery/jquery-1.5.min.js "></SCRIPT>
```

The following code snippet shows how to use Google AJAX API CDN to load a jQuery file:

```
<SCRIPT type="text/javascript" language="Javascript"
  src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.0/jquery.min.js
  "></SCRIPT>
```

Using the jQuery Library File

You have learned that to start with jQuery, you have to download the jQuery library file. The following code snippet shows how to include the jQuery library file in a Web page:

```
<BODY>
    <SCRIPT src="jquery-1.5.js"></SCRIPT>
</BODY>
```

In the preceding code snippet, the src attribute of the `SCRIPT` element is used to specify the path of your jQuery file.

Now, jQuery reads or manipulates a DOM document by using the ready event of the document. The following code snippet shows the use of the ready event to manipulate a DOM document:

```
$(document).ready(function(){
  //Add your code here
});
```

The `ready` event always includes an event handler click corresponding to a link, which is specified with the help of the A element. The following code snippet shows the click handler in the ready event:

```
<BODY>
  <A href="www.xyz.com">jQuery</A>
  <SCRIPT src="jquery-1.5.js"></SCRIPT>
  <SCRIPt>
    $(document).ready(function(){
      $("a").click(function(event){
        alert("Hello World !!!");
      });
    });
  </SCRIPT>
</BODY>
```

In the preceding code snippet, the `ready` event is used to handle the click event of the A element by displaying the Hello World!!! message. However, executing the preceding code snippet opens a link to the `www.xyz.com` website, but does not display the message `Hello World !!!`. This is because the default task or event of the A element is linked with the opening of a link. However, if you use the `event.preventDefault()` event handler, it prevents the occurrence of the default event and raises a new event. The following code snippet shows the use of the `event.preventDefault()` event handler:

```
<BODY>
  <A href="www.xyz.com">jQuery</A>
  <SCRIPT src="jquery-1.5.js"></SCRIPT>
  <SCRIPT>
    $(document).ready(function(){
      $("a").click(function(event){
        alert("Hello World !!!");
        event.preventDefault();
      });
    });
  </SCRIPT>
</BODY>
```

In the preceding code snippet, the `event.preventDefault()` event handler displays the `Hello World !!!` message instead of opening the `www.xyz.com` website.

Describing the Callback Functions

Callback is a user-defined function that is used to invoke custom code at the specified time. For instance, you can create callback functions to handle an event, iterate a collection of nodes, or animate an image. An example of the callback function in jQuery is as follows:

```
$(selector).hide(speed,call_back)
```

In the preceding example, the call_back parameter is a function that will be executed after the hide function is executed.

Scanned by CamScanner

A callback function can be passed with or without arguments. The following code snippet shows how to pass a callback function without arguments:

```
$.get('page1.html', dummyCallBack);
```

In the preceding code snippet, dummyCallBack is a callback function that does not contain any argument and not even the parentheses.

The following code snippet shows how to pass a callback function with arguments:

```
$.get('page1.html', dummyCallBack(param1, param2));
```

In the preceding code snippet, dummyCallBack is a callback function that contains two arguments, param1 and param2.

Validating an HTML Form Using JQuery

You have learned about validation of Web pages. By using simple JavaScript code, you have validated an HTML form. You can also perform validation of HTML forms using in-built JQuery functions in your code. Let's create an HTML page ValidationUsingJQuery.html to understand validation of forms using JQuery. Listing 32 shows the code of ValidationUsingJQuery.html:

Listing 32: Creating the ValidationUsingJQuery.html File

```
<!DOCTYPE HTML>
<HEAD>
<TITLE>Validating required fields using JQuery</TITLE>
<META charset="utf-8" />
<SCRIPT LANGUAGE="javascript" TYPE="text/javascript"
  SRC="https://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></scri
  pt>
<SCRIPT language="javascript" TYPE="text/javascript"
  SRC="http://ajax.aspnetcdn.com/ajax/jquery.validate/1.10.0/jquery.validate.mi
  n.js"></SC   RIPT>
<SCRIPT>
$(function() {
   $( "#myform" ).validate();
});
</SCRIPT>
</HEAD>
<BODY>
<FORM ID="myform" ACTION = "Submit.html" METHOD="get" >
                    User Name:<INPUT TYPE= "text" NAME="uname"
  CLASS="required"/>*<br/>
          Password: <INPUT TYPE= "password" NAME="pw" CLASS="required"/>*<br/>
          <INPUT TYPE="submit" VALUE="submit"/>
   </FORM>
</BODY>
</HTML>
```

In the preceding listing, two JQuery source files are used for validating required file in the HMTL form. The `validate()` method is used to validate input fields on the HTML page. User Name and Password input fields are validated by specifying the required value for the class attribute. If a user leaves any field blank from the User Name and Password, a message "This field is required" is displayed on that field. Figure 43 shows the output of Listing 32:

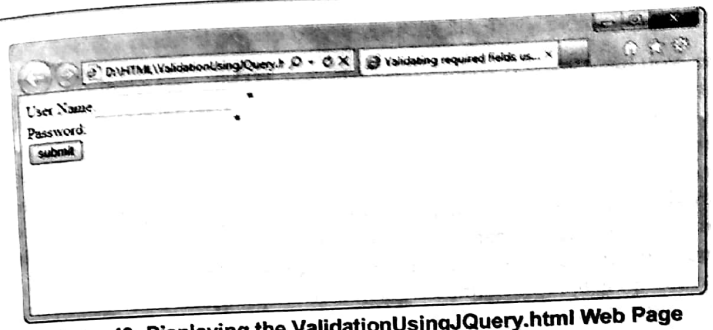

Figure 43: Displaying the ValidationUsingJQuery.html Web Page

When a user clicks on the Submit button without filling the User Name and Password fields, a message This field is required will be shown on fields, as shown in Figure 44:

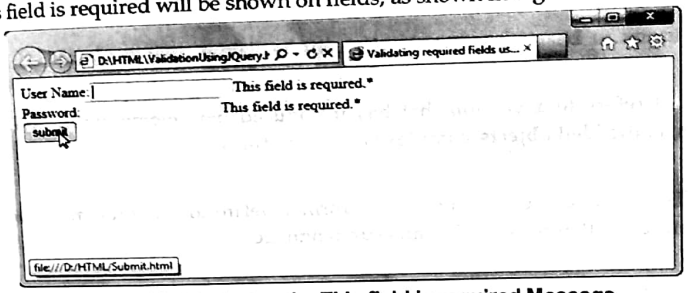

Figure 44: Displaying the This field is required Message

When users fill both fields and click the Submit button, the users redirect to the Submit.html page, as shown in Figure 45:

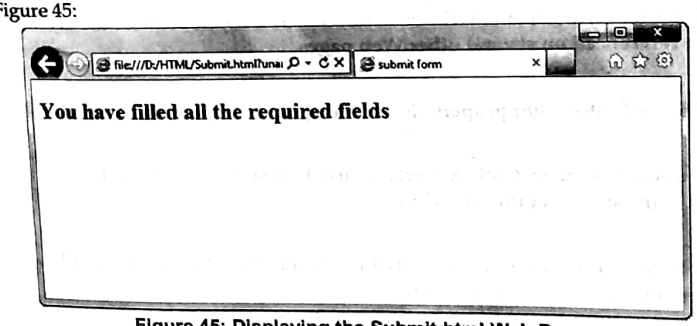

Figure 45: Displaying the Submit.html Web Page

The code for the Submit.html file is given in Listing 30.

Let's now summarize the topics that you have learned in this chapter.

Scanned by CamScanner

Summary

In this chapter, you have learned about Web designing, HTML, and client-side scripting. Today, it is crucial to have a well-designed website, in order to attract more and more visitors, for which a specific mark-up language, known as HTML, is used. The chapter started with a brief idea of the basic structure of an HTML document, the formatting of text in HTML using basic HTML elements, and use of anchors, lists, images, frames, and forms. In addition, it has introduced you to CSS, its application to decorate texts and backgrounds, and explained about the concept of links and positioning in CSS. Further, the chapter has explored the client-side scripting language JavaScript, its basic programming constructs, and various built-in objects, browser objects, and document object model in it. JavaScript is the most common client-side scripting and data-validation language used to add interactivity and dynamism to the static Web pages. Toward its end, the chapter has presented an overview of JQuery programming and the concept of validating the forms using JQuery, apart from presenting a guide to cookies and the process of form-validation using JavaScript.

Exercise

True/False

Q1. **HTML 3 refers to a version that has introduced new elements for style sheets, scripts, frames, embedded objects, complex tables, and forms.**

Ans. False

Q2. **A DTD is a separate file containing formal definition of grammar, such as supported elements and attributes used in markup language.**

Ans. **True**

Q3. **The COLGROUP in an HTML table is used to specify the number of columns in that table.**

Ans. **False**

Q4. **The internal style sheet is written within the HEAD element of the HTML document and can be referenced on several other Web pages.**

Ans. **False**

Q5. **In CSS, the font-weight property is used to specify the boldness or thickness of fonts.**

Ans. **True**

Q6. **The punctuation-trim CSS property is used to specify whether to trim the punctuation marks at the start or at the end of a line.**

Ans. **True**

Q7. **The relative text positioning property in CSS specifies the position of a block element with respect to the normal flow of the content.**

Ans. **False**

Q8. **In an HTML document, a JavaScript code can be only added or referenced inside the HEAD element.**

Ans. **False**

Q9. **In JavaScript, the confirm popup box is used to display a message as well as return a true or false value.**

Ans. True

Q10. **Consider the following number: 0.26**

In JQuery, the given number is in valid number format.

Ans. True

Multiple Choice Questions

Q1. **In which of the following versions of HTML, FIG, LISTS, LINK, and NOTE elements are introduced?**

a. HTML 1.0
b. **HTML 2.0**
c. HTML 3.0
d. **HTML 5.0**

Ans. c

Q2. **In an HTML document, which of the following tags is used to specify the version of HTML?**

a. **<!DOCTYPE>**
b. **<HTML>**
c. **<HEAD>**
d. **<BODY>**

Ans. b

Q3. **In an HTML document, which of the following tags are used to create hyperlinks?**

a. **<A></A>**
b. **<LI>**
c. **<TD></TD>**
d. **<P></P>**

Ans. a

Q4. **Which of the following is a possible value of the text-decoration-line CSS text property?**

a. Blink
b. **ink**
c. justify
d. **line-through**

Ans. d

Q5. **Which of the following sequences represents the correct sequence of applying CSS link properties for a hyperlink in an HTML document?**

a. **link→visited→hover→active**
b. **active→hover→visited→link**
c. **link→visited→active→hover**
d. **link→ active→visited→hove**

Ans. a

Q6. **In an HTML document, which of the following attributes of the SCRIPT element specifies whether the browser can continue parsing the Web page or not?**

a. async
b. src
c. defer
d. charset

Ans. c

Scanned by CamScanner

Q7. Consider the following code snippet of JavaScript:

```
<script>
var num1, num2
    num1 = 50
    num2=num1%3
    alert(num2)
</script>
```

What will be the output of the given code snippet?

a. 12 b. 3

c. 50 d. 2

Ans. d

Q8. In JavaScript code, which of the following in-built objects is used to validate patterns of characters is in correct format or not?

a. RegExp b. String

c. Boolean d. Array

Ans. a

Q9. In JQuery, which of the following methods accesses DOM elements and returns the number of DOM elements that are matched with the specified jQuery object?

a. .get() b. .index()

c. .size() d. .toArray()

Ans. c

Q10. In HTTP connections, which of the following values is used for Flag Attributes of a cookie to allow the cookie to be sent over a secure HTTP connection?

a. 0 b. 1

c. 1.1 d. 1.2

Ans. b

Short Answer Type Questions

Q1. Explain the process of inserting image in an HTML document.

Ans. HTML allows you to insert an image in a Web page with the help of IMG element. This element uses several attributes, such as src, id, lang, dir, and alt. Out of all these attributes of the IMG element, only the src attribute is necessary and all the other attributes are optional. The src attribute provides information about the location of the image file to the Web browser. If your image and HTML files are stored in the same folder, then there is no need to specify the full path of the image file in the src attribute. However, if your image and HTML files are in different folders, you have to specify the full path of the image file in the src attribute of the IMG element. The syntax to insert an image in a Web page is as follows:

```
<IMG src="name of the file" />      // when both files are in same folder
   Or
<IMG src="full path of the image file with file name"/> //when image file in
other folder
```


Q2. Describe TABLE element in an HTML document.

Ans. HTML provides you the TABLE element to create a table. The TABLE element helps you to display the information in more than one dimensions, i.e., in a tabular format. A table consists of columns and rows, where each row is divided into several data cells. A cell can contain text, lists, images, forms, and other tables. A table in a Web page can be defined by using the starting and ending tags of the TABLE element. The following code snippet shows an example of using the TABLE element:

```
<TABLE>
  ...the other table elements and the data...
</TABLE>
```

Q3. Explain the difference between P and DIV HTML elements.

Ans. The DIV element basically works as a container for other HTML elements. Using the DIV element, you can group the HTML elements into sections and apply Cascading Style Sheet (CSS) on them. The DIV element is a block-level element and cannot be used inside the P element. The P element is only used for creating paragraphs, while the DIV element is used to divide a Web page into different sections.

Q4. Explain attributes of HTML form tag.

Ans. An HTML form is a Web page that contains form elements. All the input elements should be enclosed within the opening `<FORM>` and closing `</FORM>` tags. The form tag uses the following two attributes:

- **action**—Defines where to send the data when the submit button is clicked.
- **method**—Represents the HTTP method that sends data to the action URL. The possible values for this attribute are set and get. The default value for the method attribute is get.

Q5. Explain the process for applying CSS in an HTML document inline style sheet.

Ans. The inline style properties are written in a single line separated by semicolons. These properties are placed inside the style attribute of the HTML element on which you want to apply the style, as shown in the following code snippet:

```
<P style="background:#ccc; color:#fff; border: solid black 1px;">
```

In the preceding code snippet, the P element is styled.

Q6. Discuss any three CSS text properties.

Ans. Three CSS text properties are listed as follows:

- **color**—Specifies the color of the text in a Web page. The value of the color property can be the name of the color or the hex code for the color, such as `#ff0000` is the hex code of red color.
- **line-break**—Defines a set of line breaking rules to be used with text.
- **punctuation-trim**—Specifies whether to trim the punctuation marks at the start or at the end of a line.

Q7. Write JavaScript code for adding two numbers and displaying result in an alert box.

Ans. The code for adding two numbers and displaying result in alert box is as follows:

```
<SCRIPT>
var num1=10, num2 = 20, sum
   sum = num1+num2;
   alert("sum = " + sum)  // "+" means combine into a string
</SCRIPT>
```

Scanned by CamScanner

Q8. Write a program in JavaScript to demonstrate the use of the if statement in JavaScript.

Ans. A program to demonstrate the use of the if statement in JavaScript is as follows:

```
<SCRIPT>
var x="";
var time=new Date().getHours();
if (time<12)
  {
  x="Good Morning";
  }
else if(time<17)
{
x="Good Afternoon"
}
else
{
x="Good Night"
}
alert(x);
</SCRIPT>
```

The preceding program shows a message as Good Morning, Good Afternoon or Good Night according to the current time.

Q9. Describe three common advantages of using JQuery.

Ans. Three common advantages of using jQuery are as follows:

- Allows you to add animated effects, such as fading in or out and sliding in or out, to HTML elements
- Enables a Web page to make AJAX requests to a Web server to add the data without reloading the page
- Allows you to manipulate DOM by adding, removing, and reordering the content of a Web page

Q10. Describe the callback functions in JQuery.

Ans. Callback is a user-defined function that is used to invoke custom code at the specified time. For instance, you can create callback functions to handle an event, iterate a collection of nodes, or animate an image. An example of the callback function in jQuery is as follows:

```
$(selector).hide(speed,call_back)
```

In the preceding example, the call_back parameter is a function that will be executed after the hide function is executed.

A callback function can be passed with or without arguments. The following code snippet shows how to pass a callback function without arguments:

```
$.get('page1.html', dummyCallBack);
```

In the preceding code snippet, dummyCallBack is a callback function that does not contain any argument and not even the parentheses.

Debugging Exercise

Q1. Consider the following JavaScript code

```
<SCRIPT>
var num1=1, num2=1;
for(num1=1;num1<=5;num1++)
{
num2=num2*num1;
}
alert(num2);
</SCRIPT>
```

what will be the output of this code?

Ans. The output will be 120.

Q2. Consider the following code snippet in JavaScript:

```
<SCRIPT>
var num1=0, num2=0, i=0
var myarray=[1,0,1,0,1,1,1,0,0,1]
for (i=0;i<myarray.length;i++)
  {
if(myarray[i]==0)
{
num1++
}
else
{
num2++
}
}
alert(num1 +" "+ num2)
</SCRIPT>
```

What will be the output of this code?

Ans. The output will be 4 6.

Q3. Consider the following code of JavaScript:

```
<SCRIPT>
var f_term, c_difference, nth_term;
           f_term =1
           c_difference =5
           n = 100
           nth_term = f_term + (n - 1) * c_difference;
           alert(nth_term);
</SCRIPT>
```

What will be the output of the given code?

Ans. The output will be 496.

Q4. Consider the following code snippet in JavaScript:

```
<script>
var txt1 = "Hello ";
var txt2 = "World!";
var txt3=txt1.concat(txt2.length);
var txt1=txt3.concat(txt2.length);
var txt2=txt2.concat(txt2.length);
alert(txt3+" "+txt1+" "+txt2)
</script>
```

What will be the output of the given code snippet?

Ans. The output will be Hello 6 Hello 66 World!6.

Q5. Consider the following code snippet in JavaScript:

```
<script>
var num1,num2, temp, a,b, hcf, lcm;
           num1 = 20;
```


Scanned by CamScanner

```
        num2 = 25;
      a = num1;
       b = num2;

       while (b != 0)
       {
            temp = b;
            b = a % b;
            a = temp;
       }

       hcf = a;
       lcm = (num1*num2 ) / hcf;

      alert(hcf +" "+ lcm);
</script>
```

What will be the output of the given code snippet?

Ans. The output will be 5 100.

Scanned by CamScanner

Introduction

In the previous chapters, you have learned about HyperText Markup Language (HTML), JavaScript, ASP.NET, JavaServer Pages (JSP), Hypertext Preprocessor (PHP), and ADO.NET. These are the major technologies used for Web development in today's scenarios. Apart from these main stream technologies, some other Web technologies are also used with them for building different types of Web applications.. XML is developed by the World Wide Web Consortium (W3C), which is used as a format to store and transfer data over the Web. XML is a simple and flexible language, which is widely used to exchange a large variety of data, such as text, images, and video, over the Internet.

XML is also the foundation for Web Services Description Language (WSDL). Web services comprise programming, data, and sometimes human resources as well. A Web user or Web-connected program can avail these services from a Web server. Overall demand on network bandwidth is one of the major concerns that may occur due to increase in Web services. Software developers are made available with a number of new emerging products for creating and modifying the existing applications. These applications are then made accessible as Web services. An XML document can be converted into another format, such as HTML and Portable Data Format (PDF), to make it more readable. Extensible Stylesheet Language Transformations (XSLT) is used to parse an XML document into any other format using XSLT language; for instance, parsing an XML document in an HTML format on the basis of instructions given in an XSLT stylesheet. The XSLT stylesheet is a document that provides instructions to transform the structure and content of an XML document into other formats, such as HTML, XHTML, and PDF. In this chapter, you learn about the basics of XML. You also learn about XSL, XSLT and its elements, and transforming an XML document using XSLT, and basics of Web Services. Further, the chapter explains Really Simple Syndication (RSS) Web feed, which is an easy way for creating and using distributed or scattered content.

Let's first learn about XML.

Introducing XML

XML is a markup language based on simple, platform-independent rules for processing and displaying textual information in a structured way. The platform-independent nature of XML makes an XML document an ideal format for exchanging structured textual information among different applications. In the recent years, XML has evolved as the de facto standard for document markup. Since its advent, XML has been used to implement the following various operations:

- Configuration information
- Publishing
- Electronic Data Interchange (EDI)
- Voicemail systems
- Vector graphics
- Remote Method Invocation (RMI)
- Object serialization

XML provides customized tags to format and display textual information. What makes XML an integral element of enterprise computing is that XML documents are simple text documents that represent data in a platform-neutral manner. For example, an XML document generated by an application running on Microsoft Windows can be easily consumed by an application running on Sun Solaris.

XML Basics

This section explains the basics of XML and what XML is all about and how it is used. Let's know about the major components that make XML great for information storage and interchange. An XML document contains many components, such as syntax, elements, attributes, and declarations. The following components are discussed in brief in this section:

- XML syntax
- XML declaration
- XML elements
- XML attributes
- Viewing XML
- XML parser

XML Syntax

Every XML document abides by some syntax rules that specify how a document is created. The declaration statements or markup tags define the syntax for creating XML documents. The syntax used to create an XML document is called markup syntax. It is used to define the structure of data in the document. The following rules are associated with the markup syntax:

- XML documents support both in-built and user-defined tags
- XML documents must have a starting tag and a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML documents must have a root element and only one root element
- XML attributes values must be quoted
- In XML, white space is preserved

Let's consider an XML file as shown in the following code snippet:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Employee>
  <h1><ul><FirstName>Ambrish</FirstName></ul></h1>
  <MiddleName>Kumar</MiddleName>
  <LastName>Singh</LastName>
  <Age>25</Age>
  <EMPID id="A24"> </EMPID>
</Employee>
```

In the preceding code snippet, the document contains numerous starting and closing tags. These tags are case sensitive. It means if you write `<FirstName>` as a starting tag and close it with `</firstname>`, then it would generate an error. In the code, given previously, all elements are properly nested. For example, consider the following line of code:

```
<h1><ul><FirstName>Ambrish</FirstName></ul></h1>
```

In this code, the first tag is `<h1>` and the line closes with the same tag in the same order. You cannot write like this:

```
<h1><ul><FirstName>Ambrish</FirstName></h1></ul>
```

In this case, it will give an error.

Scanned by CamScanner

XML Declaration

The XML declaration statement is used to indicate that the specified document is an XML document. Although it is not required to have an XML declaration, it is considered a good practice to include it. If the XML declaration statement is present, then it will be the first line in the document which defines the XML version and character encoding. An XML declaration is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

Some important points to note about XML declaration are as follows:

- XML declaration starts with `<?xml` and ends with `?>`
- If it is included, then it must include the `version` attribute as it is required, but the encoding and standalone attributes are optional
- The XML declaration must be at the beginning of the file
- The version, encoding, and standalone attributes must be in the same order as shown in the preceding example of XML declaration

XML Elements

An XML element is the basic syntactic construct of an XML document. You can say that the building blocks of any XML document are its elements. A start tag and an end tag delimit an element in an XML document. An example of an XML element is as follows:

```
<Employees></Employees>
```

A start tag within an element is delimited by the < and > characters and has a tag name. In the previous start tag, the name is `Employees`. However, it is useful to keep in mind that a tag name must begin with a letter and can contain hyphen (-) and underscore (_) characters. An end tag is delimited by the </ and > character sequences and also contains a tag name. A document must have a single root element, which is also known as the document element. If you assume that the `Employees` element is your root element, then your document would be as follows:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<Employees></Employees>
```

This is an example of a well-formed XML document, where the XML declaration on the first line is optional; omitting the XML declaration would still leave you with a well-formed document.

However, the following rules are associated with elements:

- Element names can start with letters or underscore (_) character, but not numbers or other punctuation characters.
- Element names cannot contain spaces.
- Element names cannot start with the letters `xml`, in uppercase, lowercase, or mixed (i.e., `xml`, `XML`, `XmL`, or any other combination).
- There cannot be a space after the opening < character; the name of the element must come immediately after it. However, there can be space before the closing > character, if desired.

Nested Elements

An XML element can contain other nested elements. For example, the root element may contain a nested element having the text content "Ambrish":

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<Employees>
  <FirstName>Ambrish</FirstName>
</Employees>
```


Here, the `<FirstName>` element is nested in the `<Employees>` element and contains the text `Ambrish`.

Empty Elements

An element may have no nested element or content. Such an element is termed an empty element, and it can be written with a special start tag that has no end tag. For example, the `<MiddleName/>` is an empty element. If you include this empty element within your document, the document will appear as follows:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<Employees>
  <FirstName>Ambrish</FirstName>
  <MiddleName/>
</Employees>
```

Elements can have attributes that are specified in the start tag. An example of an attribute is `<LastName title="Singh"></LastName>`. An attribute is defined as a name-value pair. In the previous example, the name of the attribute is `title`, and the value of the attribute is `Singh`. With an attribute added, the example document appears as follows:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<Employees>
  <FirstName>Ambrish</FirstName>
  <MiddleName/>
  <LastName title="Singh"></LastName>
</Employees>
```

Escaping Delimiter Characters

XML tags use angle brackets (> and < symbols) for enclosing tag names. Thus, these characters cannot appear in the element data. If you want to include these symbols in your data, you must use the replacement character sequences. Table 1 lists all the special characters and their corresponding replacement character sequences in XML:

Table 1: Special Characters and Replacement Character Sequences in XML

Special characters	Replacement characters
&	&amp;
'	&apos;
"	&quot;
<	&lt;
>	&gt;

Now, let's assume that you want to add another element `Birthday` having an attribute name `date` with the value `<24/01/1982>`. It has been mentioned in the earlier section that you are not allowed to include delimiter characters within an attribute value. However, you can use the `<` character sequence to escape < and the `>` character sequence to escape >. So, with that in place, the document now appears as follows:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<Employees>
  <FirstName>Ambrish</FirstName>
  <MiddleName/>
  <LastName title="Singh"></LastName>
```

Scanned by CamScanner

```
    <Birthday date="&lt; 24/01/1982&gt;"> </Birthday>
</Employees>
```

Another mechanism for including delimiter characters within the body of a construct is to use escaped numeric references. For example, the numeric American Standard Code for Information Interchange (ASCII) value for the > character is 62. So, you can use the `&#62;` character sequence instead of `>`. Using escaped numeric references is the most general mechanism for including delimiter characters within a construct's body.

XML Attributes

XML attributes provide additional information about elements. Let's again consider the following code snippet:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<Employees>
  <FirstName>Ambrish</FirstName>
  <MiddleName/>
  <LastName title="Singh"></LastName>
  <Birthday date="&lt;24/01/1982&gt;"></Birthday>
</Employees>
```

Here, `date` is an attribute for the element `<Birthday>` as the `title` is for the element `<LastName>`. Attributes values are always given in single or double quotes. In case the value itself contains double quotes, then give it in single quote; otherwise, use the double quote syntax. Attributes are used for storing data. However, the data can also be stored in child elements. There is no rule associated with XML documents as to when to use child elements or attributes for storing data. The best practice says that you should avoid using attributes. Some problems while using attributes are as follows:

- Attributes have no multiple values, whereas child elements have
- You cannot expand attributes easily
- Attributes cannot describe structures, whereas child elements can
- Attributes are difficult to maintain programmatically
- It is difficult to test attributes values against Document Type Definition (DTD)

Viewing XML

Let's now understand how XML documents are viewed in your browsers. You can use XML for any browser. Each XML file contains a plus (+) or minus (-) sign on the left of the elements. When you click on the minus sign, the elements collapse, and when you click on the plus sign, the elements expand. For example, when you open the valid_doc.xml in Internet Explorer (IE), it will appear as shown in Figure 1.

A sample of a valid XML document (valid_doc.xml) is shown in Listing 1:

Listing 1: Showing the Code for the valid_doc.xml File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<books>
<book>
<subject standard="class V">History</subject>
<author>D.P.Mishra</author>
<publisher>FGB</publisher>
<price>Rs200/-</price>
</book>
```



```
<book>
<subject standard="class IV">Math</subject>
<author>P.K.Sharma</author>
<publisher>FGB</publisher>
<price>Rs100/-</price>
</book>

<book>
<subject standard="class VI">English</subject>
<author>T.P.Sastri</author>
<publisher>FGB</publisher>
<price>Rs150/-</price>
</book>

<book>
<subject standard="class V">Geographt</subject>
<author>P.Sharma</author>
<publisher>FGB</publisher>
<price>Rs240/-</price>
</book>
</books>
```

The valid_doc.xml file will be displayed as a Web page as shown in Figure 1:

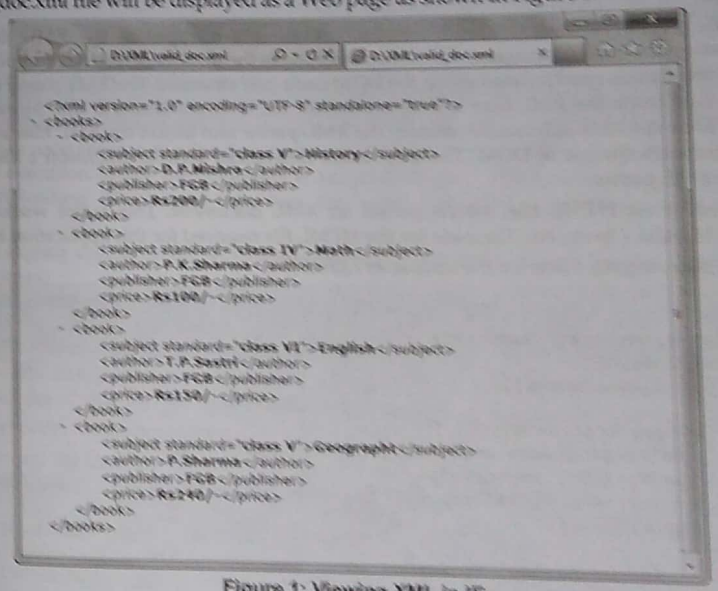

Figure 1: Viewing XML in IE

Suppose you click on the second and third minus (-) signs from the top, then the display will appear as shown in Figure 2:

Scanned by CamScanner

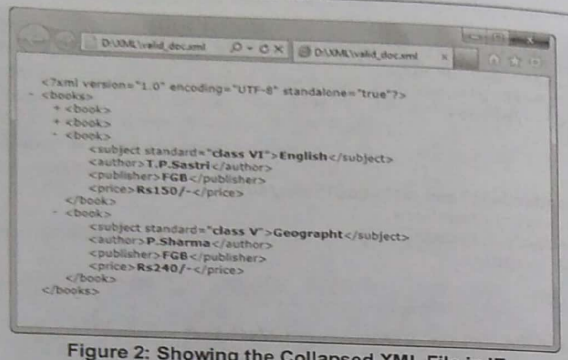

Figure 2: Showing the Collapsed XML File in IE

In Listing 1, the XML declaration line specifies the version and encoding format of the XML document. Next, the root element, that is, books, contains the book element four times. The book element further contains the subject, author, publisher, and price child elements.

XML Parser

The XML parser is used to read, update, create, and manipulate the XML document. For manipulating the XML document, the XML parser loads the document into the computer's memory and then the data is manipulated using the DOM node-tree structure. The XML parser is a part of the software that reads the XML files and tests whether the XML document is well formed against the given DTD or the XML schema. Moreover, the XML parser also makes the XML files available to the application with the use of DOM. There are some differences between Microsoft's XML parser and Mozilla's XML parser.

Let's consider an HTML file, which parses an XML document. This script works for Internet Explorer, Mozilla, Opera, etc. The code for the HTML file required for this application is in Listing 2:

Listing 2: Showing the Code for the xmlparser2.html File

```
<html>
<head>
<script type="text/javascript">
var xmlParseDoc;
function loadParseXML()
{
  if (window.ActiveXObject) {
        xmlParseDoc=new ActiveXObject("Microsoft.XMLDOM");
        xmlParseDoc.async=false;
        xmlParseDoc.load("emp.xml");
        getmessage();
  }
  else if (document.implementation &&
  document.implementation.createDocument) {
        xmlParseDoc=document.implementation.createDocument("","",null);
        xmlParseDoc.load("emp.xml");
        xmlParseDoc.onload=getmessage;
  } else {
        alert('The Script is not Compatible with your Browser');
  }
}
function getmessage()
{
  document.getElementById("FirstName").innerHTML=
  xmlParseDoc.getElementsByTagName("FirstName")[0].childNodes[0].nodeValue;
  document.getElementById("MiddleName").innerHTML=
  xmlParseDoc.getElementsByTagName("MiddleName")[0].childNodes[0].nodeValue;
  document.getElementById("LastName").innerHTML=
  xmlParseDoc.getElementsByTagName("LastName")[0].childNodes[0].nodeValue;
  document.getElementById("Age").innerHTML=
  xmlParseDoc.getElementsByTagName("Age")[0].childNodes[0].nodeValue;
}
</script>
</head><body onload="loadParseXML()">
<h1>Employee Details</h1>
<p><b>First Name:</b> <span id="FirstName"></span><br />
<b>Middle Name:</b> <span id="MiddleName"></span><br />
<b>Last Name:</b> <span id="LastName"></span><br />
<b>Age:</b> <span id="Age"></span>
</p>
</body>
</html>
```

Consider the following lines:

```
xmlParseDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlParseDoc.async=false;
  xmlParseDoc.load("emp.xml");
```

In the preceding code snippet, the first line creates an instance of Microsoft's XML parser. The script will not start execution before the document is properly loaded. This can be done by turning off asynchronized loading. The second line does the same. The third line simply loads the XML documents.

The code will slightly change when performing the same activities in Mozilla, Firebox, and Opera. It appears as follows:

```
xmlParseDoc=document.implementation.createDocument("","",null);
  xmlParseDoc.load("emp.xml");
  xmlParseDoc.onload=getmessage;
```

In the first line, the first parameter specifies the XML namespace, the second parameter specifies the root of the XML document, and the third is always null as it is not implemented yet.

Listing 3 presents the XML document:

Listing 3: Showing the Code for the emp.xml File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Employee>
  <FirstName>Ambrish</FirstName>
  <MiddleName>Kumar</MiddleName>
  <LastName>Singh</LastName>
  <Age>25</Age>
</Employee>
```


Scanned by CamScanner

When you run the `xmlparser2.html` file on your browser, a Web page is displayed as shown in Figure 3:

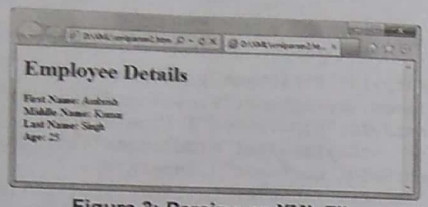

Figure 3: Parsing an XML File

You have learned the basics of XML. Let's learn about XSL and XSLT in the next section.

Introducing XSL

XSL is a language used for transforming and formatting XML documents. W3C developed XSL because an XML-based stylesheet language was needed to transform and format XML documents. XSL began as a single language to serve the purpose, but further, it was divided into three parts. The three parts of the XSL language are as follows:

- **XSLT**—Refers to a language used for transforming XML document
- **XPath**—Refers to a language used for navigating in XML document
- **XSL-FO**—Refers to a language used for formatting XML document

Introducing XSLT

XSLT is an XML-based language that allows you to transform the XML documents into other formats, such as HTML, XHTML, or PDF. It is considered as a declarative language because it declares what the program should perform rather than how to perform. To transform an XML document into an HTML or XHTML document, XSLT uses parsers and stylesheets. XSLT parsers parse an XML document on the basis of the rules and instructions defined in the XSLT stylesheet.

An XSLT stylesheet contains XSLT instructions in the form of XML elements and attributes and transforms your XML file in other formats. XSLT helps you to add or remove elements and attributes from the final output. It also enables you to rearrange the elements, sort the elements, and perform tests to make decisions about the hiding or displaying of elements. Similar to a programming language, XSLT provides various types of data types, such as boolean, number, and string; operation elements, such as xsl:template and xsl:sort; and flow-control elements, such as `xsl:if`, `xsl:for-each`, and `xsl:choose`.

Let's learn about XSLT elements and attributes that are required to parse the XML data into other formats.

Exploring XSLT Elements and Attributes

The XSLT language became a W3C recommendation on November 16, 1999. It describes a transformation process to transform an XML source document into an XML result document. An XML source document is an XML file, while the XML result document is a file in other format, such as an HTML file, XHTML file, or a PDF file. XSLT uses the XPath language to define parts of the source documents in the transformation process. These documents must match with at least one predefined templates. When the document matches with the templates, XSLT transforms the matching part of source document in the result document. To use the XSLT language, you need to

create a .xsl file that represents a stylesheet for an XML document. This stylesheet should contain either the `xsl:stylesheet` element or the `xsl:transform` element to transform files of one format into another.

You can declare a stylesheet by using either of the following syntaxes:

```
<xsl:stylesheet>
    :
</xsl:stylesheet>
```

or

```
<xsl:transform>
    :
</xsl:transform>
```

The preceding syntaxes show how to declare an XSLT stylesheet.

Table 2 describes some of the attributes of the `xsl:stylesheet` and `xsl:transform` elements:

Table 2: Attributes of the xsl:stylesheet and xsl:transform Elements

Name	Value	Description
version (mandatory)	version	Specifies the XSLT version of the stylesheet
extension-element-prefixes (optional)	list	Represents a white-space separated list of namespace prefixes used for extension elements
exclude-result-prefixes (optional)	list	Represents a white-space separated list of namespace prefixes that should not be sent to the output
id (optional)	name	Represents the unique id for the stylesheet

You can use the attributes described in Table 2 with both the `xsl:stylesheet` and `xsl:transform` elements, as shown in the following code snippet:

```
<xsl:stylesheet
  id="name of stylesheet"
  version= "version of XSLT"
  extension-element-prefixes="list"
  exclude-result-prefixes="list">
  :
  :
</xsl:stylesheet>
```

The preceding code snippet shows a stylesheet with all the attributes of the `xsl:stylesheet` element.

Table 3 describes a complete list of XSLT elements:

Table 3: Lists of XSLT Elements

Element	Description
`xsl:apply-imports`	Specifies an element that can invoke an overridden template rule
`xsl:apply-templates`	Specifies an element that can direct the XSLT processor to find the appropriate template to apply on other nodes of the XML document on the basis of type and context of the selected node
`xsl:attribute`	Specifies an element that creates an attribute node
`xsl:attribute-set`	Specifies an element that creates a set of attributes
`xsl:call-template`	Specifies an element that invokes a template by using its name

Scanned by CamScanner

Table 3: Lists of XSLT Elements

Element	Description
xsl:choose	Specifies an element that provides multiple conditional testing in conjunction with the xsl:otherwise and xsl:when elements
xsl:comment	Specifies an element that generates comments in the output
xsl:copy	Specifies an element that can copy a node from a source file to the output
xsl:copy-of	Specifies an element that copies and inserts the data of parent elements and child elements in the resultant output
xsl:decimal-format	Specifies an element that can display a number in decimal format
xsl:element	Specifies an element that creates an element node with the specified name
xsl:fallback	Specifies an element that handles those XSLT elements that are not handled by a parser
xsl:for-each	Specifies an element that applies a template repeatedly to every node of a set
xsl:if	Specifies an element that implements a simple if conditional in a template
xsl:import	Specifies an element that imports another XSLT file
xsl:include	Specifies an element that includes another XSLT file in the current XSLT file
xsl:key	Specifies an element that declares a named key
xsl:message	Specifies an element that sends a text message to either the message buffer or a message dialog box
xsl:namespace-alias	Specifies an element that replaces the prefix associated with a given namespace with another prefix
xsl:number	Specifies an element that inserts a number with the specified format in the final output
xsl:otherwise	Specifies an element that provides multiple conditional testing in conjunction with the xsl:choose and xsl:when elements
xsl:output	Specifies an element that produces output
xsl:param	Specifies an element that helps in declaring parameters with the xsl:stylesheet or xsl:template element
xsl:preserve-space	Specifies an element that preserves white space in a document
xsl:processing-instruction	Specifies an element that generates a processing instruction in the output
msxsl:script*	Specifies an element that defines global variables and functions for script extensions
xsl:sort	Specifies an element that inserts sort criteria for node lists selected by using the xsl:for-each or xsl:apply-templates
xsl:strip-space	Specifies an element that removes white space from a document
xsl:stylesheet	Specifies an element that defines a root element of the style sheet
xsl:template	Specifies an element that defines a reusable template to produce a desirable output
xsl:text	Specifies an element that generates text in the output
xsl:transform	Specifies an element that defines a root element of the style sheet
xsl:value-of	Specifies an element that inserts the value of a selected node as text

Table 3: Lists of XSLT Elements

Element	Description
xsl:variable	Specifies an element that acts as a variable whose value can be varied in an expression
xsl:when	Specifies an element that provides multiple conditional testing in conjunction with the xsl:choose and xsl:otherwise elements
xsl:with-param	Specifies an element that passes parameters to a template

Let's describe some commonly used XSLT elements in the following sections.

- The xsl:template element
- The xsl:apply-templates element
- The xsl:import element
- The xsl:call-template element
- The xsl:include element
- The xsl:element element
- The xsl:attribute element
- The xsl:attribute-set element
- The xsl:value-of element

Let's now discuss all these elements in detail.

The xsl:template Element

The xsl:template element defines a template or a set of rules to produce output. It is a top-level element and uses a match attribute for defining different names and patterns for nodes. The nodes are matched with a template on the basis of these names and patterns by using the match attribute of the xsl:template element. The syntax of the xsl:template element is as follows:

```
<xsl:template
name="name"
match="pattern"
mode="mode"
priority="number">

<!-Content-->

</xsl:template>
```

In the preceding syntax, the following attributes of the xsl:template element are used:

- **name**—Specifies a name for the template
- **match**—Specifies a pattern for the template, such as /
- **mode**—Specifies a mode for the template
- **priority**—Specifies a number to indicate the numeric priority of the template

Now, let's consider the following code snippet showing the name attribute of the xsl:template element:

```
<xsl:template name="PRODUCTNAME" >
```

In the preceding code snippet, the name of the node is PRODUCTNAME. After defining a template, you need to call the xsl:template element by using either the xsl:apply-templates or

Scanned by CamScanner

xsl:call-template element to apply it on an element. The following code snippet shows the use of the xsl:call-template element:

```
<xsl:call-template name="PRODUCTNAME" >
```

Note that the name of the node specified in the xsl:template element and in the xsl:call-template element must be matched.

Table 4 describes the attributes of the xsl:template element:

Table 4: Attributes of the xsl:template Element

Name	Value	Description
match (optional)	Pattern	Specifies which node is eligible to be processed. If this attribute is absent, then the name attribute must be present.
name (optional)	Name	Specifies a name for a template. If this attribute is absent, then there must be a match attribute.
priority (optional)	Number	Represents a number (positive or negative, integer, or decimal) to denote the priority of a template. This attribute is used when several templates match the same node.
mode (optional)	List of mode names	Specifies the mode or modes to which this template rule applies.

The xsl:apply-templates Element

The xsl:apply-templates element defines a set of nodes to be processed by selecting an appropriate template rule. It is an instruction which is used within a template. This element selects a set of nodes and processes each node by finding a matching template. The xsl:sort element may be nested within the xsl:apply-templates element. If the xsl:sort element is nested within the xsl:apply-templates element, then it determines the order in which the nodes are processed; otherwise, the nodes are processed in the order they are specified in the document.

The syntax of the xsl:apply-templates element is as follows:

```
<xsl:apply-templates  select="expression" mode="mode" >
<!-- --- Content: (xsl:sort|xsl:with-param*) -- --- >
</xsl:apply-templates>
```

The xsl:apply-templates element contains two attributes, which are described in Table 5:

Table 5: Attributes of the xsl:apply-templates Element

Name	Value	Description
select (optional)	expression	Specifies the nodes to be processed. If this attribute is not specified, then the parser processes all the children of the current node.
mode (optional)	name	Represents template rules used to process the selected nodes.

The xsl:import Element

The xsl:import element imports the content of one stylesheet into another. The importing stylesheet is higher in precedence than the imported stylesheet. It means that the XML document is first transformed using the importing stylesheet, but if importing stylesheet is unable to perform the transformation, then the imported stylesheet will perform the transformation, if it can. The xsl:import element is the first element among the top-level elements. To import the InsertStylesheet.xsl stylesheet, you can use the xsl:import element, as shown in the following

code snippet (assuming that InsertStylesheet.xsl is situated in the same directory as the importing stylesheet):

```
<xsl:import href="InsertStylesheet.xsl"/>
```

The value of the href attribute may be a relative Uniform Resource Identifier (URI), as shown in the preceding code snippet, or an absolute URI. Note that the href attribute should contain a valid reference of an XSLT stylesheet.

The syntax of the xsl:import element is as follows:

```
<xsl:import  href="URI"/ >
```

The xsl:import element contains one attribute, which is described in Table 6:

Table 6: Attribute of the xsl:import Element

Name	Value	Description
href	URI	Specifies the URI of the imported stylesheet

The xsl:call-template Element

It has been discussed earlier that the xsl:template element may have a name attribute and can be used to call an element by its name. The xsl:template does not work alone. It is manually called by using the xsl:call-template element, which takes an attribute, the name attribute. Note that the name attribute of the xsl:call-template element and the name attribute of the xsl:template element must match with each other; otherwise, the specified template will not be applied on an element. The xsl:call-template element may have one or more nested xsl:with-param elements.

The syntax of the xsl:call-template element is as follows:

```
<xsl:call-template name ="templatename" >
<!-- --- Content: xsl:param* -- --- >
</xsl:call-template>
```

Now, consider the template shown in the following code snippet:

```
<xsl:template name="CalledTemplate">
<!-- The works of the template goes here. -->
</xsl:template>
```

The template shown in the preceding code snippet can be called by using the xsl:call-template element, as shown in the following code snippet:

```
<xsl:call-template name="CalledTemplate"/>
```

You can also pass a parameter to this template, as shown in the following code snippet:

```
<xsl:call-template name="CalledTemplate">
<xsl:with-param name="ID" select="StudentID"/>
</xsl:call-template>
```

Table 7 describes the attribute of the xsl:call-template element:

Table 7: Attribute of the xsl:call-template Element

Name	Value	Description
name	templatename	Specifies the name of the template to be called

The xsl:include Element

The xsl:include element is a top-level element, i.e., an element that comes in the starting of XML stylesheet, and used to include the content of one stylesheet into another. It has only one attribute,

Scanned by CamScanner

the href attribute, which is a required attribute. The syntax of the xsl:include element is as follows:

```
<xsl:include href="URI"/ >
```

Now, include a stylesheet called Included.xsl in the href attribute of the xsl:include element, as shown in the following code snippet:

```
<xsl:include href="Included.xsl"/>
```

The preceding code snippet shows how to include a stylesheet by using the xsl:include element.

Table 8 describes the href attribute of the xsl:include element:

Table 8: Attribute of the xsl:include Element

Name	Value	Meaning
href	URI	Specifies the URI of the included stylesheet

The xsl:element Element

The xsl:element element provides the qualified name for an element. The purpose of the xsl:element element is to create an element node in the result tree. The syntax of the xsl:element element is as follows:

```
<xsl:element name = "name" namespace ="URI" use-attribute-sets = "namelist" >
< ! -- -- Content:template -- -- >
</xsl:element>
```

Attributes can be added to a new element node by using the xsl:attribute, xsl:copy, or xsl:copy-of element.

Table 9 describes the attributes of the xsl:element element:

Table 9: Attributes of the xsl:element Element

Name	Value	Description
name (mandatory)	Name	Represents the name of the element to be generated
namespace (optional)	URI of namespace	Represents the namespace URI of the generated element
use-attribute-sets (optional)	Whitespace-separated list of attribute-sets	Represents the list of the named attributes

The xsl:attribute Element

The xsl:attribute element is used to create the attribute node and add it to any element of XSLT.

The syntax of the xsl:attribute element is as follows:

```
<xsl:attribute name = "nameofattribute" namespace ="URI" >
< ! -- -- Content:template -- -- >
</xsl:attribute>
```

The xsl:attribute element has two attributes, which are described in Table 10:

Table 10: Attributes of the xsl:attribute Element

Name	Value	Description
name (mandatory)	Name	Represents the name of the attribute to be generated
namespace (optional)	URI of namespace	Represents the namespace URI of the generated attribute

The xsl:attribute-set Element

The xsl:attribute-set element is one of the starting elements of an XSL stylesheet and is used to define a named set of attribute names and values. The resulting attribute can be applied as a whole to any output element. The syntax of the xsl:attribute-set element is as follows:

```
<xsl:attribut-set name = "name" use-attribute-sets = "namelist" >
< ! -- -- Content:template -- -- >
</xsl:attribute-set>
```

Table 11 describes the attributes of the xsl:attribute-set element:

Table 11: Attributes of the xsl:attribute-set Element

Name	Value	Description
name (mandatory)	Name	Represents the name of the element to be generated
use-attribute-sets (optional)	Whitespace-separated list of attribute-sets	Represents the list of named attributes

The xsl:value-of Element

The xsl:value-of element is mostly used for writing text to a result tree. It is used for constructing a Text node and extracting the value of the node. The syntax of the xsl:value-of element is as follows:

```
<xsl:value-of
select = "expression"
disable-output-escaping = "yes|no"/ >
```

Table 12 describes the attributes of the xsl:value-of element:

Table 12: Attributes of the xsl:value-of Element

Name	Value	Description
select (mandatory)	expression	Represents the expression given in the XPath expression language form.
disable-output-escaping (optional)	Yes or No	Specifies that the special character (such as <) will be displayed as it is or as < according to the value of Yes or No. The default value is No.

Now, let's explore variables and parameters used in XSLT.

Exploring XSLT Variables and Parameters

A variable is a location in the computer's memory where you can store a piece of information or some data. For example, you can store the phone number of your friend in the computer's memory with the help of the phnumber variable. XSLT also uses variables for storing the values or information in the stylesheet. The variables can be used to specify global constants that are available throughout a stylesheet or to specify local constants that are available within the areas in which they are defined. To declare a variable in a stylesheet, you can use the xsl:variable element, as shown in the following code snippet:

```
<xsl:variable
name="name"
select="expression">
<!--Content:template-->
</xsl:variable>
```

Scanned by CamScanner

The preceding code snippet declares the name variable. The `xsl:variable` element contains two attributes, name and select.

Table 13 describes the attributes of the `xsl:variable` element:

Table 13: Attributes of the xsl:variable Element

Name	Value	Description
`name` (mandatory)	name	Specifies the name of the variable
`select` (optional)	expression	Specifies the value of the variable

Now, let's consider the following code snippet, which shows the declaration of the variable:

```
<xsl:variable name="color" select="red">
</xsl:varibale>
```

You can also declare the same variable in one line, as shown in the following code snippet:

```
<xsl:variable name="color" select="red"/>
```

The preceding code snippet declares a variable whose name is color and the value is red.

Similar to variables, parameters can also be used to store data or information. The parameters can be specified within the templates or in the user-defined functions. They can also be passed as a part of the calling process of the function. To declare a parameter, you can use the `xsl:param` element, as shown in the following code snippet:

```
<xsl:param
name="name"
select="expression">
<!--Content:template-->
</xsl:param>
```

The preceding code snippet declares the name parameter. The `xsl:param` element contains two attributes, name and select.

Table 14 describes the attributes of the `xsl:param` element:

Table 14: Attributes of the xsl:param Element

Name	Value	Description
`name` (mandatory)	name	Specifies the name of the parameter
`select` (optional)	expression	Specifies the default value of the parameter

Now, let's consider the following code snippet, which shows the declaration of the parameter:

```
<xsl:param name="x" select="1"/>
```

The preceding code snippet declares a parameter whose name is x and value is 1.

The following section discusses transforming an XML document using XSLT.

Transforming an XML Document Using XSLT

As already discussed, XSLT is used to transform an XML document into other documents, such as HTML or XHTML. Let's create an XML document, named product.xml, to transform it into an HTML document.

Listing 4 shows the code of the product.xml file:

Listing 4: Showing the Code of the product.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="product.xsl"?>
<PRODUCTDATA>
    <PRODUCT>
        <PRODID id="P001" ></PRODID>
        <PRODUCTNAME>Barbie Doll</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group below 5 years
            </DESCRIPTION>
        <PRICE>240.00</PRICE>
        <QUANTITY>12</QUANTITY>
    </PRODUCT>
    <PRODUCT>
        <PRODID id="P002" ></PRODID>
        <PRODUCTNAME>Mini Bus</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group of 5-10 years
            </DESCRIPTION>
        <PRICE>420.00</PRICE>
        <QUANTITY>6</QUANTITY>
    </PRODUCT>
    <PRODUCT>
        <PRODID id="P003" ></PRODID>
        <PRODUCTNAME>Car</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group of 10-15 years
            </DESCRIPTION>
        <PRICE>600.00</PRICE>
        <QUANTITY>21</QUANTITY>
    </PRODUCT>
    <PRODUCT>
        <PRODID id="P004" ></PRODID>
        <PRODUCTNAME>Air Plane</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group of 08-15 years
        </DESCRIPTION>
        <PRICE>700.00</PRICE>
        <QUANTITY>25</QUANTITY>
    </PRODUCT>
</PRODUCTDATA>
```

Listing 4 creates an XML document, which contains the details of the products, such as product id, product name, price, quantity, and description. The following code snippet from Listing 4 indicates that the XML document is using the XSLT stylesheet:

```
<?xml-stylesheet type="text/xsl" href="product.xsl"?>
```

When you open the XML document in a browser, the browser calls the product.xsl stylesheet. Let's create a stylesheet named product.xsl to transform the product.xml document into an HTML document. Listing 5 shows the code of the product.xsl file:

Listing 5: Showing the Code of the product.xsl File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Product Details</h2>
<table border="1">
<tr bgcolor="#CCCFFF">
<th align="left">Product Name</th>
<th align="left">Quantity</th>
<th align="left">Price</th>
</tr>
```

Scanned by CamScanner

```
    <xsl:for-each select="PRODUCTDATA/PRODUCT">
    <tr>

    <td><xsl:value-of select="PRODUCTNAME"/></td>
    <td><xsl:value-of select="QUANTITY"/></td>
    <td><xsl:value-of select="PRICE"/></td>
    </tr>

    </xsl:for-each>
    </table>
  </body>
  </html>

  </xsl:template>
  </xsl:stylesheet>
```

Listing 5 creates a stylesheet, which extracts data from the XML document and displays the data in the tabular format. It extracts the name, quantity, and price of the product by using the `xsl:value-of` element, as shown in the following code snippet:

```
<xsl:value-of select="PRODUCTNAME"/>
<xsl:value-of select="QUANTITY"/>
<xsl:value-of select="PRICE"/>
```

When the product.xml document is opened in a browser, an HTML page appears, which displays the name, quantity, and the price of the products in a tabular format.

Figure 4 shows the output of the product.xml file:

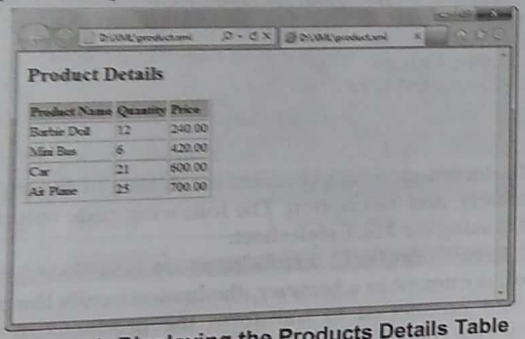

Figure 4: Displaying the Products Details Table

As you see, when you open an XML document, an HTML document appears as output and displays the content of the XML document in a table by using the XSLT stylesheet.

You have learned about transforming an XML document using XSLT. Let's learn about Web feeds.

Introducing Web Feeds

Nowadays, people frequently search contents for any updates and download Web pages from different websites ranging from new portals to job portals. This process takes a lot of time as the users have to open each website and browse the particular pages that they want to access for the updates. This can be done in a much simpler way by accessing the updated content using Web syndication format such as RSS.

Syndication of a Web page is a fast and simple method to generate and use scattered content like breaking news. Presently, RSS in used in many fields such as marketing, bug reporting, consistent updating, and publications. RSS is very commonly used on various blogs and community websites. RSS facilitates the websites to send preview of articles and alerts about new updates and products to customers.

Let's learn about RSS feed.

Introducing RSS Feed

RSS is one of the Web feed formats which keeps you updated of the changes occurring in selected website. A Web feed provides regularly updated content of a Web page. It is a document (mostly XML-based) comprising content along with Web links. Web feeds are designed in such a way that is machine readable (computer) instead of human-readable. RSS also contains Xml document that frequently scans the website's content for any update and then displays it to the user through feed. The update that is sent contains a headline and a small amount of text. The text may be a summary or link to the whole text.

Table 15 lists the various versions of RSS:

Table 15: RSS Versions

RSS Version	Description
RSS 0.90	Designed by Netscape for building portals of headlines to news websites. This version is not used nowadays.
RSS 0.91	Proposed by Netscape, but developed by UserLand. Developers use this version for developing Web-based software. This version is called Rich Site Summary and is relatively easy to use. It is commonly used for basic syndication and offers an easy migration path to developers.
RSS 0.92, RSS 0.93, RSS 0.94	Developed by UserLand, this version allows richer metadata than 0.91, but has now been replaced by 2.0.
RSS 1.0	Developed by RSS-DEV Working Group; this format was based on Resource Description Framework (RDF) and is used in applications where the use of advance RDF-specific modules is desired. This ensures extensibility of a website, which can be managed by more than one vendor.
RSS 2.0	Developed by UserLand, this version offers extensibility by using modules and has an easy migration path from the 0.9x branch. It is used for general-purpose and metadata-rich syndication.

Without involving any user activity, RSS update can be received automatically. Some special programs such as Feedreader or aggregator perform the task of receiving update automatically. These programs reduce the time and effort of checking updates of website by gathering the syndicated Web contents such as blogs and news headlines. An aggregator checks and retrieves new content at user-adjusted interval, after getting subscribed to a feed. A website provides the combined view of contents on a single screen or desktop application with the help of RSS readers or aggregators. An RSS document matches with XML 1.0 specifications, as broadcasted on the World Wide Web Consortium (W3C) website.

The syntax of RSS 2.0 is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
```


Scanned by CamScanner