ଓଡ଼ିଶା ରାଜ୍ୟ ମୁକ୍ତ ବିଶ୍ୱବିଦ୍ୟାଳୟ, ସମ୍ବଲପୁର, ଓଡ଼ିଶା
Odisha State Open University, Sambalpur, Odisha
Established by an Act of Government of Odisha.

DIPLOMA IN COMPUTER APPLICATION

DCA-04

WEB -DESIGN

Block

3

JAVA SCRIPT

Unit -5
Getting Started

Unit -6
Advanced Java Script

ଓଡ଼ିଶା ରାଜ୍ୟ ମୁକ୍ତ ବିଶ୍ୱବିଦ୍ୟାଳୟ, ସମ୍ବଲପୁର, ଓଡ଼ିଶା
Odisha State Open University, Sambalpur, Odisha
Established by an Act of Government of Odisha.

EXPERT COMMITTEE

Dr.P.K.Behera **(Chairman)**
Reader in Computer Science
Utkal University , Odisha

DrJ.R.Mohanty **(Member)**
Prof And HOD
KIIT University, Odisha

ShPabitranandaPattnaik **(Member)**
Scientist –E,NIC
NIC, ,BhubaneswarOdisha

Sh Malaya Kumar Das **(Member)**
Scientist –E , NIC
Bhubaneswar , Odisha

Dr.BhagirathiNayak **(Member)**
Professor And Head (IT & System)
Sri Sri University, Odisha

Dr.ManoranjanPradhan **(Member)**
Professor and Head
G.I.T.A
Bhubaneswar, Odisha

Mr.V.S.Sandilya **(Convener)**
Academic Consultant I.T
Odisha State Open University,
Sambalpur,Odisha

DIPLOMA IN COMPUTER APPLICATION

COURSE WRITER

Mr. Dhruba Charan Pradhan
Department of Computer Science,
OAVS, Th.Rampur, Bhabanipatna
Kalahandi, Odisha

Mr. Sushant Kumar Mohanty
Department of Information Technology,
Shailabala Women's College, Cuttack,
Odisha

Unit -5
Getting Started

Learning objectives:

After the Completion of this unit you should be able to know

- The importance of JavaScript
- Features of Java Script
- Writing format in Java Script
- Variable declaration & initialization
- Types of operators
- Conditional statements
- Repetitive statements
- Dialog box in Java Script

Structure

5.1 Introduction

JavaScript is most commonly used as a client side scripting language. This means that JavaScript code is written into an HTML page. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it. JavaScript has nothing to do with Java. JavaScript and Java are completely different languages, both in concept and design. JavaScript programs are run by an interpreter built into the user's web browser (not on the server). It is lightweight interpreted programming language and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. JavaScript is a very free-form language compared to Java.

5.1.1 Definition:

Java Script is a dynamic scripting computer programming language used to make web pages interactive in HTML pages.

5.1.2 History

JavaScript was developed by ***Brendan Eich*** in 1995 at Netscape Corporation (LiveScript), when Eich was working for Netscape Communications Corporation and became an ECMA (European Computer Manufacturers Association) standard in 1997. The technology was first called ***Mocha***, then ***Live Script***. Eventually it was named ***JavaScript*** to follow the marketing of another programming language called Java. Java was developed by Sun Microsystems and is a completely different programming language and technology. JavaScript was a competitive technology to VBScript, a Microsoft product. While VBScript worked only on the Internet Explorer browser, JavaScript was supported on other browsers, too. This made JavaScript a preferred language for global applications.ECMA-262 is the official name. ECMA Script 6 (released in June 2015) is the latest JavaScript version.

5.1.3 Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages.
2. **CSS** to specify the layout of web pages.
3. **JavaScript** to program the behavior of web pages.

5.1.4 What can JavaScript Do?

- JavaScript can dynamically modify an HTML page.
- JavaScript can react to user input.
- JavaScript can validate user input.
- JavaScript can be used to create cookies
- JavaScript is a full-featured programming language
- JavaScript user interaction does not require any communication with the server

5.1.5 Advantages of JavaScript.

The advantages of using JavaScript are:

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

5.1.6 Browser Compatibility.

JavaScript is widely supported. It is available in the following browsers:

- Netscape Navigator (beginning with version 2.0)
- Microsoft Internet Explorer (beginning with version 3.0)
- Firefox
- Safari
- Opera
- Google Chrome

5.2 Java Script syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page. You can place the **<script> tags**, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head> tags**. The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<Script ...>

 JavaScript code

</script>
```

The script tag takes two important attributes:

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

Syntax

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

    JavaScript code

</script>
```

5.3 Enabling Java Script in Browsers

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

5.3.1 JavaScript in Internet Explorer

Here are the steps to turn on or turn off JavaScript in Internet Explorer:

1. Follow **Tools →Internet Options** from the menu.
2. Select **Security** tab from the dialog box.
3. Click the **Custom Level** button.
4. Scroll down till you find the **Scripting** option.
5. Select Enable radio button under **Active scripting**.
6. Finally click **OK** and come out.

Tips: To disable JavaScript support in your Internet Explorer, you need to select Disable radio button under Active scripting.

5.3.2 JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox:

1. Open a new tab → type about: config in the address bar.
2. Then you will find the warning dialog. Select I'll be careful, I promise!
3. Then you will find the list of configure options in the browser.
4. In the search bar, type JavaScript. Enabled.
5. There you will find the option to enable or disable JavaScript by right-clicking on the value of that option -> select toggle.

Tips:If JavaScript. Enabled is true; it converts to false upon clicking toogle. If javascript is disabled; it gets enabled upon clicking toggle.

5.3.3 JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome:

- Click the Chrome menu at the top right hand corner of your browser.
- Select **Settings**.
- Click **Show advanced settings** at the end of the page.
- Under the **Privacy** section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

5.3.4 JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera:

- Follow **Tools-> Preferences** from the menu.
- Select **advanced** option from the dialog box.
- Select **Content** from the listed items.
- Select **Enable JavaScript** checkbox.
- Finally click OK and come out.

Tips: To disable JavaScript support in Opera, you should not select the Enable JavaScript checkbox.

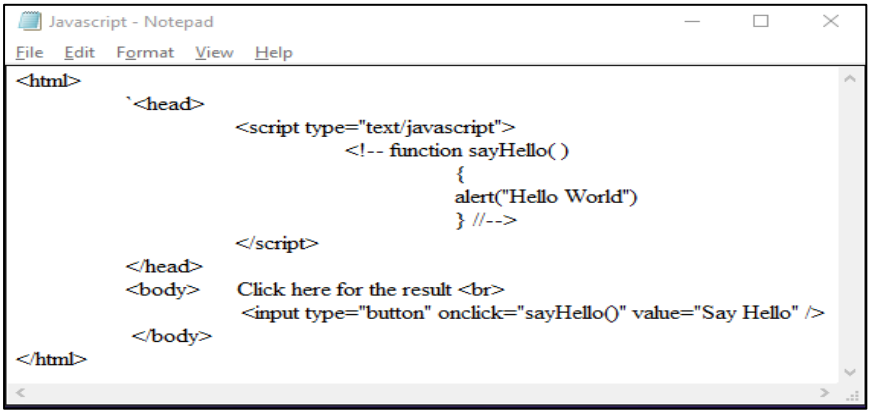
5.4 Placing JavaScript

JavaScript code can be inserted into anywhere in an HTML document by using the SCRIPT tag. You can have any number of scripts. To include JavaScript in an HTML file are as follows:

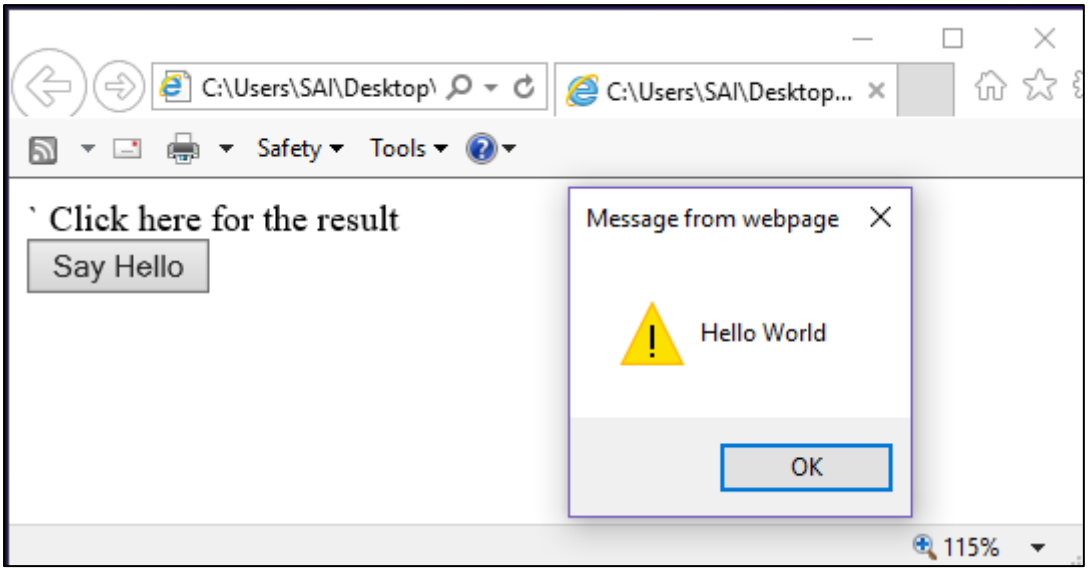
1. Script in <head>...</head> section.
2. Script in <body>...</body> section.
3. Script in <body>...</body> and <head>...</head> sections.
4. Script in an external file and then include in <head>...</head> section.

5.4.1 JavaScript in <head>...</head> Section

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

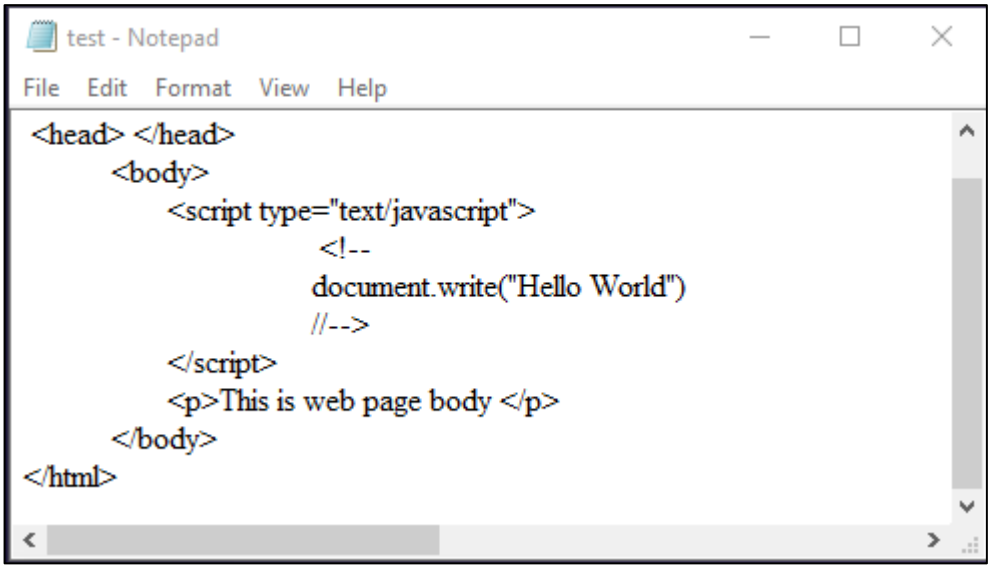


This code will produce the following results:

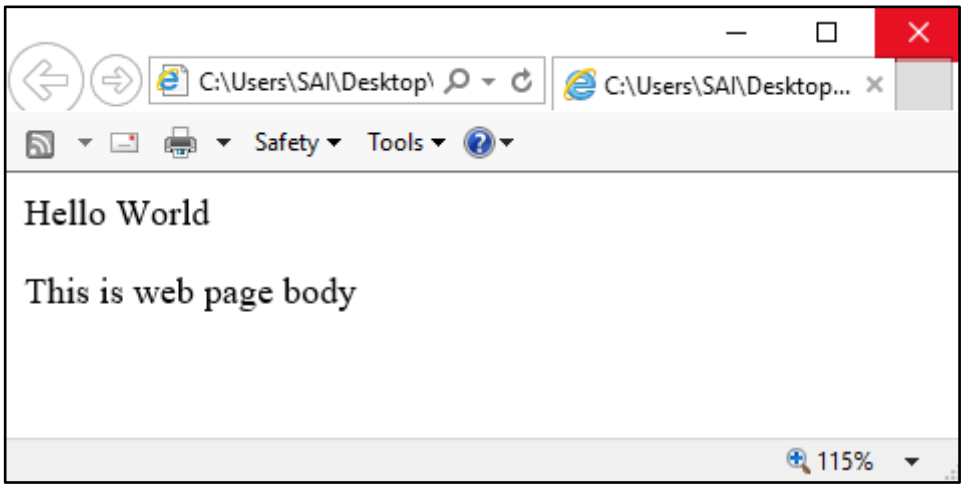


5.4.2 JavaScript in <body>...</body> Section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

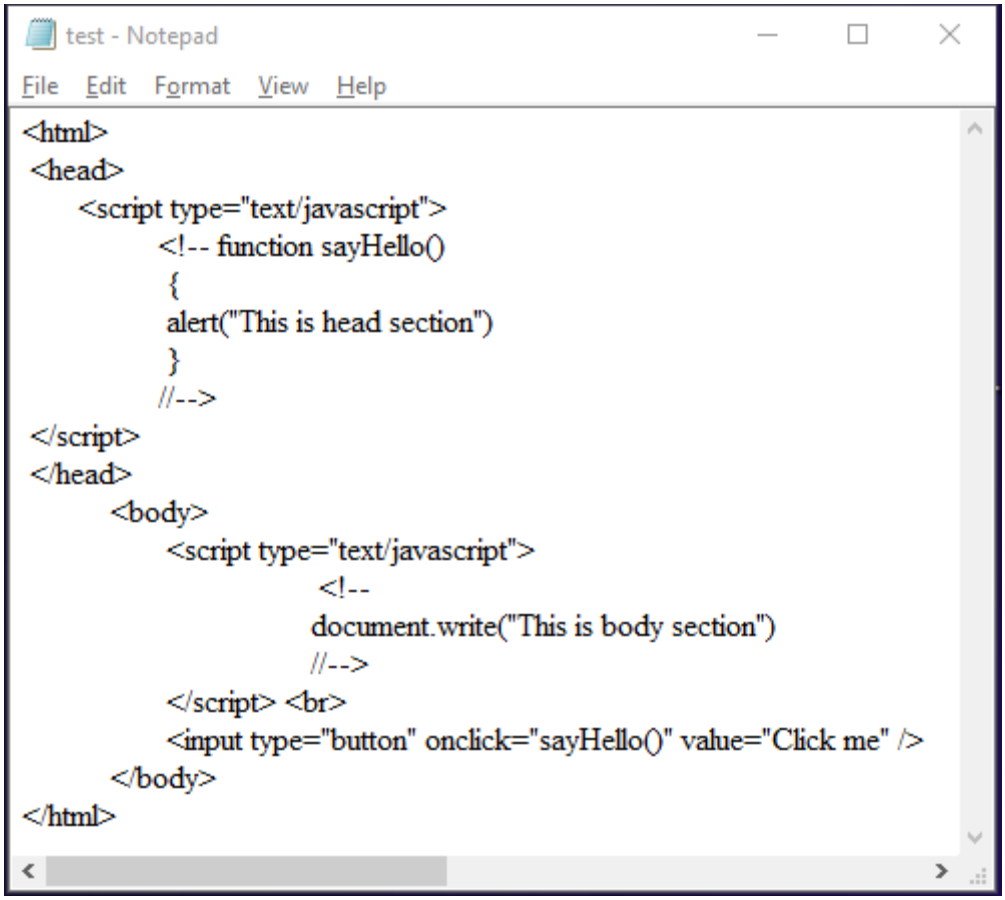


This code will produce the following results:

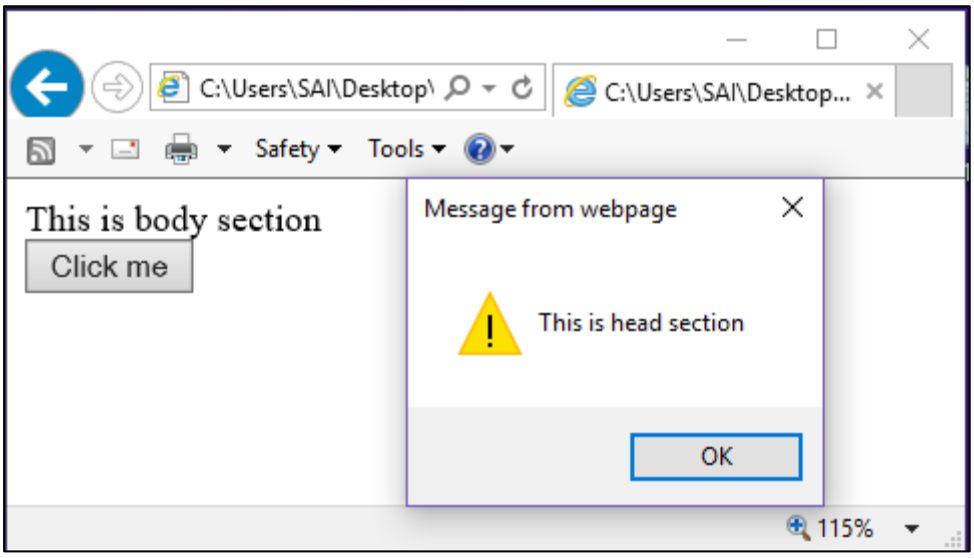


5.4.3 JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows.



This code will produce the following results:



5.4.4 JavaScript in External File

You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files. Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

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

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use sayHello function in your HTML file after including the filename.js file.

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

CHECK YOUR PROGRESS 1

Q1. What is Java Script?

Answer:_____
_____
_____

Q2Who developed java script?

Answer:_____
_____
_____

Q3. What can java script do?

Answer:_____
_____
_____

Q4. Write the syntax of java script ?

Answer:_____
_____
_____

Q5. Write the steps to turn on or turn off JavaScript in Internet Explorer.
Answer:_____
_____
_____

5.5 Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

5.5.1 Definition

It is a quantity whose value can be change during the execution of the program.

5.5.2 Declaration of variable

Variables are declared with the **var** keyword as follows.

```
var money;
var name;
```

You can also declare multiple variables with the same var keyword as follows:

```
var money, name;
```

5.5.3Variable Initialization

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable. For instance, you might create variable named money and assign the value 523.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
var name = "Shan";
var money;
money = 523.50;
```

Tips:Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

5.5.4 JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables**: A global variable has global scope which means it can be defined anywhere in your JavaScript code.

- **Local Variables**: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

  Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
var myVar = "global";                    // Declare a global variable

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

5.6 Operators

5.6.1 Definition

An Operator is a symbol that tells to perform specific operation.

Let us take a simple expression **5 + 3 is equal to 8**. Here 5 and 3 are called **operands** and '+' is called the **operator**.

5.6.2 Types of Operator

JavaScript supports the following types of operators.

1. Arithmetic Operators
2. Comparison Operators
3. Logical (or Relational) Operators
4. Assignment Operators
5. Conditional (or ternary) Operators

Sl No	Operator	Description	Example Let A=5, B=2	Result
1	+	Addition	A + B	7
2	-	Subtraction	A-B	3
3	*	Multiplication	A * B	10
4	/	Division	A /B	2.5
5	%	Modulus (Remainder of an integer division)	A%B	1
6	+ +	Increment (Increases an integer value by one)	A++	6
7	- -	Decrement (Decreases an integer value by one)	A - -	4

5.6.2.1 Arithmetic Operators

JavaScript supports the following arithmetic operators: Assume variable A holds 5 and variable B holds 2, then:

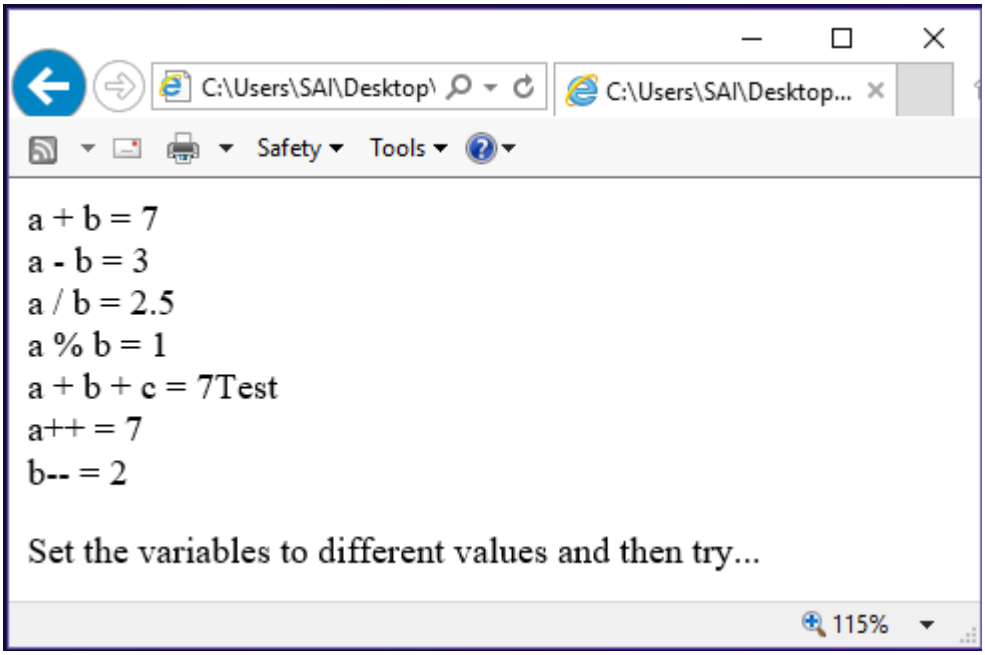
Note: Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 5 will give "a5".

Example

```
<html>
<body>
<script type="text/javascript">
<!--
        var a = 5;
        var b = 2;
        var c = "Test";
        var linebreak = "<br />";
        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);
        document.write("a - b = ");
        result = a - b;
        document.write(result);
        document.write(linebreak);
        document.write("a / b = ");
        result = a / b;
        document.write(result);
        document.write(linebreak);
        document.write("a % b = ");
        result = a % b;
        document.write(result);
        document.write(linebreak);
        document.write("a + b + c = ");
        result = a + b + c;
        document.write(result);
        document.write(linebreak);
        a = a++;
        document.write("a++ = ");
        result = a++;
        document.write(result);
        document.write(linebreak);
        b = b--;
        document.write("b-- = ");
        result = b--;
        document.write(result);
        document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and then try...</p>
```

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

This code will produce the following results:



5.6.2.2 Comparison Operators

JavaScript supports the following comparison operators: Assume variable A holds 10 and variable B holds 20, then:

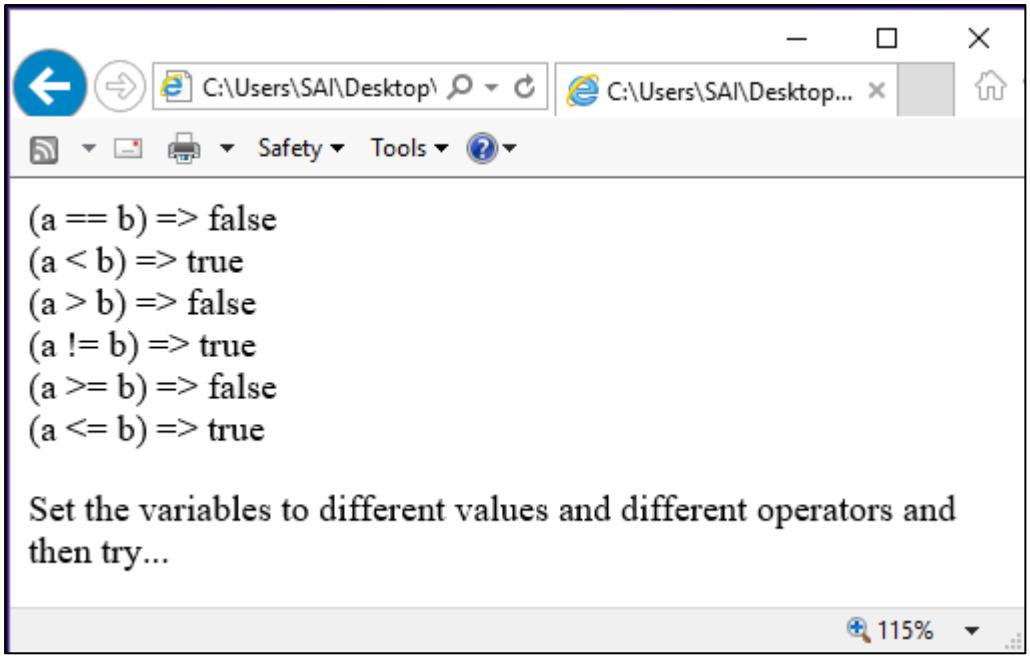
Sl No	**Operator**	**Description**	**Example Let A= 10, B= 20**	**Result**
1	==	**Equal** Checks if the values of two operands are equal or not, if yes, then the condition becomes true.	A==B	False
2	!=	**Not Equal** Checks if the values of two operands are equal or not, if the values are not equal, then the condition becomes true.	A != B	True

3	>	**Greater than** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.	A>B	False
4	>=	**Greater than or Equal to** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.	A>=B	False
5	<	**Less than** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.	A<B	True
6	<=	**Less than or Equal to** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.	A<=B	True

Example

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

                document.write(result);
                document.write(linebreak);
                document.write("(a < b) => ");
                result = (a < b);

                document.write(result);
                document.write(linebreak);
                document.write("(a > b) => ");
                result = (a > b);

                document.write(result);
                document.write(linebreak);
                document.write("(a != b) => ");
                result = (a != b);

                document.write(result);
                document.write(linebreak);
                document.write("(a >= b) => ");
                result = (a >= b);

                document.write(result);
                document.write(linebreak);
                document.write("(a <= b) => ");
                result = (a <= b);

                document.write(result);
                document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

This code will produce the following results:



5.6.2.3 Logical (or Relational) Operators

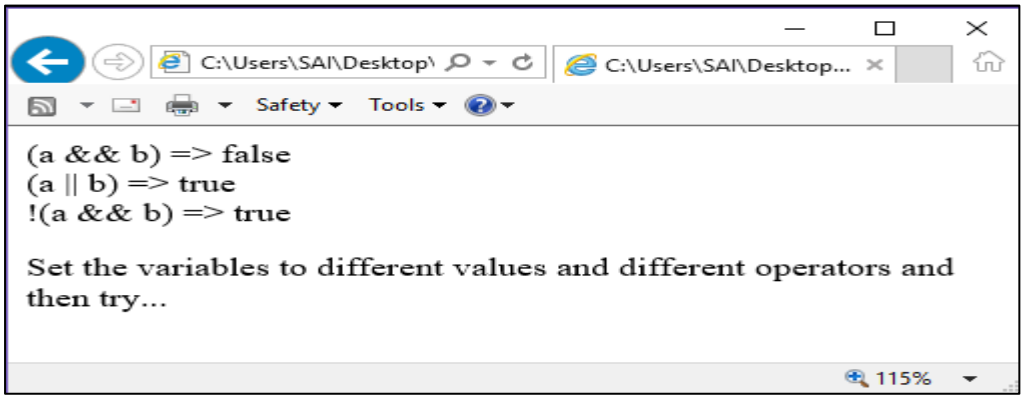
JavaScript supports the following logical operators: Assume variable A holds 5 and variable B holds 2, then:

Sl No	Operator	Description
1	&&	**Logical AND** If both the operands are non-zero, then the condition becomes true.
2	||	**Logical OR** If any of the two operands are non-zero, then the condition becomes true.
3	!	**Logical NOT** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Example

```
<html>
<body>
<script type="text/javascript">
<!--
        var a = true;
        var b = false;
        varlinebreak = "<br />";
        document.write("(a &&  b) => ");
        result = (a && b);
        document.write(result);
        document.write(linebreak);
        document.write("(a || b) => ");
        result = (a || b);
        document.write(result);
        document.write(linebreak);
        document.write("!(a && b) => ");
        result = (!(a && b));
        document.write(result);
        document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

This code will produce the following results:



5.6.2.4 Assignment Operators

JavaScript supports the following assignment operators:

Sl No	Operator	Description
1	=	**Simple Assignment** Assigns values from the right side operand to the left side operand Ex: C = A + B will assign the value of A + B into C
2	+=	**Add and Assignment** It adds the right operand to the left operand and assigns the result to the left operand. Ex: C += A is equivalent to C = C + A
3	-=	**Subtract and Assignment** It subtracts the right operand from the left operand and assigns the result to the left operand. Ex: C -= A is equivalent to C = C - A
4	*=	**Multiply and Assignment** It multiplies the right operand with the left operand and assigns the result to the left operand. Ex: C *= A is equivalent to C = C * A
5	/=	**Divide and Assignment**) It divides the left operand with the right operand and assigns the result to the left operand. Ex: C /= A is equivalent to C = C / A
6	%=	Modules and Assignment It takes modulus using two operands and assigns the result to the left operand. Ex: C %= A is equivalent to C = C % A

Note: Same logic applies to Bitwise operators, so they will become <<=, >>=, >>=, &=, |= and ^=.

5.6.2.5 Conditional (or ternary) Operator

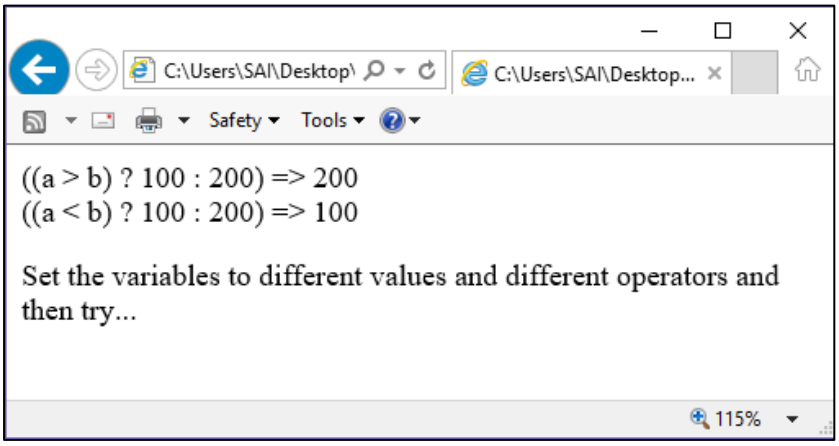
The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sl No	Operator	Description
1	? :	**Conditional** If Condition is true? Then value X : Otherwise value Y

Example

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";
document.write ("((a > b) ?100 : 200) => "); result = (a > b) ? 100 : 200;
document.write(result);
document.write(linebreak);
document.write ("((a < b) ?100 : 200) => ");
result = (a < b) ? 100 : 200;
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

This code will produce the following results:



CHECK YOUR PROGRESS 2

Q1. What is Variable?

Answer:_____
_____
_____

Q2. What is Operator?

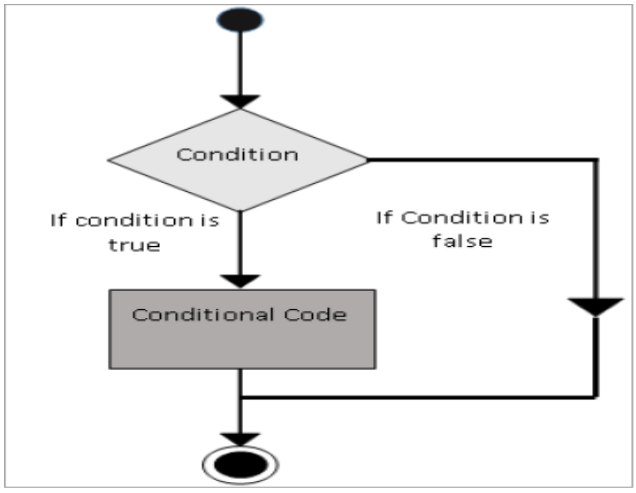
Answer:_____
_____
_____

5.7 IF….ELSE

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

5.7.1 Flow Chart of if-else

The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of if…...else statement:

1. if statement
2. if...else statement
3. if...else if... statement

5.7.2 if Statement

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

Syntax

The syntax for a basic if statement is as follows:

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

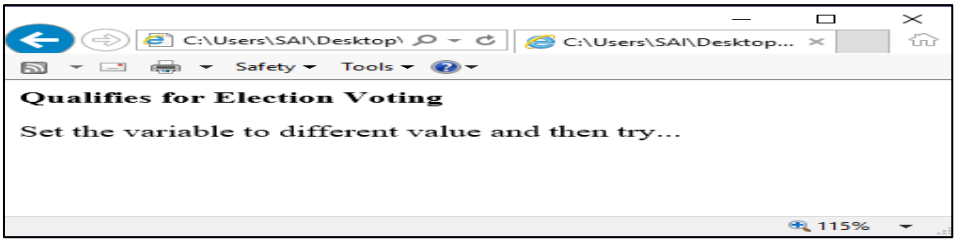
Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

The following example to understand how the if statement works.

```
<html>
<body>
<script type="text/javascript">
<!--
        var age = 20;
        if( age >=18 ){
        document.write("<b>Qualifies for Election Voting</b>");}
//-->
</script>
        <p>Set the variable to different value and then try...</p>
</body>
</html>
```

This code will produce the following results:



5.7.3 if..else Statement

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

Syntax

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

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

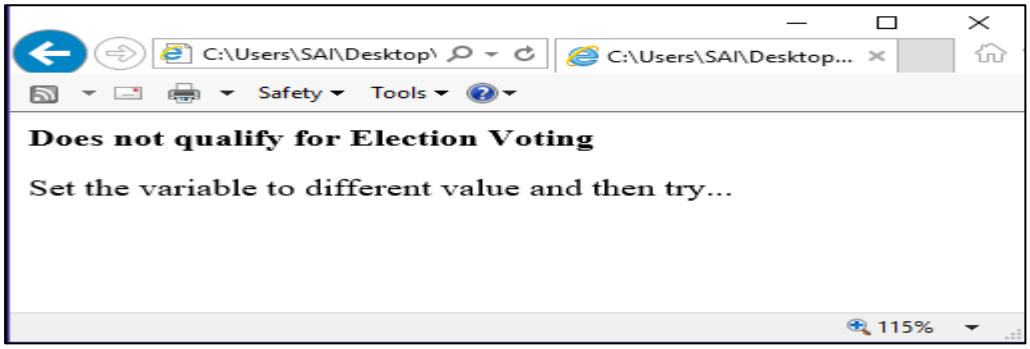
Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

The following code to learn how to implement an if-else statement in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var age = 15;
if( age >= 18 )
        {
        document.write("<b>Qualifies for Election Voting</b>"); }else{
        document.write("<b>Does not qualify for Election Voting</b>");
        }
//-->
</script>
        <p>Set the variable to different value and then try...</p>
</body>
</html>
```

This code will produce the following results:



5.7.4 if….else….if Statement

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

Syntax

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

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

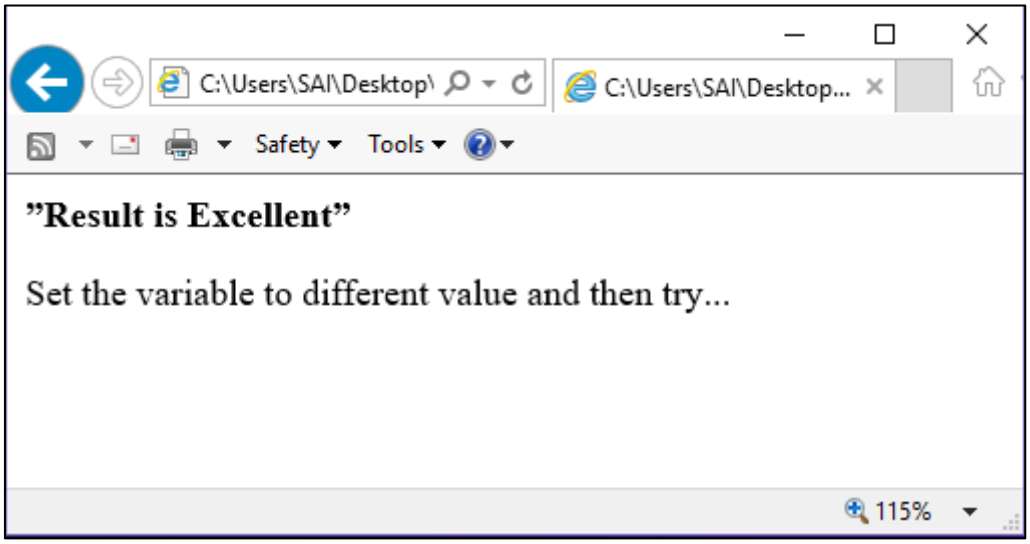
There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

The following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
        var mark = 92;
        if(mark >= 90 ){
        document.write("<b>"Result is Excellent"</b>"); }
        else if( mark >=60 ){
        document.write("<b>"Result is 1st Class"</b>"); }
        else if( mark>=50 ){
        document.write("<b>"Result is 2nd Class"</b>"); }
        else if( mark>=40 ){
        document.write("<b>"Result is 3rd Class"</b>"); }
        else{
        document.write("<b>Fail</b>");}
//-->
</script>
        <p>set the variable to different value and then try...</p>
</body>
</html>
```

This code will produce the following results:

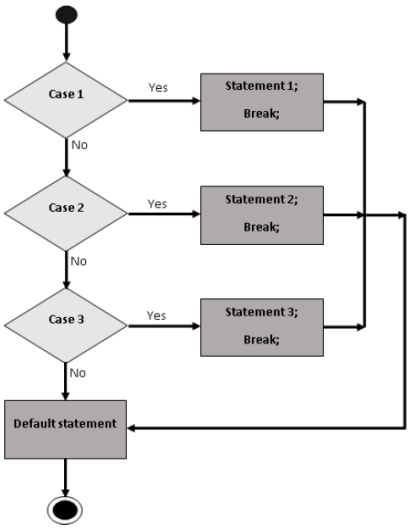


5.8 Switch Case

You can use multiple if...else…if statements, as in the previous chapter, to perform a multi way branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable. Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

5.8.1 Flow Chart

The following flow chart explains a switch-case statement works.



5.8.2 Syntax

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

```
switch (expression)
{
case condition 1: statement(s)
break;
case condition 2: statement(s)
break;
...
case condition n: statement(s)
break;
default: statement(s)
        }
```

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

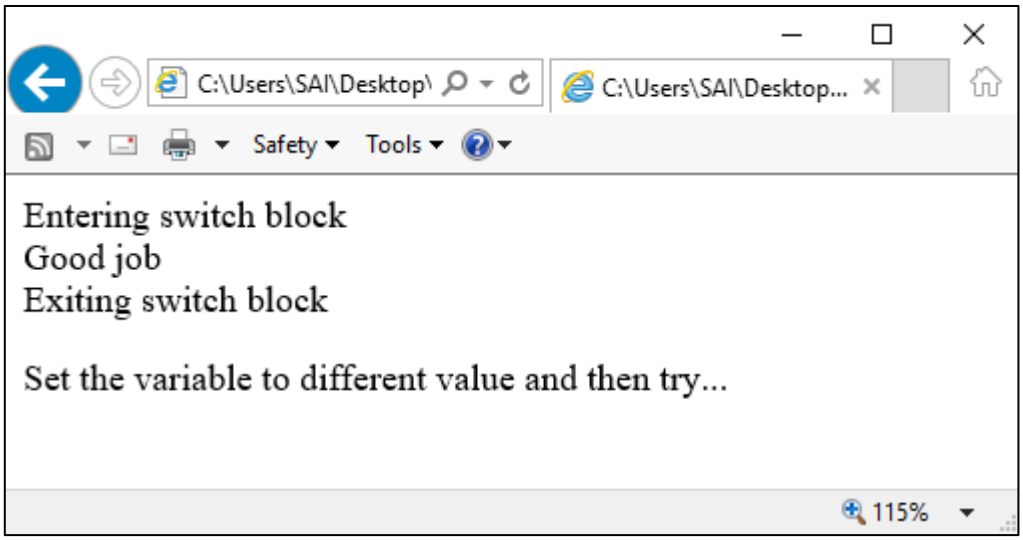
Example

The following example to implement switch-case statement.

```
<html>
<body>
    <script type="text/javascript">
    <!--
        var grade='A';
        document.write("Entering switch block<br />");
        switch (grade)
        {
        case 'A': document.write("Good job<br />");
        break;
        case 'B': document.write("Pretty good<br />");
        break;
        case 'C': document.write("Passed<br />");
        break;
        case 'D': document.write("Not so good<br />");
        break;
        case 'F': document.write("Failed<br />");
        break;
        default: document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
    //-->
    </script>
        <p>Set the variable to different value and then try...</p>
```

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

This code will produce the following results:



5.9 Loops

Iterative statements, also called loop statements, specify certain commands to be executed repeatedly until some condition is met. The loops are often used to iterate the values of an array (hence the name) or to work though repetitious mathematical tasks. It is a command that execute again and again till condition fulfill.

5.9.1 Types of loop

There are different types of loop used in java script . Some of the loop are :

1. do…. While loop
2. while loop
3. for loop

5.9.2 Do----while loop

The do-while statement is a post-test loop, meaning that the evaluation of the escape condition is only done after the code inside the loop has been executed. This means that the body of the loop is always executed at least once before the expression is evaluated.

Syntax:

do {

statement

} while (expression);

For example:

```
var i = 0;
do {
i += 2;
} while (i < 10);
```

5.9.3 While loop

The while statement is a pretest loop. This means the evaluation of the escape condition is done before the code inside the loop has been executed. Because of this, it is possible that the body of the loop is never executed.

Syntax:

while(expression) statement

For example:

```
var i = 0; while (i < 10) {
i += 2;
}
```

5.9.4 For loop

The *for statement* is also a pretest loop with the added capabilities of variable initialization before entering the loop and defining post loop code to be entered.

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

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

You can put all the three parts in a single line separated by semicolons.

Syntax:

for (initialization; expression; post-loop-expression) statement

For example:

```
for (var i=0; i <iCount; i++){

alert(i);

}
```

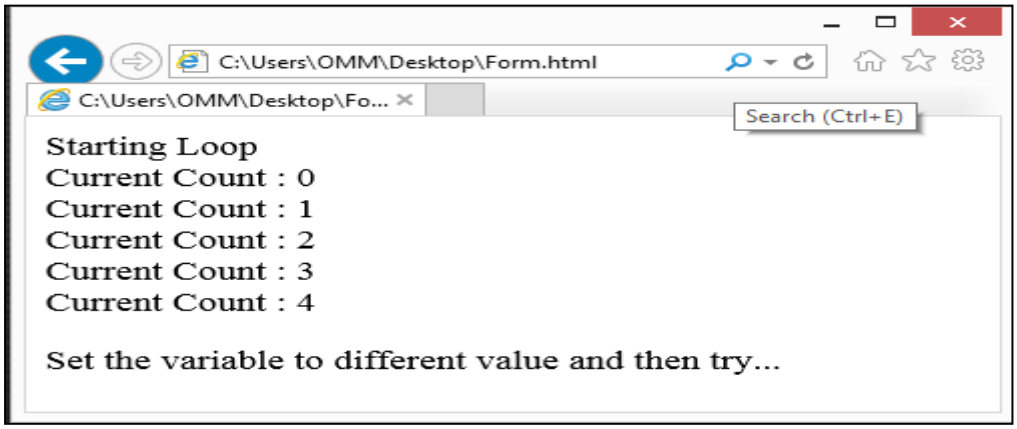
This code defines a variable i that begins with the value 0. The for loop is entered only if the conditional expression (i <iCount) evaluates to true, making it possible that the body of the code might not be executed. If the body is executed, the postloop expression is also executed, iterating the variable i.

Example

The following example to learn how a **for** loop works in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 5; count++){
document.write("Current Count : " + count );
document.write("<br />");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output



5.10 Functions

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

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

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

5.10.1 Function Definition

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

Syntax

The basic syntax is shown here.

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

Example

The following example. It defines a function called sayHello that takes no parameters:

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

5.10.2 Calling a Function

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

```
<html>
<head>
<script type="text/javascript"> function sayHello()
{
document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello"> </form>
<p>Use different text in write method and then try...</p> </body>
</html>
```

Output

5.10.3 Function Parameters

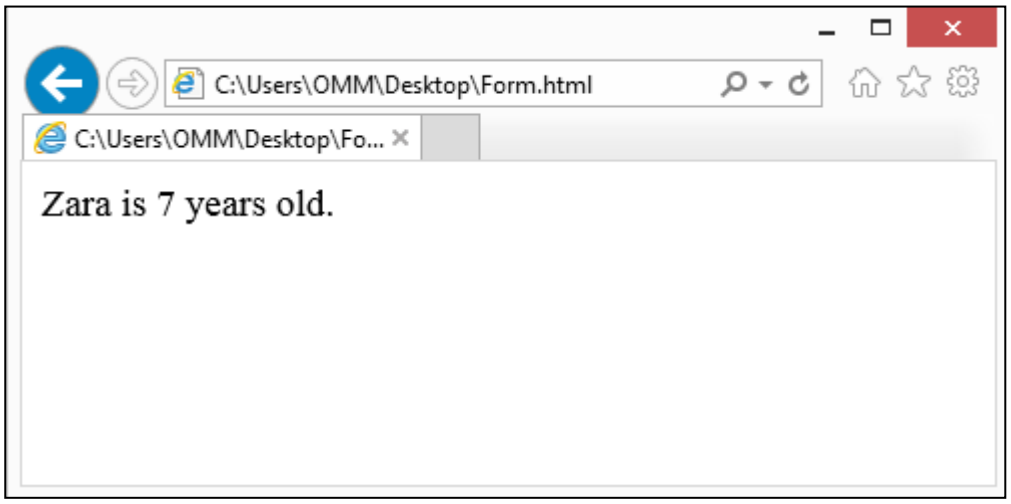
Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

The following example. We have modified our sayHello function here. Now it takes two parameters.

```
<html>
<head>
<script type="text/javascript"> function sayHello(name, age)
{
document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output



5.10.4 The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
<head>
<script type="text/javascript">  function concatenate(first, last) {
var full;
 full = first + last;  return  full;
}
function secondFunction() {
var result;
 result = concatenate('Zara', 'Ali');  document.write (result );
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

5.10.5 Nested Functions

Prior to JavaScript 1.2, function definition was allowed only in top level global code, but JavaScript 1.2 allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the function statement.

Example

The following example to learn how to implement nested functions.

```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
function square(x) { return x*x; }
return Math.sqrt(square(a) + square(b));
}
function secondFunction(){
var result;
result = hypotenuse(1,2);
document.write ( result );
}
//-->
</script>
</head>
```

```
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

5.11 Events and Event handling

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc. Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable. Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Here we will see a few examples to understand the relation between Event and JavaScript.

5.11.1 Onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

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

</head>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Output

5.11.2 onsubmitEvent Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

The following example.

```
<html>
<head>
<script type="text/javascript">
<!--
function validation() {
all validation goes here
.........
return either true or false
}
//-->
</script>
</head>
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
.......
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

5.11.3 onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<html>
<head>
<script type="text/javascript">
<!--
function over() {
document.write ("Mouse Over");
}
function out() {
document.write ("Mouse Out");
}
//-->
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the result:</p>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

5.11.4 HTML5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
onblur	script	Triggers when the window loses focus
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click

oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged
ondragend	script	Triggers at the end of a drag operation
ondragenter	script	Triggers when an element has been dragged to a valid drop target
ondragleave	script	Triggers when an element leaves a valid drop target
ondragover	script	Triggers when an element is being dragged over a valid drop target
ondragstart	script	Triggers at the start of a drag operation
ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
onemptied	script	Triggers when a media resource element suddenly becomes empty.
onended	script	Triggers when media has reach the end
onerror	script	Triggers when an error occur
onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes
onforminput	script	Triggers when a form gets user input
onhaschange	script	Triggers when the document has changed
oninput	script	Triggers when an element gets user input
oninvalid	script	Triggers when an element is invalid
onkeydown	script	Triggers when a key is pressed
onkeypress	script	Triggers when a key is pressed and released
onkeyup	script	Triggers when a key is released
oncontextmenu	script	Triggers when a context menu is triggered

onload	script	Triggers when the document loads
onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
onloadstart	script	Triggers when the browser starts to load the media data
onmessage	script	Triggers when the message is triggered
onmousedown	script	Triggers when a mouse button is pressed
onmousemove	script	Triggers when the mouse pointer moves
onmouseout	script	Triggers when the mouse pointer moves out of an element
onmouseover	script	Triggers when the mouse pointer moves over an element
onmouseup	script	Triggers when a mouse button is released
onmousewheel	script	Triggers when the mouse wheel is being rotated
onoffline	script	Triggers when the document goes offline
onoine	script	Triggers when the document comes online
ononline	script	Triggers when the document comes online
onpagehide	script	Triggers when the window is hidden
onpageshow	script	Triggers when the window becomes visible
onpause	script	Triggers when media data is paused
onplay	script	Triggers when media data is going to start playing

5.12 Cookies

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

5.12.1 How It Works?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.
Cookies are a plain text data record of 5 variable-length fields:

- **Expires:** The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain:** The domain name of your site.
- **Path:** The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure:** If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value:** Cookies are set and retrieved in the form of key-value pairs.

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the **cookie** property of the **Document** object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

5.12.2 Storing Cookies

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this.

document.cookie = "key1=value1;key2=value2;expires=date";

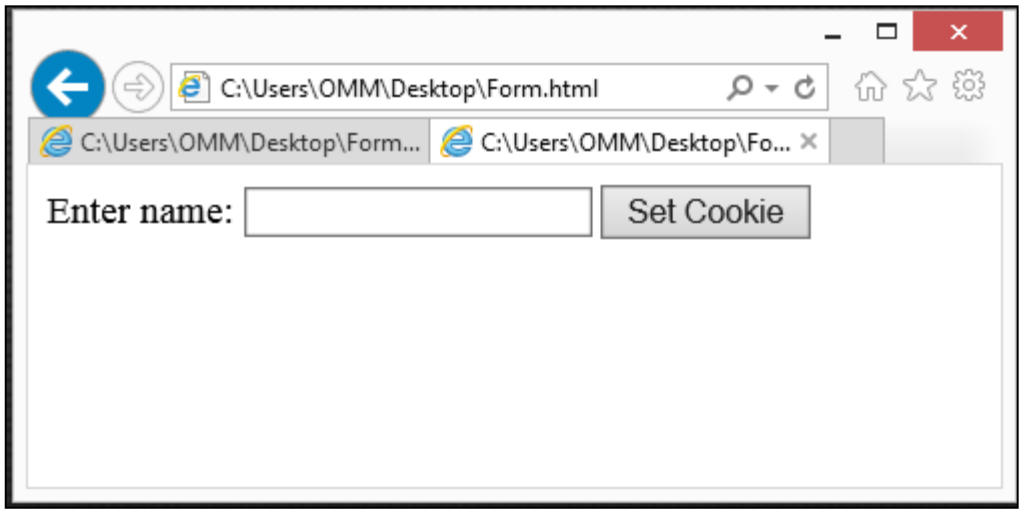
Here the **expires** attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

Note: Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript **escape()** function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding **unescape()** function when you read the cookie value.

Example
The following. It sets a customer name in an input cookie.

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
if( document.myform.customer.value == "" ){
alert ("Enter some value!");
return;
}
cookievalue= escape(document.myform.customer.value) + ";";
document.cookie="name=" + cookievalue;
document.write ("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>

</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie();"/>
</form>
</body>
</html>
```

Output



Now your machine has a cookie called **name**. You can set multiple cookies using multiple key=value pairs separated by comma.

5.12.3 Reading Cookies

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The document.cookie string will keep a list of name=value pairs separated by semicolons, where **name** is the name of a cookie and value is its string value.
You can use strings' **split()** function to break a string into key and values as follows:

Example
The following example to get all the cookies.

```
<html>
<head>
<script type="text/javascript">
<!--
function ReadCookie()
{
var allcookies = document.cookie;
document.write ("All Cookies : " + allcookies );

// Get all the cookies pairs in an array
cookiearray = allcookies.split(';');
// Now take key value pair out of this array
for(var i=0; i<cookiearray.length; i++){
name = cookiearray[i].split('=')[0];
value = cookiearray[i].split('=')[1];
document.write ("Key is : " + name + " and Value is : " + value);
}
}
//-->
</script>
</head>
<body>
<form name="myform" action="">
<p> click the following button and see the result:</p>
<input type="button" value="Get Cookie" onclick="ReadCookie()"/>
</form>
</body>
</html>
```

Note: Here **length** is a method of **Array** class which returns the length of an array. We will discuss Arrays in a separate chapter. By that time, please try to digest it.

Note: There may be some other cookies already set on your machine. The above code will display all the cookies set on your machine.

5.12.4 Setting Cookies Expiry Date

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the '**expires**' attribute to a date and time.

Example

The following example. It illustrates how to extend the expiry date of a cookie by 1 Month.

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
var now = new Date();
now.setMonth( now.getMonth() + 1 );
cookievalue = escape(document.myform.customer.value) + ";"
document.cookie="name=" + cookievalue;
document.cookie = "expires=" + now.toUTCString() + ";"
document.write ("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

5.12.5 Deleting Cookies

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

Example

The following example. It illustrates how to delete a cookie by setting its expiry date to one month behind the current date.

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

```
function WriteCookie()
{
var now = new Date();
now.setMonth( now.getMonth() - 1 );
cookievalue = escape(document.myform.customer.value) + ";"
document.cookie="name=" + cookievalue;
document.cookie = "expires=" + now.toUTCString() + ";"
document.write("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

5.13 Page Redirection

You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to **page redirection**. This concept is different from JavaScript Page Refresh.

There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons:

- You did not like the name of your domain and you are moving to a new one. In such a scenario, you may want to direct all your visitors to the new site. Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.
- You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.
- The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

5.13.1 JavaScript Page Refresh

You can refresh a web page using JavaScript **location.reload** method. This code can be called automatically upon an event or simply when the user clicks on a link. If you want to refresh a web page using a mouse click, then you can use the following code:

```
<a href="javascript:location.reload(true)">Refresh Page</a>
```

5.13.2 Auto Refresh

You can also use JavaScript to refresh the page automatically after a given time period. Here **setTimeout()** is a built-in JavaScript function which can be used to execute another function after a given time interval.

Example

The following example. It shows how to refresh a page after every 5 seconds. You can change this time as per your requirement.

```
<html>
<head>
<script type="text/JavaScript">
<!--
function AutoRefresh( t ) {
setTimeout("location.reload(true);", t);
}
// -->
</script>
</head>
<body onload="JavaScript:AutoRefresh(5000);">
<p>This page will refresh every 5 seconds.</p>
</body>
</html>
```

5.13.3 How Page Re-direction Works?

The implementations of Page-Redirection are as follows.

Example 1

It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```
<html>
<head>
<script type="text/javascript">
<!--
function Redirect() {
window.location="http://www.tutorialspoint.com";
}
//-->
</script>
</head>
<body>
<p>Click the following button, you will be redirected to home page.</p>
<form>
<input type="button" value="Redirect Me" onclick="Redirect();" />
</form>
</body>
</html>
```

Example 2

You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. The following example shows how to implement the same. Here **setTimeout()** is a built-in JavaScript function which can be used to execute another function after a given time interval.

```
<html>
<head>
<script type="text/javascript">
<!--
function Redirect() {
window.location="http://www.tutorialspoint.com";
}
document.write ("You will be redirected to our main page in 10 seconds!");

setTimeout('Redirect()', 10000);
//-->
</script>
</head>
<body>
</body>
</html>
```

Example 3

The following example shows how to redirect your site visitors onto a different page based on their browsers.

```
<html>
<head>
<script type="text/javascript">
<!--
var browsername=navigator.appName;
if( browsername == "Netscape" )
{
window.location="http://www.location.com/ns.htm";
}
else if ( browsername =="Microsoft Internet Explorer")
{
window.location="http://www.location.com/ie.htm";
}
else
{
window.location="http://www.location.com/other.htm";
}
//-->
</script>
</head>
<body> </body></html>
```

5.14 Dialogs

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

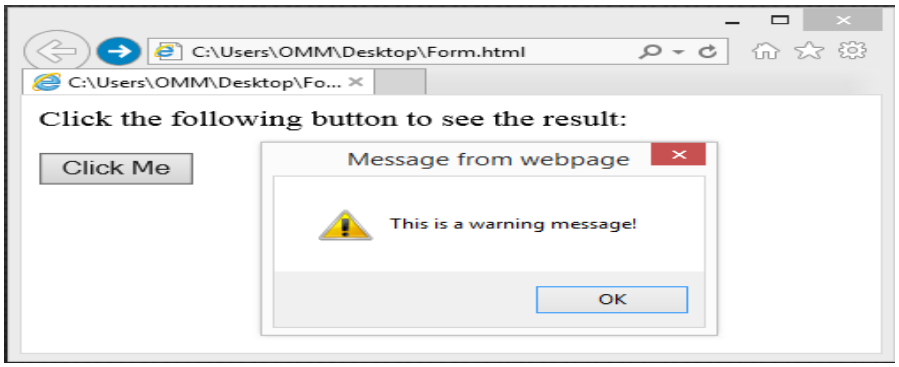
5.14.1 Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.
Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Example

```
<html>
<head>
<script type="text/javascript">
<!--
function Warn() {
alert ("This is a warning message!");
document.write ("This is a warning message!");
}
//-->
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="Warn();" />
</form>
</body>
</html>
```

Output



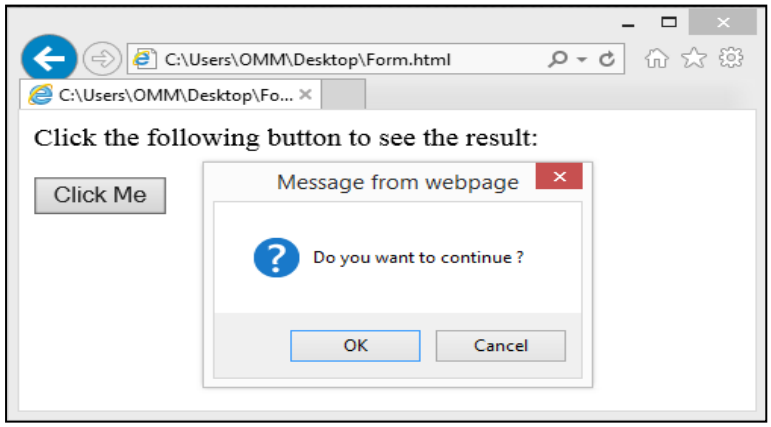
5.14.2 Confirmation Dialog Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: **OK** and **Cancel**.
If the user clicks on the OK button, the window method **confirm()** will return true. If the user clicks on the Cancel button, then **confirm()** returns false. You can use a confirmation dialog box as follows.

Example

```
<html>
<head>
<script type="text/javascript">
<!--
function getConfirmation(){
var retVal = confirm("Do you want to continue ?");
if( retVal == true ){
document.write ("User wants to continue!");
return true;
}else{
Document.write ("User does not want to continue!");
return false;
}
}
//-->
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="getConfirmation();" />
</form>
</body>
</html>
```

Output



5.14.3 Prompt Dialog Box

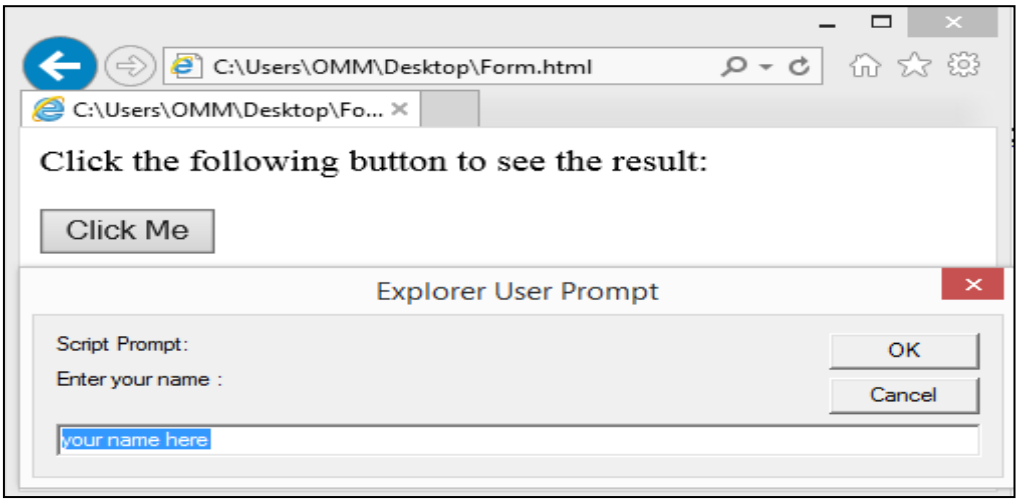
The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK. This dialog box is displayed using a method called **prompt()** which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box. This dialog box has two buttons: **OK** and **Cancel**. If the user clicks the OK button, the window method **prompt()** will return the entered value from the text box. If the user clicks the Cancel button, the window method **prompt()** returns **null**.

Example

The following example shows how to use a prompt dialog box:

```
<html>
<head>
<script type="text/javascript">
<!--
function getValue(){
var retVal = prompt("Enter your name : ", "your name here");
document.write("You have entered : " + retVal);
}
//-->
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="getValue();" />
</form>
</body>
</html>
```

Output



5.15 Void Keyword

Void is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

Syntax

The syntax of **void** can be either of the following two:

```
<head>
<script type="text/javascript">
<!--
void func()
javascript:void func()
OR
void(func())
javascript:void(func())
//-->
</script>
</head>
```

Example 1

The most common use of this operator is in a client-side *javascript: URL*, where it allows you to evaluate an expression for its side-effects without the browser displaying the value of the evaluated expression. Here the expression **alert ('Warning!!!')** is evaluated but it is not loaded back into the current document:

```
<html>
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<p>Click the following, This won't react at all...</p>
<a href="javascript:void(document.write("Hello : 0"))">Click me!</a>
</body>
</html>
```

Example 2

Take a look at the following example. The following link does nothing because the expression "0" has no effect in JavaScript. Here the expression "0" is evaluated, but it is not loaded back into the current document.

```
<html>
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<p>Click the following, This won't react at all...</p>
<a href="javascript:void(0)">Click me!</a>
</body> </html>
```

5.16 Printing WebPages using Java Script

Many times you would like to place a button on your webpage to print the content of that web page via an actual printer. JavaScript helps you to implement this functionality using the **print** function of **window** object. The JavaScript print function **window.print()** prints the current web page when executed. You can call this function directly using the **onclick** event as shown in the following example.
Let us create a file named **my_test.html**

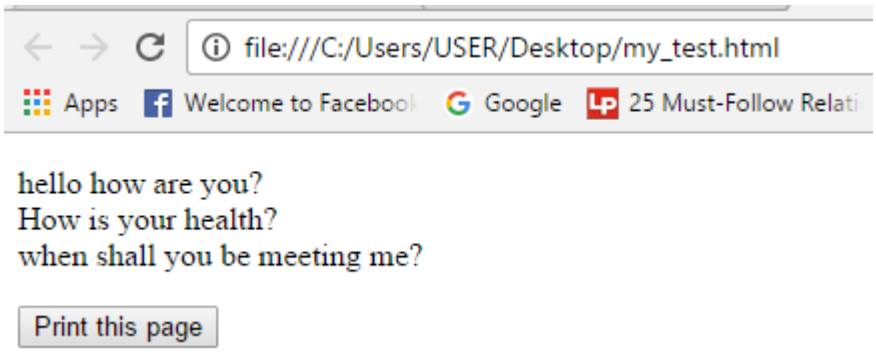
Example

The following example.

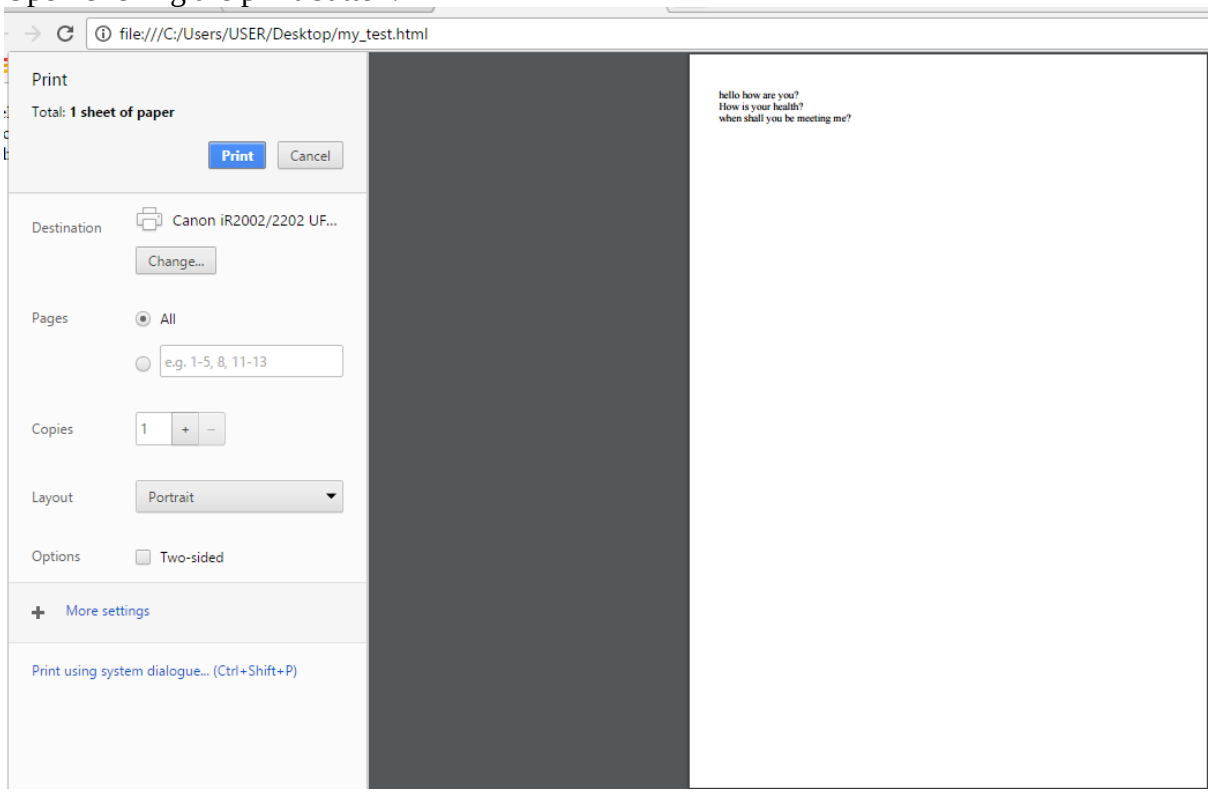
```
<!DOCTYPE html>
<html>
<body>

<p> hello  how are you?<br>
How is your health?<br>
when shall you be meeting me?<br>
</p>

<button id ='togglee' onclick="myFunction()">Print this page</button>

<script>
function myFunction() {
   var hidden = false;
      hidden = !hidden;
      if(hidden) {
         document.getElementById('togglee').style.visibility = 'hidden';
         window.print();
         window.location="my_test.html";
      } else {
         document.getElementById('togglee').style.visibility = 'visible';
      }
}
</script>

</body>
</html>
```

Output

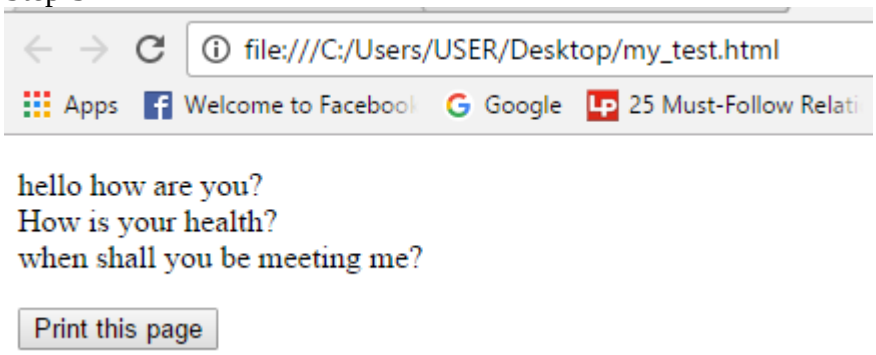
Step-1



Step-2
Upon clicking the print button.



When we click the print button the page is printed.
Step-3



CHECK YOUR PROGRESS 3

Q1. What is a Loop ?

Answer:_____
_____

Q2. What is a Function?

Answer:_____
_____

Q3. What is Cookies?

Answer:_____
_____

Q4. What is auto refresh?

Answer:_____
_____

Q5. Which function is used to print the current webpage?

Answer:_____
_____

5.17 Let us sum up

We have understood what is Javascript, its role and advantages. We came to know how to use JavaScript in web applictions. We learn about variables, operators, if else statements, switch case, loops, functions, events and event handling, cookies, Page redirecton, managing dialog boxes,void keyword, printing webpage using JavaScript

5.18 Reference

1. Java script Bible Gold Edition

2. www. Google.com

3.www. tutorialpoints.com

4. Wrox: Professional Java Script

5. Java wiley Java script Bible

5.19 Check your progress – possible answers

Answers to check your progress 1

Q1. What is Java Script?

Answer: Java Script is a dynamic scripting computer programming language used to make web pages interactive in HTML pages.

Q2 Who developed java script?
Answer: JavaScript was developed by *Brendan Eich* in 1995 at Netscape Corporation
Q3. What can java script do?
Answer:

- JavaScript can dynamically modify an HTML page.
- JavaScript can react to user input.
- JavaScript can validate user input.
- JavaScript can be used to create cookies
- JavaScript is a full-featured programming language
- JavaScript user interaction does not require any communication with the server

Q4. Write the syntax of java script ?
Answer:

A simple syntax of your JavaScript will appear as follows.

<*Script ...*>

JavaScript code

</script>

Or

<script language="javascript" type="text/javascript">

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

Q5. Write the steps to turn on or turn off JavaScript in Internet Explorer.
Answer:

Here are the steps to turn on or turn off JavaScript in Internet Explorer:

1. Follow **Tools →InternetOptions** from the menu.
2. Select **Security** tab from the dialog box.
3. Click the **Custom Level** button.
4. Scroll down till you find the **Scripting** option.
5. Select Enable radio button under **Active scripting**.
6. Finally click **OK** and come out.

Answers to check your progress 2

Q1. What is a Variable?

Answer: It is a quantity whose value can be changed during the execution of the program. It can be declare using *var* keyword.

Q2. What is an Operator?

Answer : *An Operator is a symbol that tells to perform specific operation*.

Let us take a simple expression **5 + 3 is equal to 8**. Here 5 and 3 are called **operands** and '+' is called the **operator**.

Answers to check your progress 3

Q1. What is a Loop ?

Answer:

Iterative statements, also called loop statements, specify certain commands to be executed repeatedly until some condition is met. Or A statement that execute again and again till condition fulfill.

Q2. What is a Function?

Answer:

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

Q3. What is Cookies?

Answer:

Cookies are small files which are stored on a user's computer. They are designed to hold a modest amount of data specific to a particular client and website, and can be accessed either by the web server or the client computer. This allows the server to deliver a page tailored to a particular user, or the page itself can contain some script which is aware of the data in the cookie and so is able to carry information from one visit to the website (or related site) to the next.

Q4. What is auto refresh?

Answer:

Auto Refresh is a simple extension that **refreshes** a page automatically on a given interval. There are many extensions to do this but this one is the most straight forward, and easiest to use while still allowing for customization!

Q5. Which function is used to print the current webpage?

Answer:

The JavaScript print function **window.print()** prints the current web page when executed.

Unit -6
Advanced Java Script

Learning objectives:

After the Completion of this unit you should be able to know

- In JavaScript, almost "everything" is an object.
- About object , properties and method
- The features of object oriented programming
- Mathematical functions
- String and Boolean methods
- Error and exception handling
- Animation, plug-ins
- Different shapes of image map

Structure

6.1 Working with objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers:

- **Encapsulation:** the capability to store related information, whether data or methods, together in an object.
- **Aggregation:** the capability to store one object inside another object.
- **Inheritance:** the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism:** the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object; otherwise the attribute is considered a property.

6.1.1 Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page. The syntax for adding a property to an object is:

objectName.objectProperty = propertyValue;

Example:

The following code gets the document title using the "**title**" property of the **document** object.

var str = document.title;

6.1.2 Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword. Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

Example:

Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

document. write ("This is test");

6.1.3 User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

6.1.4 The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method. In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

6.1.5 The Object ( ) Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object( )** constructor is assigned to a variable. The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example 1

The following example; it demonstrates how to create an Object.

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
var book = new Object(); // Create the object
book.subject = "Java Script"; // Assign properties to the object
book.author = "Dhruba & Sushanta";
</script>
</head>
<body>
<script type="text/javascript">
document.write("Book name is : " + book.subject + "<br>");
document.write("Book author is : " + book.author + "<br>");
</script>
</body>
</html>
```

Output

Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
function book(title, author){
this.title = title;
this.author = author;
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("Java Script", " Dhruba & Sushanta");
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
</script>
</body>
</html>
```

6.1.6 Defining Methods for an Object

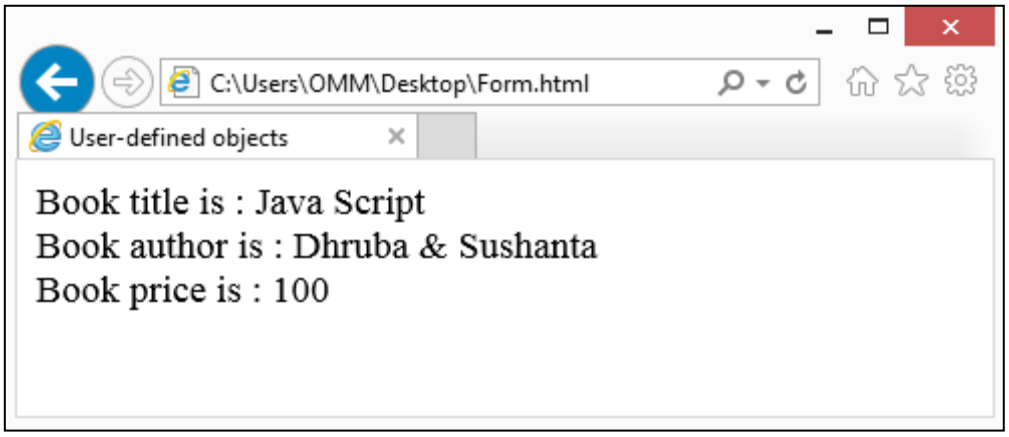
The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

The following example; it shows how to add a function along with an object.

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
// Define a function which will work as a method
function addPrice(amount){
this.price = amount;
}
function book(title, author){
this.title = title;
this.author = author;
this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
```

```
<script type="text/javascript">
var myBook = new book("Java Script", "Dhruba & Sushanta");
myBook.addPrice(100);
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>
```



6.1.7 The'with'Keyword

The **'with'** keyword is used as a kind of shorthand for referencing an object's properties or methods. The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows:

```
with (object)
{
properties used without the object name and dot
}
```

Example

The following example.

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
// Define a function which will work as a method
function addPrice(amount){
with(this){
price = amount;
}
}
function book(title, author){
this.title = title;
```

```
this.author = author;
this.price = 0;
this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("Perl", "Mohtashim");
myBook.addPrice(100);
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>
```

6.2 Working with numbers

The **Number** object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

Syntax

The syntax for creating a **number** object is as follows:

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

6.2.1 Number Properties

Here is a list of each property and their description.

Property	Description
MAX_VALUE	The largest possible value a number in JavaScript can have 1.7976931348623157E+308
MIN_VALUE	The smallest possible value a number in JavaScript can have 5E-324
NaN	Equal to a value that is not a number.
NEGATIVE_INFINITY	A value that is less than MIN_VALUE.
POSITIVE_INFINITY	A value that is greater than MAX_VALUE
prototype	A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
constructor	Returns the function that created this object's instance. By default this is the Number object.

6.2.1.1 MAX_VALUE

The **Number.MAX_VALUE** property belongs to the static **Number** object. It represents constants for the largest possible positive numbers that JavaScript can work with. The actual value of this constant is 1.7976931348623157 x 10308.

Syntax

The syntax to use MAX_VALUE is:

```
var val = Number.MAX_VALUE;
```

Example:

The following example to learn how to use MAX_VALUE.

```
<html>
<head>
<script type="text/javascript">
<!--
function showValue()
{
var val = Number.MAX_VALUE;
document.write ("Value of Number.MAX_VALUE : " + val );
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="showValue();"
/>
</form>
</body>
</html>
```

Output

6.2.1.2 MIN_VALUE

The **Number.MIN_VALUE** property belongs to the static **Number** object. It represents constants for the smallest possible positive numbers that JavaScript can work with.
The actual value of this constant is 5×10^{-324}.

Syntax

The syntax to use MIN_VALUE is:
var val = Number.MIN_VALUE;

6.2.1.3 NaN

Unquoted literal constant NaN is a special value representing Not-a-Number. Since NaN always compares unequal to any number, including NaN, it is usually used to indicate an error condition for a function that should return a valid number.

Note: Use the **isNaN()** global function to see if a value is an NaN value.

Syntax
The syntax to use NaN is:
var val = Number.NaN;

6.2.1.4 NEGATIVE_INFINITY

This is a special numeric value representing a value less than Number.MIN_VALUE. This value is represented as "-Infinity". It resembles an infinity in its mathematical behavior. For example, anything multiplied by NEGATIVE_INFINITY is NEGATIVE_INFINITY, and anything divided by NEGATIVE_INFINITY is zero. Because NEGATIVE_INFINITY is a constant, it is a read-only property of Number.

Syntax

The syntax to use NEGATIVE_INFINITY is as follows:

var val = Number. NEGATIVE_INFINITY;

6.2.1.5 POSITIVE_INFINITY

This is a special numeric value representing any value greater than Number.MAX_VALUE. This value is represented as "Infinity". It resembles an infinity in its mathematical behavior. For example, anything multiplied by POSITIVE_INFINITY is POSITIVE_INFINITY, and anything divided by POSITIVE_INFINITY is zero. As POSITIVE_INFINITY is a constant, it is a read-only property of Number.

Syntax

Use the following syntax to use POSITIVE_INFINITY.
var val = Number. POSITIVE_INFINITY;

6.2.1.6 Prototype

The prototype property allows you to add properties and methods to any object (Number, Boolean, String and Date etc.).

Note: Prototype is a global property which is available with almost all the objects.

Syntax

Use the following syntax to use Prototype.

object.prototype.name = value

6.1.2.7 Constructor

It returns a reference to the Number function that created the instance's prototype.

Syntax

Its syntax is as follows:

number. Constructor ( )

Return value

Returns the function that created this object's instance.

6.2.2 Number Methods

The Number object contains only the default methods that are a part of every object's definition.

Method	Description
toExponential()	Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
toFixed()	Formats a number with a specific number of digits to the right of the decimal.
toLocaleString()	Returns a string value version of the current number in a format that may vary according to a browser's local settings.
toPrecision()	Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
toString()	Returns the string representation of the number's value.
valueOf()	Returns the number's value.

6.2.2.1 to Exponential ()

This method returns a string representing the **number** object in exponential notation.

Syntax
Its syntax is as follows:
number.toExponential([fractionDigits])

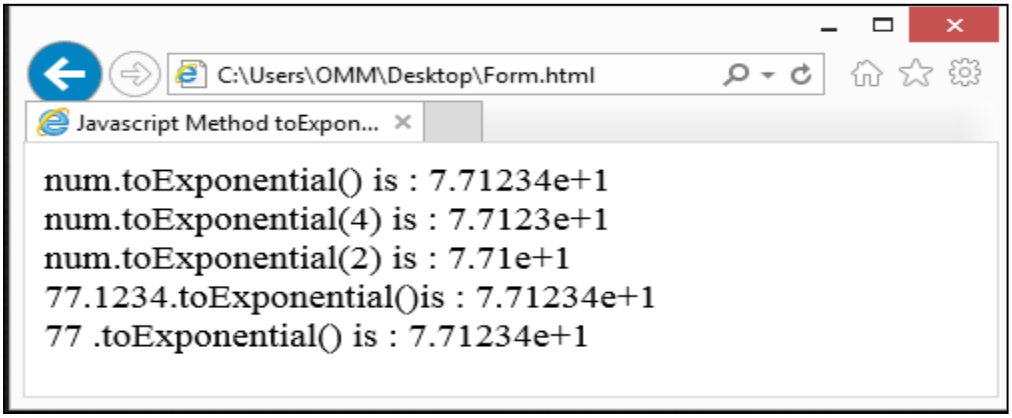
Parameter Details
fractionDigits: An integer specifying the number of digits after the decimal point. Defaults to as many digits as necessary to specify the number.

Return Value
A string representing a Number object in exponential notation with one digit before the decimal point, rounded to **fractionDigits** digits after the decimal point. If the **fractionDigits** argument is omitted, the number of digits after the decimal point defaults to the number of digits necessary to represent the value uniquely.

Example

The following example.

```
<html>
<head>
<title>Javascript Method toExponential()</title>
</head>
<body>
<script type="text/javascript">
var num=77.1234;
var val = num.toExponential();
document.write("num.toExponential() is : " + val );
document.write("<br />");
val = num.toExponential(4);
document.write("num.toExponential(4) is : " + val );
document.write("<br />");
val = num.toExponential(2);
document.write("num.toExponential(2) is : " + val);
document.write("<br />");
val = 77.1234.toExponential();
document.write("77.1234.toExponential()is : " + val );
document.write("<br />");
val = 77.1234.toExponential();
document.write("77 .toExponential() is : " + val);
</script>
</body>
</html>
```



6.2.2.2 to Fixed ()

This method formats a **number** with a specific number of digits to the right of the decimal.

Syntax

Its syntax is as follows:

number.toFixed([digits])

Parameter Details

digits: The number of digits to appear after the decimal point.

Return Value

A string representation of *number* that does not use exponential notation and has the exact number of **digits** after the decimal place.

6.2.2.3 toLocaleString ()

This method converts a **number** object into a human readable string representing the number using the locale of the environment.

Syntax

Its syntax is as follows:

number.toLocaleString()

Return Value

Returns a human readable string representing the number using the locale of the environment.

6.2.2.4 toPrecision ()

This method returns a string representing the **number** object to the specified precision.

Syntax

Its syntax is as follows:

number.toPrecision([precision])

Parameter Details

precision: An integer specifying the number of significant digits.

Return Value

Returns a string representing a Number object in fixed-point or exponential notation rounded **toprecision** significant digits.

6.2.2.5 toString ()

This method returns a string representing the specified object. The **toString**() method parses its first argument, and attempts to return a string representation in the specified radix (base).

Syntax

Its syntax is as follows:

number.toString([radix])

Parameter Details

radix: An integer between 2 and 36 specifying the base to use for representing numeric values.

Return Value

Returns a string representing the specified Number object.

6.3 Working with Boolean

The **Boolean** object represents two values, either "true" or "false". If *value* parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

Syntax

Use the following syntax to create a **boolean** object.

var val = new Boolean(value);

6.3.1 Boolean Properties

Here is a list of the properties of Boolean object:

Property	Description
constructor	Returns a reference to the Boolean function that created the object.
prototype	The prototype property allows you to add properties and methods to an object.

6.3.2 constructor ()

Javascript boolean **constructor()** method returns a reference to the Boolean function that created the instance's prototype.

Syntax

Use the following syntax to create a Boolean constructor() method.

boolean.constructor()

Return Value

Returns the function that created this object's instance.

Example

```
<html>
<head>
<title>JavaScript constructor() Method</title>
</head>
<body>
<script type="text/javascript">
var bool = new Boolean( );
document.write("bool.constructor() is : " + bool.constructor);
</script>
</body>
</html>
```

Output

6.3.3 Boolean Methods

Here is a list of the methods of Boolean object and their description.

Method	Description
toSource()	Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
toString()	Returns a string of either "true" or "false" depending upon the value of the object.
valueOf()	Returns the primitive value of the Boolean object.

6.3.4 toSource ()

Javascript boolean **toSource()** method returns a string representing the source code of the object.

Note: This method is not compatible with all the browsers.

Syntax
Its syntax is as follows:
boolean.toSource()
Return Value
Returns a string representing the source code of the object.

Example

```
<html>
<head>
<title>JavaScript toSource() Method</title>
</head>
<body>
<script type="text/javascript">
function book(title, publisher, price)
{
this.title = title;
this.publisher = publisher;
this.price = price;
}
var newBook = new book("Java Script","OSOU Inc",200);
document.write("newBook.toSource() is : "+ newBook.toSource());
</script>
</body>
</html>
```

6.3.5 valueOf ()

Javascript boolean **valueOf()** method returns the primitive value of the specified **boolean** object.

Syntax

Its syntax is as follows:

boolean.valueOf()

Return Value

Returns the primitive value of the specified **boolean** object.

Example

The following example.

```
<html>
<head>
<title>JavaScript toString() Method</title>
</head>
<body>
<script type="text/javascript">
var flag = new Boolean(false);
document.write( "flag.valueOf is : " + flag.valueOf() );
</script>
</body>
</html>
```

6.4 Working with Strings

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

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

Syntax
Use the following syntax to create a String object:
var val = new String(string);
The **string** parameter is a series of characters that has been properly encoded.

6.4.1 String Properties

Here is a list of the properties of String object and their description.

Property	Description
constructor	Returns a reference to the String function that created the object.
length	Returns the length of the string.
prototype	The prototype property allows you to add properties and methods to an object.

6.4.2 Length

This property returns the number of characters in a string.

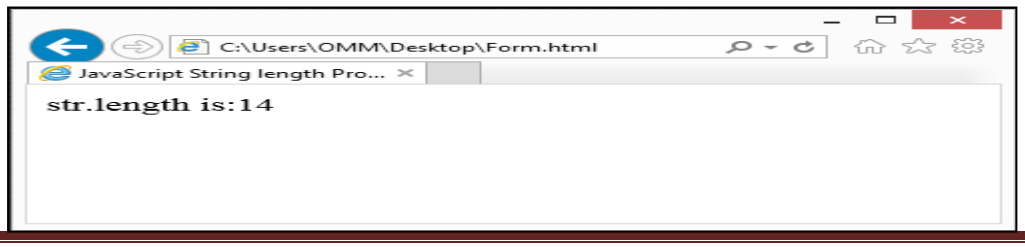
Syntax
Use the following syntax to find the length of a string:

string.length

Return Value
Returns the number of characters in the string.
Example

```
<html>
<head>
<title>JavaScript String length Property</title>
</head>
<body>
<script type="text/javascript">
var str = new String( "This is string" );
document.write("str.length is:" + str.length);
</script>
</body>
</html>
```

Output



6.4.3 String Methods

Here is a list of the methods available in String object along with their description.

Method	Description
charAt()	Returns the character at the specified index.
charCodeAt()	Returns a number indicating the Unicode value of the character at the given index.
concat()	Combines the text of two strings and returns a new string.
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
lastIndexOf()	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
localeCompare()	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order.
match()	Used to match a regular expression against a string.
replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
toString()	Returns a string representing the specified object.
toLowerCase()	Returns the calling string value converted to lower case.
toUpperCase()	Returns the calling string value converted to uppercase.

6.4.4 toLocaleUppereCase ()

This method is used to convert the characters within a string to uppercase while respecting the current locale. For most languages, it returns the same output as **toUpperCase**.

Syntax
Its syntax is as follows:

string.toLocaleUpperCase( )

Return Value
Returns a string in uppercase with the current locale.

Example
The following example.

```
<html>
<head>
<title>JavaScript String toLocaleUpperCase() Method</title>
</head>
<body>
<script type="text/javascript">
var str = "Apples are round, and Apples are Juicy.";
document.write(str.toLocaleUpperCase( ));
</script>
</body>
</html>
```

Output

6.5 Array and Array Management

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** Object.

```
var fruits = new Array( "apple", "orange", "mango" );
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295. You can create array by simply assigning values as follows:

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element

6.5.1 Array Properties

Here is a list of the properties of the Array object along with their description.

Property	Description
constructor	Returns a reference to the array function that created the object.
index	The property represents the zero-based index of the match in the string
input	This property is only present in arrays created by regular expression matches.
length	Reflects the number of elements in an array.
prototype	The prototype property allows you to add properties and methods to an object.

6.5.2 Constructor

JavaScript array **constructor** property returns a reference to the array function that created the instance's prototype.

Syntax

Its syntax is as follows:

```
array.constructor
```

Return Value

Returns the function that created this object's instance.

Example

The following example.

```
<html>
<head>
<title>JavaScript Array constructor Property</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array( 10, 20, 30 );
document.write("arr.constructor is:" + arr.constructor);
</script>
</body>
</html>
```

6.5.3 Array Methods

Here is a list of the methods of the Array object along with their description.

Method	Description
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
every()	Returns true if every element in this array satisfies the provided testing function.
filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
join()	Joins all elements of an array into a string.
lastIndexOf()	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reduce()	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
reduceRight()	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
slice()	Extracts a section of an array and returns a new array.
some()	Returns true if at least one element in this array satisfies the provided testing function.
toSource()	Represents the source code of an object

sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

6.5.4 Concat()

Javascript array **concat( )** method returns a new array comprised of this array joined with two or more arrays.

Syntax

The syntax of concat() method is as follows:

array.concat(value1, value2, ..., value n);

Parameter Details

valueN : Arrays and/or values to concatenate to the resulting array.

Return Value

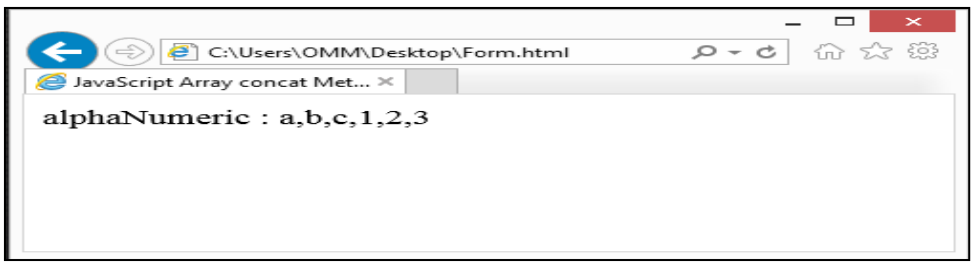
Returns the length of the array.

Example

The following example.

```
<html>
<head>
<title>JavaScript Array concat Method</title>
</head>
<body>
<script type="text/javascript">
var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric);
document.write("alphaNumeric : " + alphaNumeric );
</script>
</body>
</html>
```

Output



6.6 Working with Date

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date()** as shown below. Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.
The ECMA Script standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

Syntax
You can use any of the following syntaxes to create a Date object using Date() constructor.

new Date( )
new Date(milliseconds)
new Date(datestring)
new Date(year,month,date[,hour,minute,second,millisecond])

Note: Parameters in the brackets are always optional.
Here is a description of the parameters:

- **No Argument:** With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds:** When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring:** When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
- **7 agruments:** To use the last form of the constructor shown above. Here is a description of each argument:
- **year:** Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
- **month:** Integer value representing the month, beginning with 0 for January to 11 for December.
- **date:** Integer value representing the day of the month.
- **hour:** Integer value representing the hour of the day (24-hour scale).
- **minute:** Integer value representing the minute segment of a time reading.
- **second:** Integer value representing the second segment of a time reading.
- **millisecond:** Integer value representing the millisecond segment of a time reading.

6.6.1 Date Properties

Here is a list of the properties of the Date object along with their description.

Property	Description
constructor	Specifies the function that creates an object's prototype.
prototype	The prototype property allows you to add properties and methods to an object.

6.6.2 Constructor

JavaScript date **constructor** property returns a reference to the array function that created the instance's prototype.

Syntax

Its syntax is as follows:

date.constructor

Return Value

Returns the function that created this object's instance.

Example

The following example.

```
<html>
<head>
<title>JavaScript Date constructor Property</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date();
document.write("dt.constructor is : " + dt.constructor);
</script>
</body>
</html>
```

6.6.3 Date Methods

Here is a list of the methods used with **Date** and their description.

Method	Description
Date()	Returns today's date and time
getDate()	Returns the day of the month for the specified date according to local time.
getDay()	Returns the day of the week for the specified date according to local time.
getFullYear()	Returns the year of the specified date according to local time.
getHours()	Returns the hour in the specified date according to local time.

getMilliseconds()	Returns the milliseconds in the specified date according to local time.
getMinutes()	Returns the minutes in the specified date according to local time.
getMonth()	Returns the month in the specified date according to local time.
getSeconds()	Returns the seconds in the specified date according to local time.
getTime()	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
getTimezoneOffset()	Returns the time-zone offset in minutes for the current locale.
getUTCDate()	Returns the day (date) of the month in the specified date according to universal time.
getUTCDay()	Returns the day of the week in the specified date according to universal time.
getUTCFullYear()	Returns the year in the specified date according to universal time.
getUTCHours()	Returns the hours in the specified date according to universal time.
getUTCMilliseconds()	Returns the milliseconds in the specified date according to universal time.
setUTCMonth()	Sets the month for a specified date according to universal time.
setUTCSeconds()	Sets the seconds for a specified date according to universal time.
setYear()	**Deprecated** - Sets the year for a specified date according to local time. Use setFullYear instead.
toDateString()	Returns the "date" portion of the Date as a human-readable string.
toGMTString()	**Deprecated** - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
toLocaleDateString()	Returns the "date" portion of the Date as a string, using the current locale's conventions.
toLocaleFormat()	Converts a date to a string, using a format string.
toLocaleString()	Converts a date to a string, using the current locale's conventions.
toLocaleTimeString()	Returns the "time" portion of the Date as a string, using the current locale's conventions.
toSource()	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
toString()	Returns a string representing the specified Date object.
toTimeString()	Returns the "time" portion of the Date as a human-readable string.

toUTCString()	Converts a date to a string, using the universal time convention.
valueOf()	Returns the primitive value of a Date object.

6.6.4 Date()

Javascript **Date()** method returns today's date and time and does not need any object to be called.

Syntax

Its syntax is as follows:

Date()

Return Value

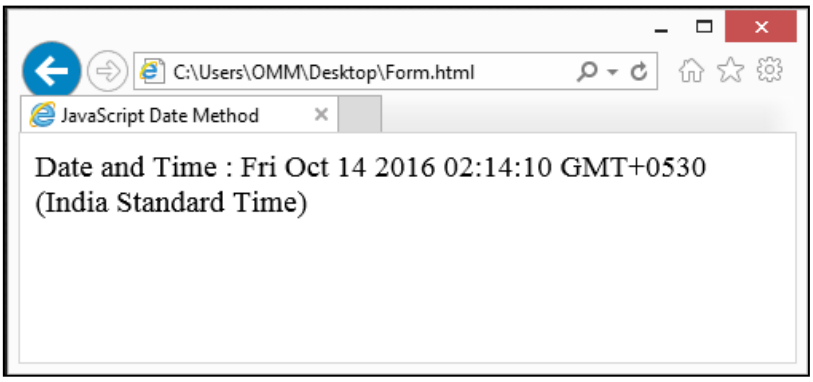
Returns today's date and time.

Example

The following example.

```
<html>
<head>
<title>JavaScript Date Method</title>
</head>
<body>
<script type="text/javascript">
var dt = Date();
document.write("Date and Time : " + dt );
</script>
</body>
</html>
```

Output



6.7 Doing Mathematical Operation

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

Thus, you refer to the constant **pi** as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

Syntax

The syntax to call the properties and methods of Math are as follows:

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

6.7.1 Math Properties

Here is a list of all the properties of Math and their description.

6.7.2 Math-E

Property	Description
E	Euler's constant and the base of natural logarithms, approximately 2.718.
LN2	Natural logarithm of 2, approximately 0.693.
LN10	Natural logarithm of 10, approximately 2.302.
LOG2E	Base 2 logarithm of E, approximately 1.442.
LOG10E	Base 10 logarithm of E, approximately 0.434.
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
SQRT1_2	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
SQRT2	Square root of 2, approximately 1.414.

This is an Euler's constant and the base of natural logarithms, approximately 2.718.

Syntax

Its syntax is as follows:

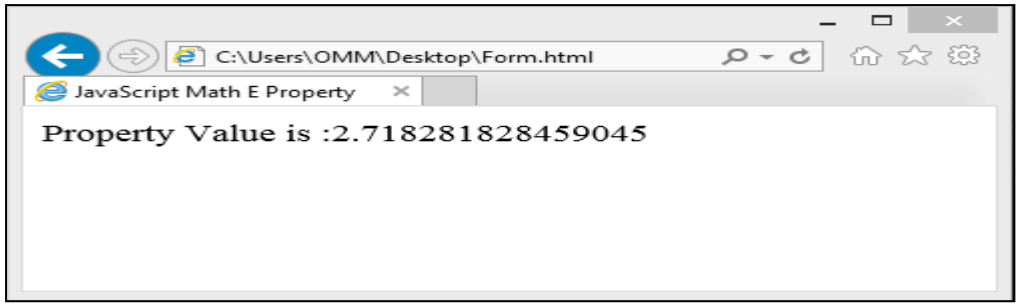
```
Math.E
```

Example

The following example program.

```
<html>
<head>
<title>JavaScript Math E Property</title>
</head>
<body>
<script type="text/javascript">
var property_value = Math.E
document.write("Property Value is :" + property_value);
</script>
</body>
</html>
```

Output



6.7.3 Math Methods

Here is a list of the methods associated with Math object and their description.

Method	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine (in radians) of a number.
asin()	Returns the arcsine (in radians) of a number.
atan()	Returns the arctangent (in radians) of a number.
atan2()	Returns the arctangent of the quotient of its arguments.
ceil()	Returns the smallest integer greater than or equal to a number.
cos()	Returns the cosine of a number.
exp()	Returns EN, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
floor()	Returns the largest integer less than or equal to a number
log()	Returns the natural logarithm (base E) of a number.
max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.
pow()	Returns base to the exponent power, that is, base exponent.
random()	Returns a pseudo-random number between 0 and 1.
round()	Returns the value of a number rounded to the nearest integer.
sin()	Returns the sine of a number.
sqrt()	Returns the square root of a number.
tan()	Returns the tangent of a number.
toSource()	Returns the string "Math".

6.7.4 sqrt ( )

This method returns the square root of a number. If the value of a number is negative, sqrt returns NaN.

Syntax
Its syntax is as follows:

Math.sqrt (x);

Parameter Details

x: A number.

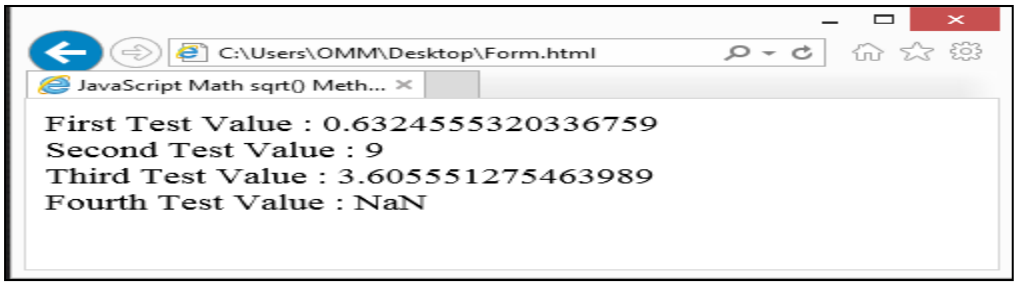
Return Value

Returns the square root of a given number.

Example
The following example program.

```
<html>
<head>
<title>JavaScript Math sqrt() Method</title>
</head>
<body>
<script type="text/javascript">
var value = Math.sqrt( 0.2 );
document.write("First Test Value : " + value );
var value = Math.sqrt( 81 );
document.write("<br />Second Test Value : " + value );
var value = Math.sqrt( 13 );
document.write("<br />Third Test Value : " + value );
var value = Math.sqrt( -4 );
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output



CHECK YOUR PROGRESS 1

Q1. What are the four capabilities of oop ?

Answer:_____
_____

Q2. Write the use of New Operator ?

Answer:_____
_____

Q3. Which method returns the square root of a number?

Answer:_____
_____

Q4. Which method returns the absolute value of a number?

Answer:_____
_____

Q5. Which property returns a reference to the array function that created the instance's prototype in java script?

Answer:_____
_____

Q6. What is array ?

Answer:_____
_____

Q7. What is String?

Answer:_____
_____

Q8. How many values represents by Boolean object?

Answer:_____
_____

6.8 Working with Regular Expression

A regular expression is an object that describes a pattern of characters. The JavaScript **RegExp**class represents regular expressions, and both String and **RegExp**define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax
A regular expression could be defined with the **RegExp()** constructor, as follows:
var pattern = new RegExp(pattern, attributes);

The description of the parameters:

- **pattern:** A string that specifies the pattern of the regular expression or another regular expression.
- **attributes:** An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively

Brackets
Brackets ([]) have a special meaning when used in the context of regular expressions.

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

6.8.1 RegExp Properties

Here is a list of the properties associated with RegExp and their description

Property	Description
constructor	Specifies the function that creates an object's prototype.
global	Specifies if the "g" modifier is set.
ignoreCase	Specifies if the "i" modifier is set.
lastIndex	The index at which to start the next match.
multiline	Specifies if the "m" modifier is set.
source	The text of the pattern

6.8.2 Multiline

Multiline is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs multiline matching, i.e., whether it was created with the "m" attribute.

Syntax
Its syntax is as follows:

RegExpObject.multiline

Return Value
Returns "TRUE" if the "m" modifier is set, "FALSE" otherwise.

Example

```
<html>
<head>
<title>JavaScript RegExp multiline Property</title>
</head>
<body>
<script type="text/javascript">
var re = new RegExp( "string" );
if ( re.multiline ){
document.write("Test1-multiline property is set");
}else{
document.write("Test1-multiline property is not set");
}
re = new RegExp( "string", "m" );
if ( re.multiline ){
document.write("<br/>Test2-multiline property is set");
}else{
document.write("<br/>Test2-multiline property is not set");
}
```

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

output

6.8.3.RegExp Methods

Method	Description
exec()	Executes a search for a match in its string parameter.
test()	Tests for a match in its string parameter.
toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.
toString()	Returns a string representing the specified object.

6.8.4 Test ( )

The **test** method searches string for text that matches regexp. If it finds a match, it returns true; otherwise, it returns false.

Syntax

Its syntax is as follows:

```
RegExpObject.test( string );
```

Parameter Details

string: The string to be searched.

Return Value

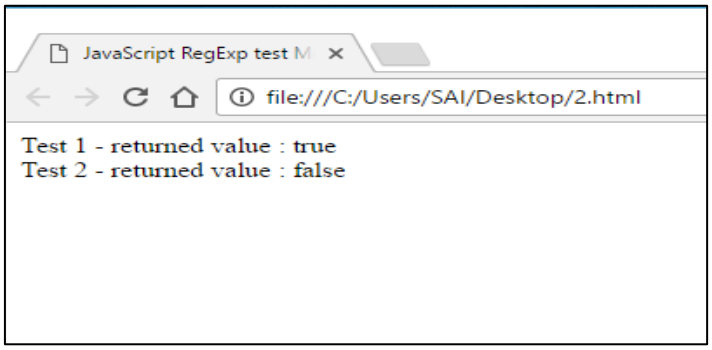
Returns the matched text if a match is found, and null if not.

Example

```
<html>
<head>
<title>JavaScript RegExp test Method</title>
</head>
<body>
<script type="text/javascript">
var str = "Javascript is interesting scripting language: Trisha pattnaik";
var re = new RegExp( "script", "g" );
var result = re.test(str);
document.write("Test 1 - returned value : " + result);
re = new RegExp( "pushing", "g" );
var result = re.test(str);
```

```
document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>
```

Output



6.9 Document Object Model (DOM)

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object:** Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object:** Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object:** Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements:** The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

6.9.1 The Legacy DOM

This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
This model provides several read-only properties, such as title, URL, and last Modified provide information about the document as a whole. Apart from that, there are various methods provided by this model which can be used to set and get document property values.

6.9.2 Document Properties in Legacy DOM

Here is a list of the document properties which are

Sl.No	Property and Description
1	alinkColor Deprecated - A string that specifies the color of activated links. Ex: document.alinkColor
2	anchors[ ] An array of Anchor objects, one for each anchor that appears in the document Ex: document.anchors[0], document.anchors[1] and so on
3	applets[ ] An array of Applet objects, one for each applet that appears in the document Ex: document.applets[0], document.applets[1] and so on
4	bgColor Deprecated - A string that specifies the background color of the document. Ex: document.bgColor
5	Cookie A string valued property with special behavior that allows the cookies associated with this document to be queried and set. Ex: document.cookie
6	Domain A string that specifies the Internet domain the document is from. Used for security purpose. Ex: document.domain
7	embeds[ ] An array of objects that represent data embedded in the document with the <embed> tag. A synonym for plugins []. Some plugins and ActiveX controls can be controlled with JavaScript code. Ex: document.embeds[0], document.embeds[1]

6.9.2 Document Methods in Legacy DOM

Here is a list of methods supported by Legacy DOM.

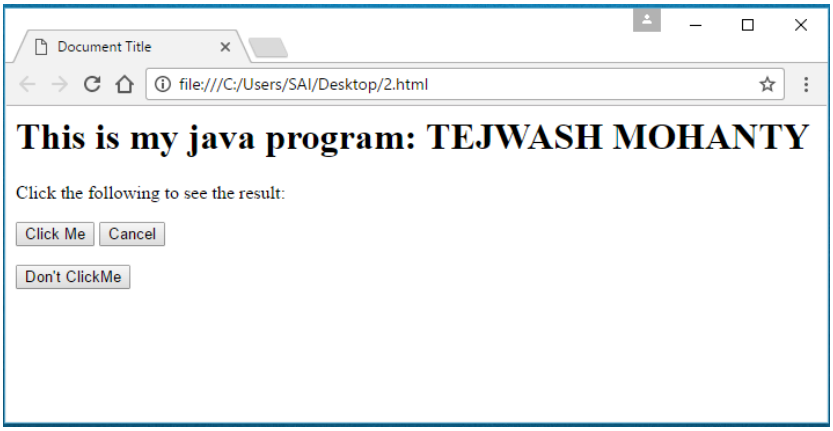
S.No	Property and Description
1	clear( ) Deprecated - Erases the contents of the document and returns nothing. Ex: document.clear( )
2	close( ) Closes a document stream opened with the open( ) method and returns nothing.
3	open( ) Deletes existing document content and opens a stream to which new document contents may be written. Returns nothing. Ex: document.open( )
4	write(value, ...) Inserts the specified string or strings into the document currently being parsed or appends to document opened with open( ). Returns nothing. Ex: document.write(value, ...)
5	writeln(value, ...) Identical to write( ), except that it appends a newline character to the output. Returns nothing. Ex: document.writeln(value, ...)

Example:

```
<html>
<head>
<title> Document Title </title>
<script type="text/javascript">
<!--
functionmyFunc()

{
var ret = document.title;
alert("Document Title : " + ret );
var ret = document.URL;
alert("Document URL : " + ret );
var ret = document.forms[0];
alert("Document First Form : " + ret );
var ret = document.forms[0].elements[1];
alert("Second element : " + ret );
}
//-->
</script>
```

```
</head>
<body>
<h1 id="title">This is my java program: TEJWASH MOHANTY </h1>
<p>Click the following to see the result:</p>
<form name="FirstForm">
<input type="button" value="Click Me" onclick="myFunc();" />
<input type="button" value="Cancel">
</form>
<form name="SecondForm">
<input type="button" value="Don't ClickMe"/>
</form>
</body>
</html>
```

Output



6.10 Error and Error Handling

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

Syntax Errors

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript

Runtime Errors

Runtime errors, also called **exceptions**, occur during execution (after compilation/interpretation).

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

The try...catch...finallyStatement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.
You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

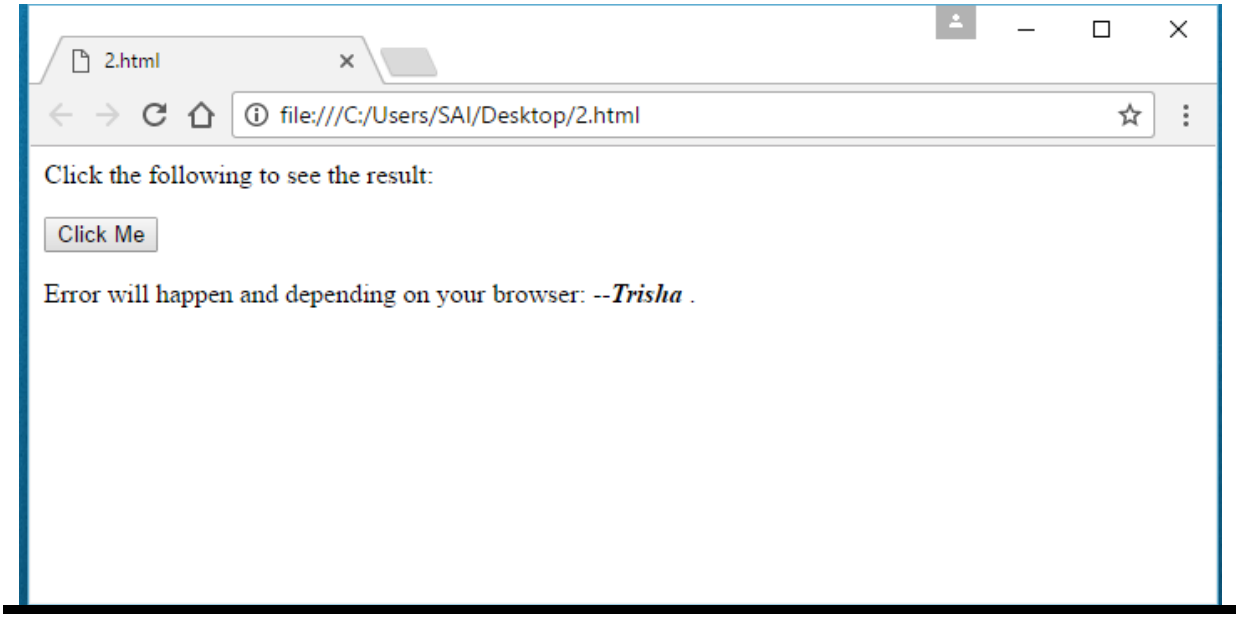
Here is the **try...catch...finally** block syntax:

```
<script type="text/javascript">
<!--
try {
// Code to run
[break;]
} catch ( e ) {
// Code to run if an exception occurs
[break;]
}[ finally {
// Code that is always executed regardless of
// an exception occurring
}]
//-->
</script>
```

Example

```
<html>
<head>
<script type="text/javascript">
<!--
functionmyFunc()
{
var a = 100;
document.write ("Value of variable a is : " + a );
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
<p>Error will happen and depending on your browser: trisha.</p>
</body>
</html>
```

OUTPUT



6.11 Client side validation

Validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** - First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** - Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

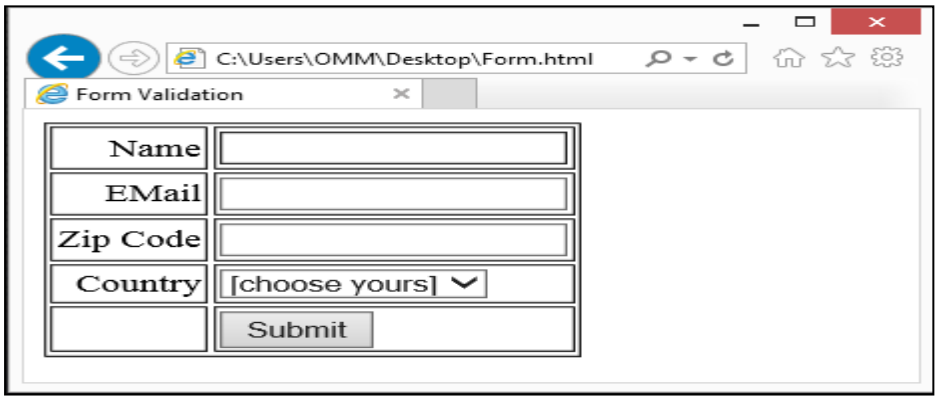
Example

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>
<body>
<form action="/cgi-bin/test.cgi" name="myForm"
```

```
onsubmit="return(validate());">
<table cellspacing="2" cellpadding="2" border="1">
<tr>
        <td align="right">Name</td>
<td><input type="text" name="Name" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" /></td>
</tr>
<tr>
<td align="right">Country</td>
<td>
<select name="Country">
<option value="-1" selected>[choose yours]</option>
<option value="1">USA</option>
<option value="2">UK</option>
<option value="3">INDIA</option>
</select>
</td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit" /></td>
</tr>
</table>
</form>
</body>
</html>
```

Output



6.11.1 Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{
if( document.myForm.Name.value == "" )
{
alert( "Please provide your name!" );
document.myForm.Name.focus() ;
return false;
}
if( document.myForm.EMail.value == "" )
{
alert( "Please provide your Email!" );
document.myForm.EMail.focus() ;
return false;

}
if( document.myForm.Zip.value == "" ||
isNaN( document.myForm.Zip.value ) ||
document.myForm.Zip.value.length != 5 )
{
alert( "Please provide a zip in the format #####." );
document.myForm.Zip.focus() ;
return false;
}
if( document.myForm.Country.value == "-1" )
{
alert( "Please provide your country!" );
return false;
}
return( true );
}
//-->
</script>
```

6.11.2 Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

```
<script type="text/javascript">
```

```
<!--
function validateEmail()
{
var emailID = document.myForm.EMail.value;
atpos = emailID.indexOf("@");
dotpos = emailID.lastIndexOf(".");
if (atpos < 1 || ( dotpos - atpos < 2 ))
{
alert("Please enter correct email ID")
document.myForm.EMail.focus() ;
return false;
}
return( true );
}
//-->
</script>
```

6.12 Animation in webpages

You can use JavaScript to create a complex animation having, but not limited to, the following elements:

- Fireworks
- Fade Effect
- Roll-in or Roll-out
- Page-in or Page-out
- Object movements

JavaScript can be used to move a number of DOM elements (<img />, , or any other HTML element) around the page according to some sort of pattern determined by a logical equation or function.
JavaScript provides the following two functions to be frequently used in animation programs.

- **setTimeout (function, duration)** - This function calls **function** after **duration** milliseconds from now.
- **setInterval (function, duration)** - This function calls **function** after every **duration** milliseconds.
- **clearTimeout (setTimeout_variable)** - This function clears any timer set by the setTimeout() function.

JavaScript can also set a number of attributes of a DOM object including its position on the screen. You can set *top* and left attribute of an object to position it anywhere on the screen. Here is its syntax.

```
// Set distance from left edge of the screen.
object.style.left = distance in pixels or points;
or
// Set distance from top edge of the screen.
object.style.top = distance in pixels or points;
```

6.12.1 Manual Animation

So let's implement one simple animation using DOM object properties and JavaScript functions as follows. The following list contains different DOM methods.

- We are using the JavaScript function **getElementById()** to get a DOM object and then assigning it to a global variable **imgObj**.
- We have defined an initialization function **init()** to initialize **imgObj** where we have set its **position** and **left** attributes.
- We are calling initialization function at the time of window load.
- Finally, we are calling **moveRight()** function to increase the left distance by 10 pixels. You could also set it to a negative value to move it to the left

Example

The following example.

```
<html>
<head>
<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
function init(){
imgObj = document.getElementById('myImage');
imgObj.style.position= 'relative';
imgObj.style.left = '0px';
}
function moveRight(){
imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
}
window.onload =init;
//-->
</script>
</head>
<body>

<form>
<img id="myImage" src="D:/4.jpg" />
<p>Click button below to move the image to right</p>
<input type="button" value="Click Me" onclick="moveRight();"
/>
</form>
</body>
</html>
```

Output

6

6.12.2 Automated Animation

In the above example, we saw how an image moves to right with every click. We can automate this process by using the JavaScript function **setTimeout()** as follows. Here we have added more methods. So let's see what is new here:

- The **moveRight()** function is calling **setTimeout()** function to set the position of *imgObj*.
- We have added a new function **stop()** to clear the timer set by **setTimeout()** function and to set the object at its initial position.

Example

The following example code.

```
<html>
<head>
<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
var animate ;
function init(){
imgObj = document.getElementById('myImage');
imgObj.style.position= 'relative';
imgObj.style.left = '0px';

}
```

```
function moveRight(){
imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
animate = setTimeout(moveRight,20); // call moveRight in 20msec
}
function stop(){
clearTimeout(animate);
imgObj.style.left = '0px';
}
window.onload =init;
//-->
</script>
</head>
<body>
<form>
<img id="myImage" src="/images/html.gif" />
<p>Click the buttons below to handle animation</p>
<input type="button" value="Start" onclick="moveRight();" />
<input type="button" value="Stop" onclick="stop();" />
</form>
</body> </html>
```

Output

6.13 Multimedia in webpages

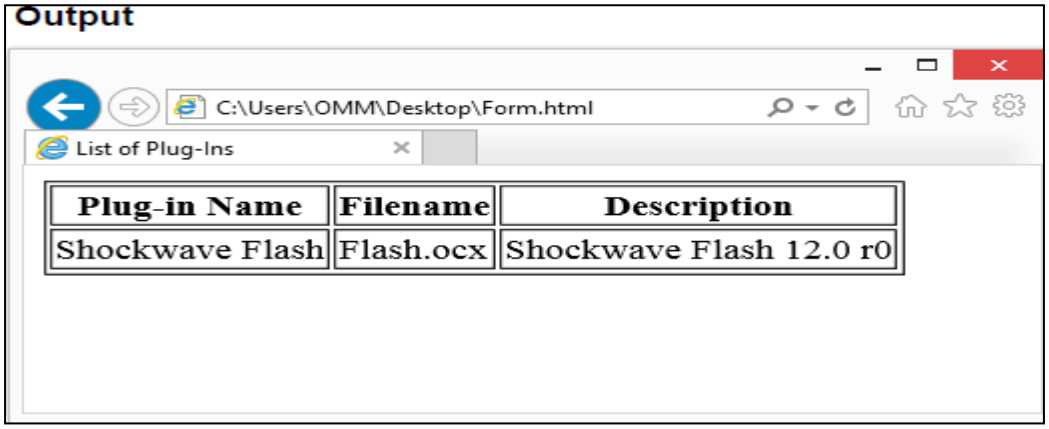
The JavaScript **navigator** object includes a child object called **plugins**. This object is an array, with one entry for each plug-in installed on the browser. The navigator.plugins object is supported only by Netscape, Firefox, and Mozilla only.

Example

Here is an example that shows how to list down all the plug-on installed in your browser:

```
<html>
<head>
<title>List of Plug-Ins</title>
</head>
<body>
<table border="1">
<tr><th>Plug-in
Name</th><th>Filename</th><th>Description</th></tr>
```

```
<script LANGUAGE="JavaScript" type="text/javascript">
for (i=0; i<navigator.plugins.length; i++) {
document.write("<tr><td>");
document.write(navigator.plugins[i].name);
document.write("</td><td>");
document.write(navigator.plugins[i].filename);
document.write("</td><td>");
document.write(navigator.plugins[i].description);
document.write("</td></tr>");
}
</script>
</table>
</body>
</html>
```



6.13.1 Checking for Plug-Ins

Each plug-in has an entry in the array. Each entry has the following properties:

- **name** - is the name of the plug-in.
- **filename** - is the executable file that was loaded to install the plug-in.
- **description** - is a description of the plug-in, supplied by the developer.
- **mimeTypes** - is an array with one entry for each MIME type supported by the plug-in.

You can use these properties in a script to find out the installed plug-ins, and then using JavaScript, you can play appropriate multimedia file. Take a look at the following example.

```
<html>
<head>
<title>Using Plug-Ins</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
media = navigator.mimeTypes["video/quicktime"];
```

```
if (media){
document.write("<embed src='quick.mov' height=100 width=100>");
}
else{

document.write("<img src='quick.gif' height=100 width=100>");
}
</script>
</body>
</html>
```

NOTE: Here we are using HTML <embed> tag to embed a multimedia file.

6.13.2 Controlling Multimedia

Let us take a real example which works in almost all the browsers.

```
<html>
<head>
<title>Using Embeded Object</title>
<script type="text/javascript">
<!--
function play()
{
if (!document.demo.IsPlaying()){
document.demo.Play();
}
}
function stop()
{
if (document.demo.IsPlaying()){
document.demo.StopPlay();
}
}
function rewind()
if (document.demo.IsPlaying()){
document.demo.StopPlay();
}
document.demo.Rewind();
}

//-->
</script>
</head>
<body>
<embed id="demo" name="demo"
src="http://www.amrood.com/games/kumite.swf"
width="318" height="300" play="false" loop="false"
pluginspage="http://www.macromedia.com/go/getflashplayer"
swliveconnect="true">
</embed>
```

```
<form name="form" id="form" action="#" method="get">
<input type="button" value="Start" onclick="play();" />
<input type="button" value="Stop" onclick="stop();" />
<input type="button" value="Rewind" onclick="rewind();" />
</form>
</body>
</html>
```

6.14 Image Map

You can use JavaScript to create client-side image map. Client-side image maps are enabled by the **usemap** attribute for the <img /> tag and defined by special <map> and <area> extension tags. The image that is going to form the map is inserted into the page using the <img /> element as normal, except that it carries an extra attribute called **usemap**. The value of the usemap attribute is the value of the name attribute on the <map> element, which you are about to meet, preceded by a pound or hash sign. The <map> element actually creates the map for the image and usually follows directly after the <img /> element. It acts as a container for the <area /> elements that actually define the clickable hotspots. The <map> element carries only one attribute, the **name** attribute, which is the name that identifies the map. This is how the <img /> element knows which <map> element to use. The <area> element specifies the shape and the coordinates that define the boundaries of each clickable hotspot. The following code combines image maps and JavaScript to produce a message in a text box when the mouse is moved over different parts of an image.

Example

```
<html>
<head>
<title>Using JavaScript Image Map</title>
<script type="text/javascript">
<!--
function show(name){
document.myform.stage.value = name
}
//-->
</script>
</head>
<body>
<form name="myform">
<input type="text" name="stage" size="20" />
</form>

<!-- Create Mappings -->
<img src="D:/usemap.gif" alt="HTML Map"
border="0" usemap="#abc"/>
<map name="abc">
<area shape="poly"
                coords="74,0,113,29,98,72,52,72,38,27"
                href="D:/html/index.htm" alt="Poly" target="_self"
                onMouseOver="show('PERL')"
```

```
                onMouseOut="show(')"/>
<area shape="rect"
                coords="22,83,126,125"
                href="D:/index.htm" alt="rect"
                target="_self"
                onMouseOver="show('HTML')"
                onMouseOut="show(')"/>
<area shape="circle"
                coords="73,168,32"
                href="D:/php/index.htm" alt="Circle"
                target="_self"
                onMouseOver="show('PHP')"
                onMouseOut="show(')"/>
</map>
</body>
</html>
```

Output

6.15 Introduction to XML

XML stands for E**X**tensible **M**arkup **L**anguage. It is a **markup language** much like HTML. It was designed to **describe data**. XML tags are not predefined in XML. You must **define your own tags**. XML is **self-describing**. XML uses a DTD (**Document Type Definition**) to formally describe the data. XML is a software- and hardware-independent tool for storing and transporting data. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

6.15.1 Definition:

Extensible Markup Language (**XML**) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable.

6.15.2 Characteristics of XML

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

6.15.3 Difference between XML & HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

6.15.4 XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

Check your progress 2

Q1. What is plug-in ?

Answer:_____
_____
_____

Q2. What is Basic Validation ?

Answer:_____
_____
_____

Q3. What is Regular expression?

Answer:_____
_____
_____

Q4. How many types of error ? what are they ?

Answer:_____
_____
_____

Q5. Write the syntax of **try...catch...finally** block ?

Answer:_____
_____
_____

Q6. Write the Full form of DOM ?

Answer:_____
_____
_____

Q7. Write the Full form of XML ?

Answer:_____
_____
_____

Q8. What is XML ?write its characteristics.

Answer:_____
_____
_____

6.16 Reference

1. Java script Bible Gold Edition

2. www. Google.com

3.www. tutorialpoints.com

4. Wrox: Professional Java Script

5. Java wiley Java script Bible

6.17 Let us sum up

we have understood working with Objects,Numbers,Boolean,Strings,

Array and array management, Date and date management, mathematical

operations, regular expression, document object model, errors and

error handling, client side validation, animations in webpages,

multimedia in webpages, image map, working with xml

6.18 Check your progress –possible answers

Answers to check your progress 1

Q1. What are the four capabilities of oop ?

Answer: The four capabilities of oop are:

1. Inheritance
2. Encapsulation
3. Aggregation
4. Polymorphism

Q2. Write the use of New Operator?

Answer: The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

Q3. Which method returns the square root of a number?

Answer: sqrt( )

Q4. Which method returns the absolute value of a number?

Answer: abs( )

Q5. Which property returns a reference to the array function that created the instance's prototype in java script?

Answer: constructor

Q5. What is array?

Answer: An **array** is a collection of data items, all of the same type, accessed using a common name. Or The **Array** object lets you store multiple values in a single variable. ... An **array** is used to store a collection of data, but it is often more useful to think of an **array** as a collection of variables of the same type.

Q6. What is String?

Answer: The String **object** lets you work with a series of characters; it wraps JavaScript's string primitive data type with a number of helper methods. As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String **object** on a string primitive.

Q7. How many values represents by Boolean object?

Answer: The **Boolean** object represents two values, either "true" or "false".

Answers to check your progress 2

Q1. What is plug-in?

Answer: The JavaScript **navigator** object includes a child object called **plugins**. This object is an array, with one entry for each plug-in installed on the browser. The navigator.plugins object is supported only by Netscape, Firefox, and Mozilla only.

Q2. What is Basic Validation?

Answer: First let us see how to do a basic form validation. In the above form, we are calling **validate ()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate () function.

Q3. What is Regular expression?

Answer: A regular expression is an object that describes a pattern of characters. The JavaScript **RegExp**class represents regular expressions, and both String and

RegExpdefine methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Q4. How many types of error? What are they?

Answer: Three types of error. Syntax, logical & runtime error.

Q5. Write the syntax of **try...catch...finally** block ?

Answer:
Here is the **try...catch...finally** block syntax:

```
<script type="text/javascript">
<!--
try {
// Code to run
[break;]
} catch ( e ) {
// Code to run if an exception occurs
[break;]
}[ finally {
// Code that is always executed regardless of
// an exception occurring
}]
//-->
</script>
```

Q6. Write the Full form of DOM ?

Answer: DOM- Document Object Model

Q7. Write the Full form of XML ?

Answer: XML- Extensible Markup Language

Q8. What is XML ?write its characteristics.

Answer: Extensible Markup Language (**XML**) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable.

Characteristics of XML

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.

XML is a public standard: XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.