CHAPTER 2

JAVASCRIPT BASICS

2.1 Comments

The JavaScript comments are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.
There are two types of comments in JavaScript.

1. Single-line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.
Example:-

```
<script>
//It is single line comment
document.write("hello javascript");
</script>
```

2. Multi-line Comment

It can be used to add single as well as multi line comments. It is represented by forward slash with asterisk then asterisk with forward slash.

```
<script>
        /* It is multi line comment.
        It will not be displayed */
        document.write("example of javascript multiline comment");
</script>
```

2.2 Variables

A Variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.
There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Example:

```
<script>
var x = 10;
```

```
var y = 20;
var z=x+y;
document.write(z);
</script>
```

2.3 Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40; //holding number
var b="Rahul"; //holding string
```

Primitive data types

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e., no value at all

Non-primitive data types

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

2.4 Operators

JavaScript operators are symbols that are used to perform operations on operands.

Example:

```
var sum=10+20;
```

Here, + is the arithmetic operator and = is the assignment operator.

Types of operators

1) Arithmetic Operators
2) Comparison (Relational) Operators
3) Logical Operators
4) Assignment Operators

Arithmetic operators are used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Comparison operators compares the two operands.

Operator	Description	Example
==	Is equal to	10==20 = false
!=	Not equal to	10!=20 = true
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

Logical Operators

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Assignment Operators

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

2.5 If Statement

The **if-else statement** is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
<script>
var a=20;
if(a>10)
{
document.write("value of a is greater than 10");
}
</script>
```

If...else Statement

It evaluates the content whether condition is true of false.

Example of if-else statement in JavaScript to find out the even or odd number.

```
<script>
    var a=20;
    if(a%2==0)
    {
        document.write("a is even number");
    }
    else
    {
        document.write("a is odd number");
    }
</script>
```

If...else if statement

It evaluates the content only if expression is true from several expressions.

A simple example of if else if statement in javascript.

```
<script>
    var a=20;
    if(a==10)
    {
        document.write("a is equal to 10");
    }
    else if(a==15)
    {
        document.write("a is equal to 15");
    }
    else if(a==20)
    {
        document.write("a is equal to 20");
    }
    else
    {
        document.write("a is not equal to 10, 15 or 20");
    }
</script>
```

2.6 Switch

The switch statement is used to execute one code from multiple expressions.

A simple example of switch statement in javascript.

```
<script>
    var grade='B';
```

```
        var result;
        switch(grade)
        {
                case 'A': result="A Grade"; break;
                case 'B': result="B Grade"; break;
                case 'C': result="C Grade"; break;
                default: result="No Grade";

        }
        document.write(result);
</script>
```

2.7 Loop

The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1) for loop
2) while loop
3) do-while loop

The for loop iterates the elements for the fixed number of times. It should be used if number of iterations is known.

Example

```
<script>
        for (i=1; i<=5; i++)
        {
                document.write(i + "<br/>") }
</script>
```

The while loop iterates the elements for the infinite number of times. It should be used if number of iteration is not known.

Example

```
<script>
        var i=11;
        while (i<=15)
        {
                document.write(i + "<br/>");
                i++;
        }
</script>
```

The do while loop iterates the elements for the infinite number of times like while loop. But code is executed at least once whether condition is true or false.

Example

```
<script>
```

```
var i=21;
do
{
document.write(i + "<br/>");
i++;
}while (i<=25);
</script>
```

2.8 Functions

Functions are blocks of code used to perform specific operations. Using function helps developers to reduce the complexity and size of the program source code, also functions once defined can be used any number of times in the program.

Example

```
<script>
        function msg()
        {
                Document.write ("Welcome to GTEC Computer Education");
        }

        //calling the function
        msg();
</script>
```

Function Arguments

Arguments or parameters can be used to call functions , The function uses the parameter values as input data to process using the code defined in the function.

Example

```
<script>
        function sumTwo(n1,n2)
        {
                var sum=n1+n2;
                document.write("Sum of " + n1 + " and " + n2 + " is "+ sum);
        }
        //calling the function multiple times
        sumTwo(4,5);
        sumTwo(-4,-7);
</script>
```

Function with Return Value

A function can return a value to the calling program , the value can be of any primitive datatype of Javascript.

Example

```
<script>
```

```
        function max(x,y)
        {
                return x>y ? x:y ;
        }
        //calling the function
        x=15;
        y=11;
        document.write( "<h1>Larger of "+ x + " and " + y + " is " +
        max(x,y));
</script>
```

CHAPTER 3

JAVASCRIPT OBJECTS

3.1 Object

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

3.1.1 Creating Object using object literal

Example

```
<script>
        emp={id:102,name:"Shyam Kumar",salary:40000}
        document.write(emp.id+" "+emp.name+" "+emp.salary);
</script
```

3.1.2 Creating instance of Object directly (using new keyword)

Example

```
<script>
        var emp=new Object();
        emp.id=101;
        emp.name="Ravi Malik";
        emp.salary=50000.
        document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

3.1.3 Creating an object using constructor (using new keyword)

```
<script>
        function emp(id,name,salary)
        {
                this.id=id;
                this.name=name;
                this.salary=salary;
        }
        e=new emp(103,"Vimal Jaiswal",30000);
        document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

3.2 Array

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

3.2.1 Creating array using array literal

Example

```
<script>
        var emp=["Ramesh", "Tommy", "Kris", "Harry", "James"];
        for (i=0; i<emp.length; i++)
        {
                document.write(emp[i] + "<br/>");
        }
</script>
```

3.2.2 Creating instance of Array directly (using new keyword)

Example

```
<script>
        var i;
        var emp = new Array();
        emp[0] = "Ram";
        emp[1] = "Tom";
        emp[2] = "Kris";
        emp[3] = "Harry";
        emp[4] = "James";
        for (i=0; i<emp.length; i++)
        {
                document.write(emp[i] + "<br>");
        }
</script>
```

3.2.3 Creating an array using Array constructor (using new keyword)

Example

```
<script>
        var emp=new Array("Jairam", "Vijaya", "Smitha", "Wilfred");
        for (i=0;i<emp.length;i++)
        {
                document.write(emp[i] + "<br>");
        }
</script>
```

3.3 String

The String is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

3.3.1 Defining a string using string literal

The string literal is created using double quotes.

Example

```
<script>
    var str="Welcome go GTEC Education";
    document.write(str);
</script>
```

3.3.2 Defining a string using string object (using new keyword)

The new keyword must be used to create instance of string.

Example

```
<script>
    var stringname=new String("Welcome to GTEC Education");
    document.write(stringname);
</script>
```

3.3.3 String Methods

Methods	Description
charAt()	It provides the char value present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching a character from the last position.
search()	It searches a specified regular expression in a given string and returns its position if a match occurs.
match()	It searches a specified regular expression in a given string and returns that regular expression if a match occurs.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.

substring()	It is used to fetch the part of the given string on the basis of the specified index.
slice()	It is used to fetch the part of the given string. It allows us to assign positive as well negative index.
toLowerCase()	It converts the given string into lowercase letter.
toUpperCase()	It converts the given string into uppercase letter.
toString()	It provides a string representing the particular object.
valueOf()	It provides the primitive value of string object.
split()	It splits a string into substring array, then returns that newly created array.
trim()	It trims the white space from the left and right side of the string.

3.4 Date

The JavaScript date object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

3.4.1 Date Constructor

You can use 4 variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

3.4.2 Date Example

```
<script>
        var date=new Date();
        var day=date.getDate();
        var month=date.getMonth()+1;
        var year=date.getFullYear();
        document.write("<br>Date is: "+day+"/"+month+"/"+year);
</script>
```

3.4.3 Current Time Example

```
Current Time: <span id="txt"></span>
<script>
        var today=new Date();
        var h=today.getHours();
        var m=today.getMinutes();
        var s=today.getSeconds();
        document.getElementById('txt').innerHTML=h+":"+m+":"+s;
</script>
```

3.4.4 Digital Clock Example

```
<h1 align="center">Current Time: <span id="clock"></span></h1>
<script>
        function getTime()
        {
                var today=new Date();
                var h=today.getHours();
                var m=today.getMinutes();
                var s=today.getSeconds();
                document.getElementById('clock').innerHTML=h+":"+m+":"+s;
        }
        setInterval("getTime()",1000);
</script>
```

3.4.5 Date Methods

Methods	Description
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
getFullYears()	It returns the integer value that represents the year on the basis of local time.
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
getMilliseconds()	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.

getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
getSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.
setDate()	It sets the day value for the specified date on the basis of local time.
setDay()	It sets the particular day of the week on the basis of local time.
setFullYears()	It sets the year value for the specified date on the basis of local time.
setHours()	It sets the hour value for the specified date on the basis of local time.
setMilliseconds()	It sets the millisecond value for the specified date on the basis of local time.
setMinutes()	It sets the minute value for the specified date on the basis of local time.
setMonth()	It sets the month value for the specified date on the basis of local time.
setSeconds()	It sets the second value for the specified date on the basis of local time.
toDateString()	It returns the date portion of a Date object.
toString()	It returns the date in the form of string.
toTimeString()	It returns the time portion of a Date object.
toUTCString()	It converts the specified date in the form of string using UTC time zone.
valueOf()	It returns the primitive value of a Date object.

3.5 Math

The Math object provides several constants and methods to perform mathematical operation, it doesn't have constructors.

3.5.1 Sample program to find the square root of the given number using math.sqrt(n) method.

```
<h1 align="center">Square Root of 17 is: <span id="p1"></span></h1>
<script>
    document.getElementById('p1').innerHTML=Math.sqrt(17);
</script>
```

3.5.2 Some important Math methods

Methods	Description
abs()	It returns the absolute value of the given number.
cbrt()	It returns the cube root of the given number.
ceil()	It returns a smallest integer value, greater than or equal to the given number.
cos()	It returns the cosine of the given number.
floor()	It returns largest integer value, lower than or equal to the given number.
max()	It returns maximum value of the given numbers.
min()	It returns minimum value of the given numbers.
pow()	It returns value of base to the power of exponent.
random()	It returns random number between 0 (inclusive) and 1 (exclusive).
round()	It returns closest integer value of the given number.
sign()	It returns the sign of the given number
sin()	It returns the sine of the given number.
sqrt()	It returns the square root of the given number
tan()	It returns the tangent of the given number.
trunc()	It returns an integer part of the given number.

function compresse

CHAPTER 4

BROWSER OBJECTS

4.1 Introduction

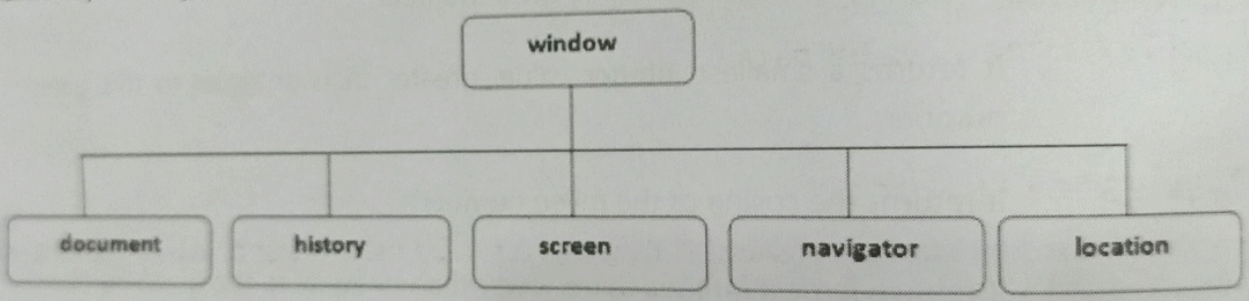
The **Browser Object Model** (BOM) is used to interact with the browser.
The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

window.alert("hello javatpoint");

Or

alert("hello javatpoint");

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth.



4.2 Window Object

The window object represents a window in browser. An object of window is created automatically by the browser.
The important methods of window object are as follows:

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

4.3 History Object

The History object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.
There are three methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

Example

```
history.back(); //for previous page
history.forward(); //for next page
history.go(2); //for next 2nd page
history.go(-2); //for previous 2nd page
```

4.4 Navigator Object

The Navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.
Properties of navigator object that returns information of the browser:

Property	Description
appName	returns the name
appVersion	returns the version
platform	returns the platform e.g. Win32.
online	returns true if browser is online otherwise false.

4.5 Screen Object

The **Screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.
Properties of screen object that returns information of the browser:

Property	Description
width	returns the width of the screen
height	returns the height of the screen
colorDepth	returns the color depth
pixelDepth	returns the pixel depth.

CHAPTER 5

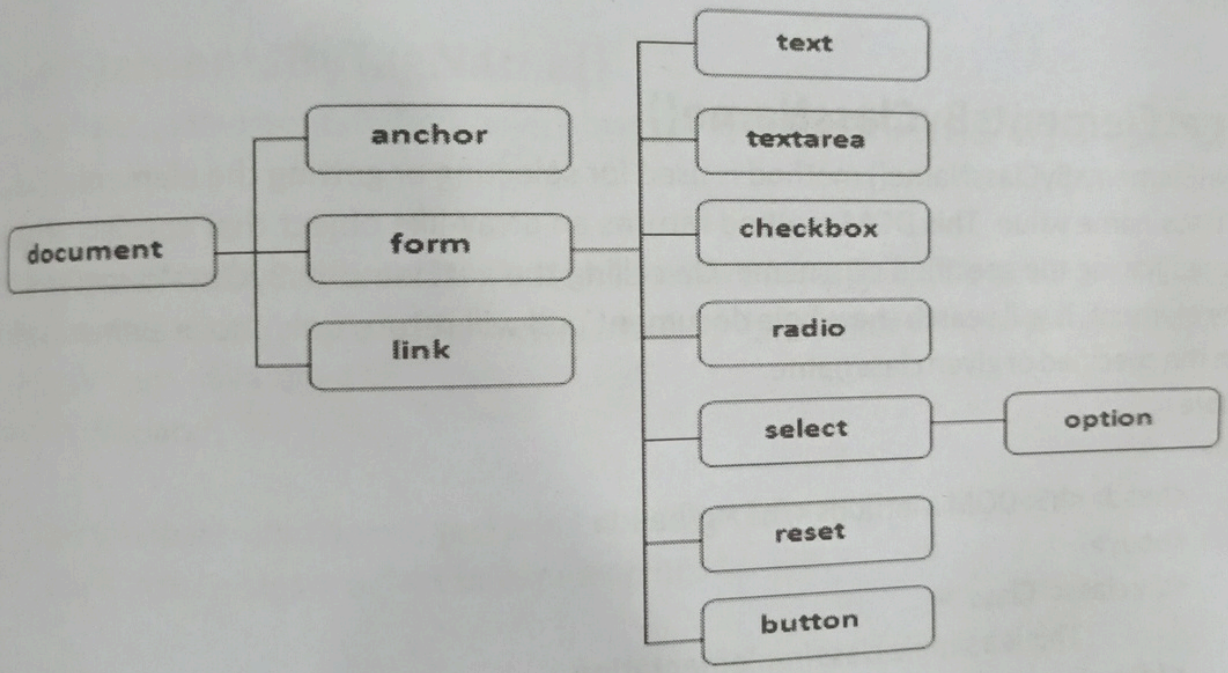
JAVASCRIPT DOM

5.1 Document Object Model

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. document objects help you to add dynamic content to our web page.

5.5.1 Properties of document object



5.5.2 Methods of document object

Method	Description
write("string")	writes the given string on the document.
writeln("string")	writes the given string on the document with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

function

5.2 getElementById()

The document.getElementById() method returns the element of specified id.

Example

```
<script type="text/javascript">
function getcube()
{
        var number=document.getElementById("number").value;
        alert(number*number*number);

}
</script>
<form>
        Enter No:<input type="text" id="number" name="number"/><br/>
        <input type="button" value="cube" onclick="getcube()"/>
</form>
```

5.3 getElementsByClassName()

The getElementsByClassName() method is used for selecting or getting the elements through their class name value. This DOM method returns an array-like object that consists of all the elements having the specified classname. On calling the getElementsByClassName() method on any element, it will search the whole document and will return only those elements which match the specified or given class name.

Example

```
<html>
        <head><h5>DOM Methods</h5></head>
        <body>
        <div class="Class">
                This is a simple class implementation
        </div>
        <script type="text/javascript">
                var x=document.getElementsByClassName('Class');
                document.write("On calling x, it will return an array-like object: <br>"+x);
        </script>
        </body>
</html>
```

5.4 getElementsByName()

The document.getElementsByName() method returns all the element of specified name.

Example

Program to count total number of genders. Here, we are using getElementsBy Name()

```
method to get all the genders.
<script type="text/javascript">
function totalelements()

{       var allgenders=document.getElementsByName("gender");
        alert("Total Genders:"+allgenders.length);

}
</script>
<form>
        Male:<input type="radio" name="gender" value="male">
        Female:<input type="radio" name="gender" value="female">
        <input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

5.5 getElementsByTagName()

The document.getElementsByTagName() method returns all the element of specified tag name.

Example

Program to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

```
<script type="text/javascript">
function countpara()
{
        var totalpara=document.getElementsByTagName("p");
        alert("total p tags are: "+totalpara.length);

}
</script>

<p>Welcome to GTEC Education</p>
<p>Here we are going to count total number of paragraphs by
getElementByTagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

5.6 innerHTML property()

The innerHTML property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example

Script to create the html form when user clicks on the button. we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

```
<script type="text/javascript">
function showcommentform()
{
        var data="Name:<input type='text' name='name'>";
        data+="<br>Comment:<br>";
        data+="<textarea rows='5' cols='80'></textarea>" ;
        data+="<br><input type='submit' value='Post Comment'>";
        document.getElementById('mylocation').innerHTML=data;
}
</script>

<form name="myForm">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"></div>
</form>
```

5.7 innerText property()

The innerText property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

```
<script type="text/javascript">
function validate()
{
        var msg;
        if(document.myForm.userPass.value.length>5)
        {
                msg="good";
        }
        else
        {
                msg="poor";
        }
        document.getElementById('mylocation').innerText=msg;
}
```

```
</script>

<form name="myForm">
        <input type="password" value="" name="userPass" onkeyup="validate()">
        Strength:<span id="mylocation">no strength</span>
</form>
```

CHAPTER 6

JAVASCRIPT VALIDATION

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.
JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.
Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

6.1 form validation

Example
we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>
function validateform()
{
        var name=document.myform.name.value.
        var password=document.myform.password.value;
        if (name==null || name=="")
        {
                alert("Name can't be blank");
                return false;
        }
        else if(password.length<6)
        {
                alert("Password must be at least 6 characters long.");
                return false;
        }
}
</script>
<body>
        <form name="myform" method="post" action="abc.jsp"
                onsubmit="return validateform()" >
                Name:<input type="text" name="name"><br/>
```

```
        Password: <input type="password" name="password"><br/>
        <input type="submit" value="register">
    </form>
</body>
```

6.2 email validation

There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after . (dot).

Example

```
<script>
    function validateemail()
    {
        var x=document.myform.email.value;
        var atposition=x.indexOf("@");
        var dotposition=x.lastIndexOf(".");
        if (atposition<1 || dotposition<atposition+2
        || dotposition+2>=x.length)
        {
            alert("Please enter a valid e-mail address \n tpostion:"+
            atposition+ "\n dotposition:"+ dotposition);
            return false;
        }
    }
</script>

<body>
    <form name="myform"  method="post"
        action="#" onsubmit="return validateemail();">
        Email: <input type="text" name="email"><br/>
        <input type="submit" value="register">
    </form>
</body>
```

CHAPTER 7

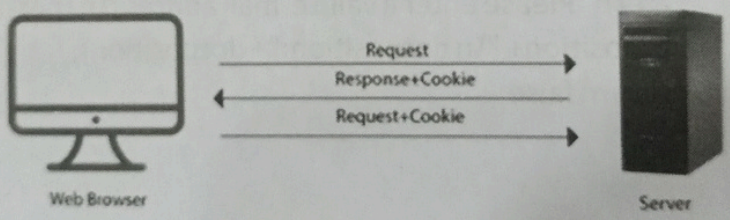
COOKIES

7.1 Cookies

A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.
A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

7.1.1 Cookies Workflow

- When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
- So, to recognize the old user, we need to add the cookie with the response from the server.
- browser at the client-side.
- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



7.1.2 Creating a Cookie and accessing a cookie

In JavaScript, we can create, read, update and delete a cookie by using document.cookie property.

Example

```
<html>
    <script>
            function setCookie()
            {
                    document.cookie="username=Sujith Kumar";
            }
            function getCookie()
            {
                    if(document.cookie.length!=0)
```

```
                {
                        alert(document.cookie);
                }
                else
                {
                        alert("Cookie not available");
                }
        }
</script>
<body>
        <input type="button" value="setCookie" onclick="setCookie()">
        <input type="button" value="getCookie" onclick="getCookie()">
</body>
</html>
```

7.1.3 Displaying a Cookie

```
<html>
  <body>
        <input type="button" value="setCookie" onclick="setCookie()">
        <input type="button" value="getCookie" onclick="getCookie()">
  </body>
  <script>
        function setCookie()
        {
                document.cookie="username=Duke Martin";
        }
        function getCookie()
        {
                if(document.cookie.length!=0)
                {
                var array=document.cookie.split("=");
                        alert("Name="+array[0]+" "+"Value="+array[1]);
                }
                else
                {
                        alert("Cookie not available");
                }
        }
  </script>
</html>
```

7.2 Cookie Attributes

JavaScript provides some optional attributes that enhance the functionality of cookies. Here, is the list of some attributes with their description.

Attributes	Description
expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie up to the specified time. Here, time is given in seconds.
path	It expands the scope of the cookie to all the pages of a website.
domain	It is used to specify the domain for which the cookie is valid.

7.2.1 Cookie expires attribute

The cookie expires attribute provides one of the ways to create a persistent cookie. Here, a date and time are declared that represents the active period of a cookie. Once the declared time is passed, a cookie is deleted automatically.

Example

```
<html>
    <body>
            <input type="button" value="setCookie" onclick="setCookie()">
            <input type="button" value="getCookie" onclick="getCookie()">
    </body>
    <script>
            function setCookie()
            {
                    document.cookie="username=Sujith Kumar;
                expires=Sun, 20 Aug 2020 12:00:00 UTC";
            }
            function getCookie()
            {
                    if(document.cookie.length!=0)
                    {
                            var array=document.cookie.split("=");
                            alert("Name="+array[0]+" "+"Value="+array[1]);
                    }
                    else
                    {
```

```
                    alert("Cookie not available");
                }
        }
    </script>
</html>
```

7.2.2 Cookie max-age attribute

The cookie max-age attribute provides another way to create a persistent cookie. Here, time is declared in seconds. A cookie is valid up to the declared time only.

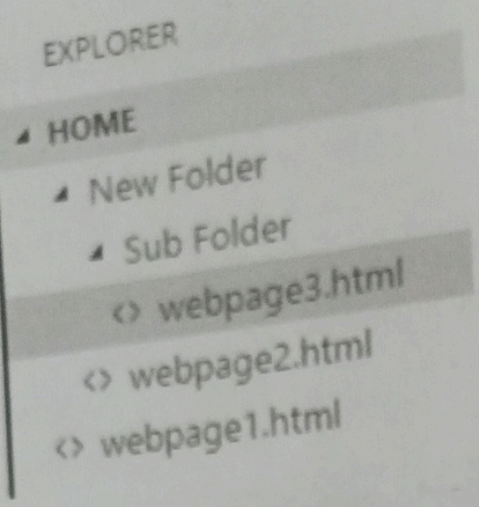
Example

```
<html>
    <body>
        <input type="button" value="setCookie" onclick="setCookie()">
        <input type="button" value="getCookie" onclick="getCookie()">
    </body>
    <script>
        function setCookie()
        {
            document.cookie="username=Sujith Kumar"
        + ";max-age=" + (60 * 60 * 24 * 365) + ";"
        }
        function getCookie()
        {
            if(document.cookie.length!=0)
            {
                var array=document.cookie.split("=");
                alert("Name="+array[0]+" "+"Value="+array[1]);
            }
            else
            {
                alert("Cookie not available");
            }
        }
    </script>
</html>
```

7.2.3 Cookie path attribute

If a cookie is created for a webpage, by default, it is valid only for the current directory and sub-directory. JavaScript provides a path attribute to expand the scope of cookie up to all the pages of a website.

Example



Here, if we create a cookie for webpage2.html, it is valid only for itself and its sub-directory (i.e., webpage3.html). It is not valid for webpage1.html file. here, we use path attribute to enhance the visibility of cookies up to all the pages. Here, you all just need to do is to maintain the above directory structure and put the below program in all three web pages. Now, the cookie is valid for each web page.

```
<html>
    <body>
        <input type="button" value="setCookie" onclick="setCookie()">
        <input type="button" value="getCookie" onclick="getCookie()">
    </body>
    <script>
        function setCookie()
        {
            document.cookie="username=Duke Martin;max-age="
            +(60 * 60 * 24 * 365) + ";path=/;"
        }
        function getCookie()
        {
            if(document.cookie.length!=0)
            {
                var array=document.cookie.split("=");
                alert("Name="+array[0]+" "+"Value="+array[1]);
            }
            else
            {
                alert("Cookie not available");
            }
        }
```

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

7.2.4 Cookie domain attribute

A JavaScript domain attribute specifies the domain for which the cookie is valid.

7.3 Cookie with multiple Name

In JavaScript, a cookie can contain only a single name-value pair. However, to store more than one name-value pair, we can use the following approach: -

- Serialize the custom object in a JSON string, parse it and then store in a cookie.
- For each name-value pair, use a separate cookie.

Example

Script to check whether a cookie contains more than one name-value pair.

```
<html>
    <body>
        Name: <input type="text" id="name"><br>
        Email: <input type="email" id="email"><br>
        Course: <input type="text" id="course"><br>
        <input type="button" value="Set Cookie" onclick="setCookie()">
        <input type="button" value="Get Cookie" onclick="getCookie()">
    </body>
    <script>
        function setCookie()
        {
            var info="Name="+
            document.getElementById("name").value";Email=" +
            document.getElementById("email").value +
            ";Course="+document.getElementById("course").value;
            document.cookie=info;
        }
        function getCookie()
        {
            if(document.cookie.length!=0)
            {
                alert(document.cookie);
            }
            else
            {
                alert("Cookie not available")
            }
        }
```

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

7.4 Deleting Cookies

JavaScript also allows us to delete a cookie. Mentioned below are the possible ways to delete a cookie.

- A cookie can be deleted by using expire attribute.
- A cookie can also be deleted by using max-age attribute.
- We can delete a cookie explicitly, by using a web browser.

Example

Deleting a cookie using expire attribute by providing expiry date (i.e., any date before the system date) to it.

```
<html>
    <body>
        <input type="button" value="Set Cookie" onclick="setCookie()">
        <input type="button" value="Get Cookie" onclick="getCookie()">
    </body>
    <script>
        function setCookie()
        {
            document.cookie="name=Sujith Kumar;" +
            "expires=Sun, 20 Aug 2000 12:00:00 UTC";
        }
        function getCookie()
        {
            if(document.cookie.length!=0)
            {
                alert(document.cookie);
            }
            else
            {
                alert("Cookie not avaliable");
            }
        }
</script>
</body>
</html>
```

CHAPTER 8

JAVASCRIPT EVENTS

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the kev

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

8.1 addEventListener()

The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

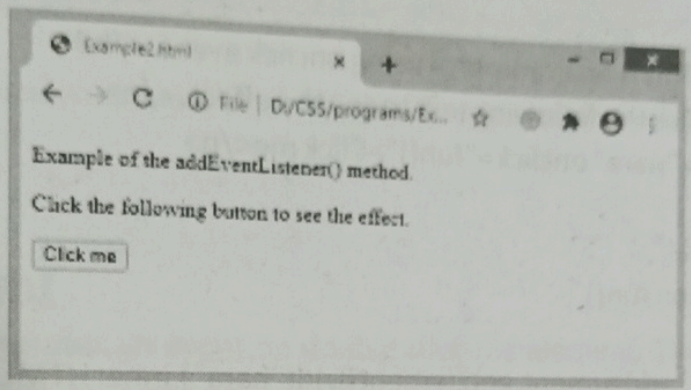
The **addEventListener()** method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

Example

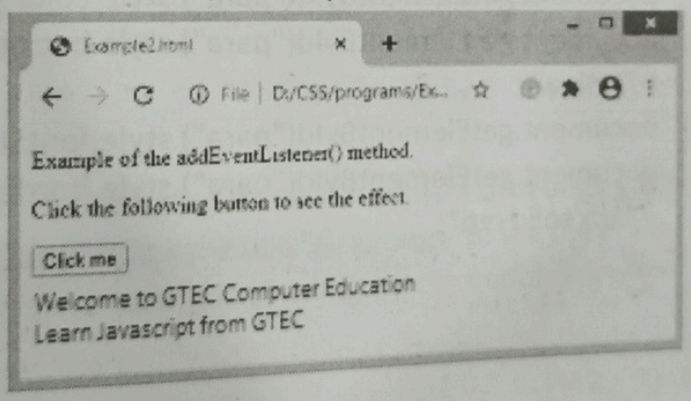
```
<html>
    <body>
        <p> Example of the addEventListener() method. </p>
        <p> Click the following button to see the effect. </p>
        <button id = "btn"> Click me </button>
        <p id = "para"></p>
```

```
    </body>
    <script>
        document.getElementById("btn").addEventListener("click", fun);
        function fun()
        {
            document.getElementById("para").innerHTML =
            "Welcome to GTEC Computer Education" + "<br>" +
            "Learn Javascript from GTEC";
        }
    </script>
</html>
```

Output



After clicking the given HTML button, the output will be -



8.2 onclick event

The **onclick** event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked. This event can be used for validating a form, warning messages and many more.

Using JavaScript, this event can be dynamically added to any element. It supports all HTML elements except the following tags.

**<html>, <head>, <title>, <style>, <script>, <base>, <iframe>, <bdo>,
, <meta>,** and **<param>.**

In HTML, we can use the **onclick** attribute and assign a JavaScript function to it. We can also use

the JavaScript's **addEventListener()** method and pass a **click** event to it for greater flexibility.

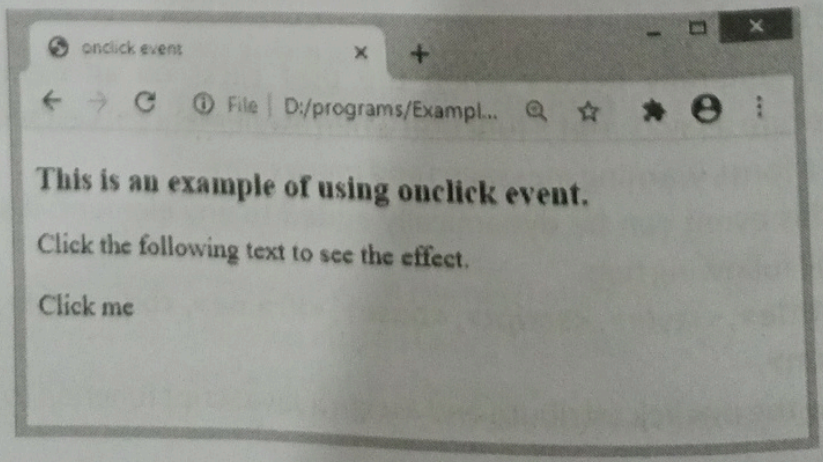
Example

In this example, we are using JavaScript's **onclick** event. Here we are using the **onclick** event with the paragraph element.

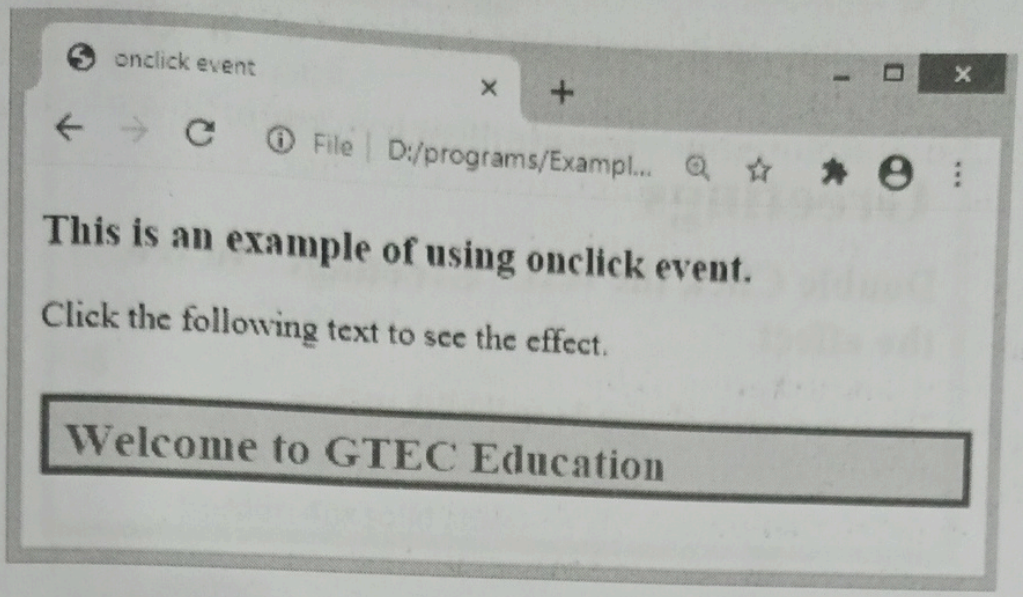
When the user clicks on the paragraph element, the corresponding function will get executed, and the text of the paragraph gets changed. On clicking the **<p>** element, the background color, size, border, and color of the text will also get change.

```
<html>
    <head>
        <title> onclick event </title>
    </head>
    <body>
        <h3> This is an example of using onclick event. </h3>
        <p> Click the following text to see the effect. </p>
        <p id = "para" onclick="fun()">Click me</p>
    </body>
    <script>
        function fun()
        {
            document.getElementById("para").innerHTML =
            "Welcome to the GTEC Education";
            document.getElementById("para").style.color = "blue";
            document.getElementById("para").style.backgroundColor
            = "yellow";
            document.getElementById("para").style.fontSize = "25px";
            document.getElementById("para").style.border
            = "4px solid red";
        }
    </script>
</html>
```

Output



After clicking the text Click me, the output will be -



8.3 dblclick event

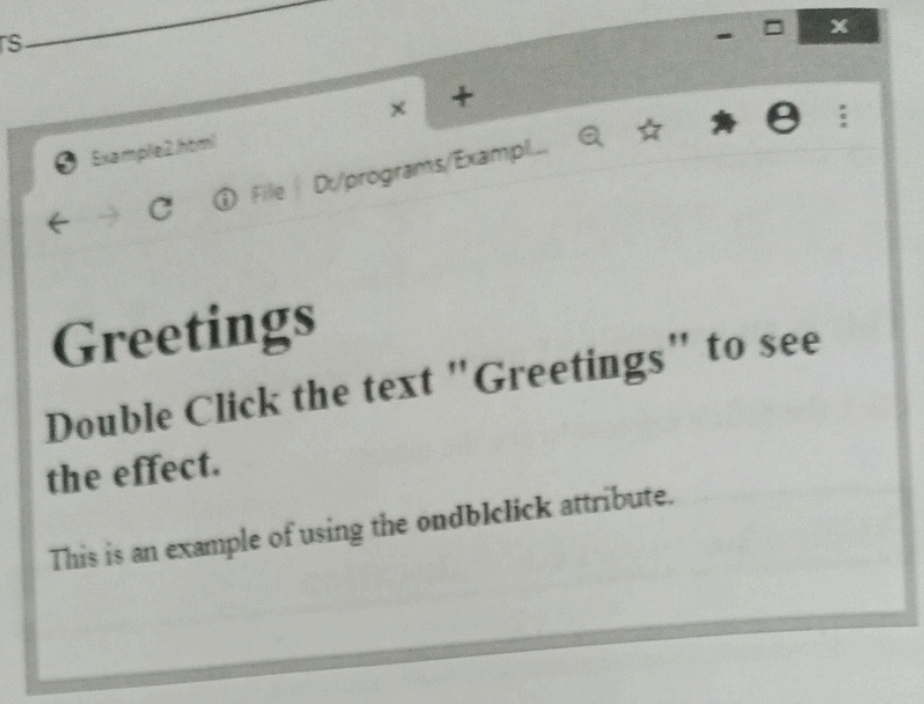
The **dblclick** event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's **addEventListener()** method to fire the double click event. In HTML, we can use the **ondblclick** attribute to create a double click event.

Example

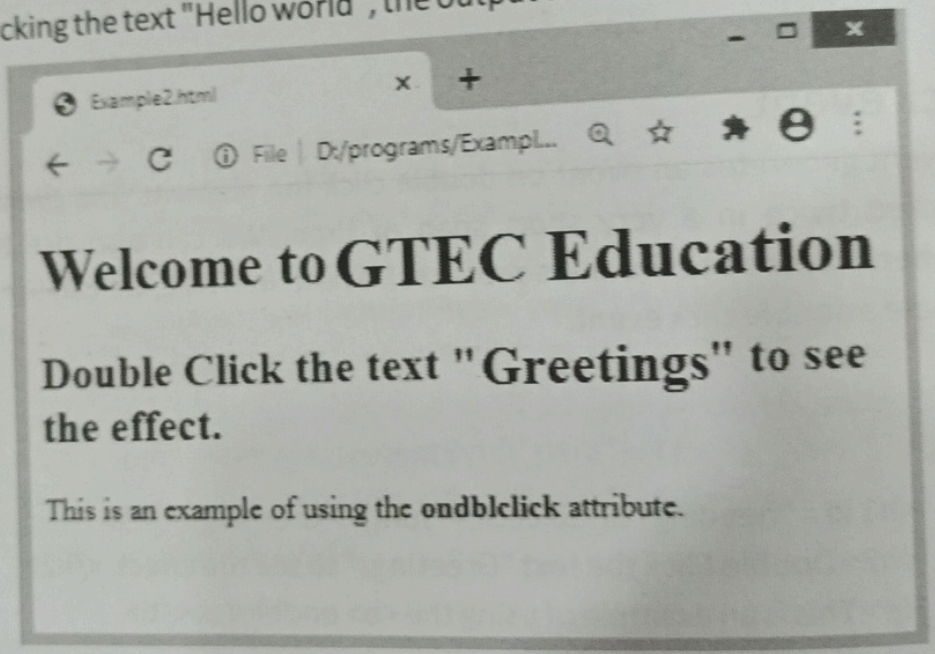
```
<html>
    <body>
        <h1 id = "heading" ondblclick = "fun()"> Greetings </h1>
        <h2> Double Click the text "Greetings" to see the effect. </h2>
        <p> This is an example of using the <b> ondblclick </b>
attribute. </p>
    </body>
    <script>
        function fun()
        {
            document.getElementById("heading").innerHTML =
            " Welcome to the GTEC Education ";

        }
    </script>
</html>
```

Output

After the execution of the above code, the output will be -



After double-clicking the text "Hello world", the output will be -



8.4 onload event

In HTML, the onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.

In **HTML,** the **onload** attribute is generally used with the **<body>** element to execute a script once the content (including CSS files, images, scripts, etc.) of the webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

The **document.onload** triggers before the loading of images and other external content. It is fired before the **window.onload.** While the **window.onload** triggers when the entire page loads, including CSS files, script files, images, etc.

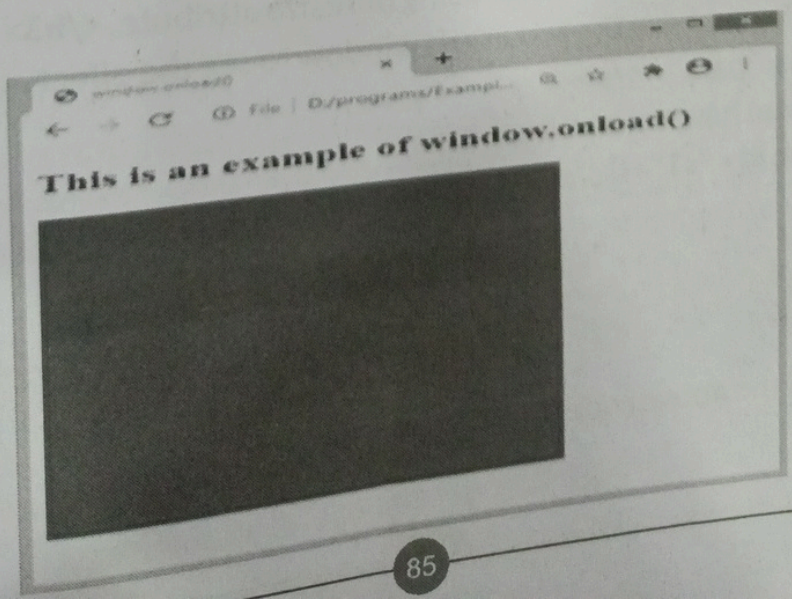
Example

In this example, there is a div element with a height of 200px and a width of 200px. Here, we are using the **window.onload()** to change the background color, width, and height of the **div** element after loading the web page.

The background color is set to **'red'**, and width and height are set to **300px** each.

```
<html>
    <head>
        <title> window.onload() </title>
        <style type = "text/css">
        #bg
        {
            width: 200px;    height: 200px;
            border: 4px solid blue;
        }
        </style>
        <script type = "text/javascript">
            window.onload = function()
            {
                document.getElementById("bg").style.backgroundColor
                = "red";
                document.getElementById("bg").style.width = "300px";
                document.getElementById("bg").style.height = "300px";
            }
        </script>
    </head>
    <body>
        <h2> This is an example of window.onload() </h2>
        <div id = "bg"></div>
    </body>
</html>
```

Output



8.5 onresize event

The **onresize** event in JavaScript generally occurs when the window has been resized. To get the size of the window, we can use JavaScript's the window. **outerWidth** and **window.outerHeight** events.

We can also use the JavaScript's properties such as **innerWidth, innerHeight, clientWidth, ClientHeight, offsetWidth, offsetHeight** to get the size of an element.

In HTML, we can use the **onresize** attribute and assign a JavaScript function to it.

Example

In this example, we are using the HTML onresize attribute. Here, we are using the **window.outerWidth** and **window.outerHeight** events of JavaScript to get the height and width of the window.

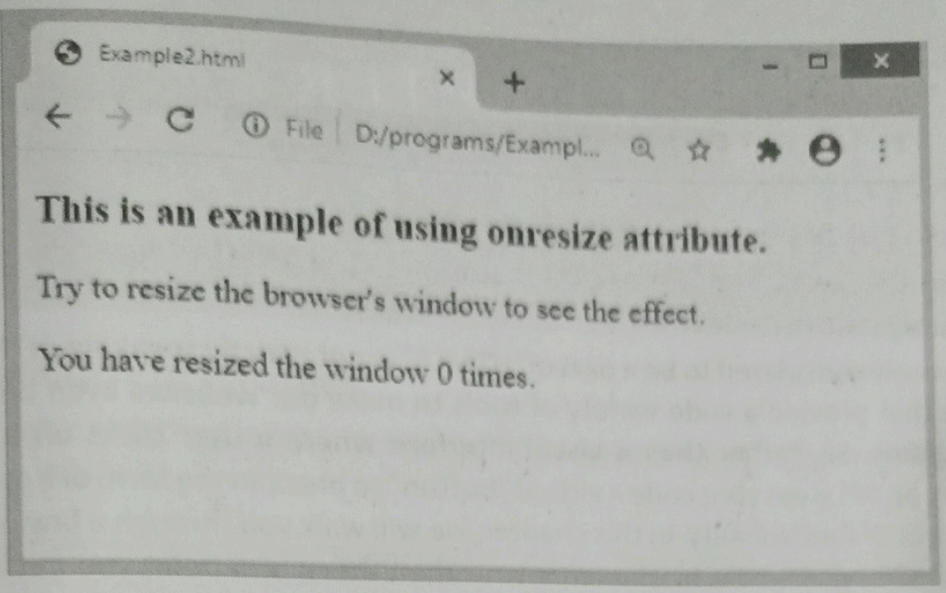
When the user resizes the window, the updated width and height of the window will be displayed on the screen. It will also display how many times the user tried to resize the window. When we change the height of the window, the updated height will change accordingly. Similarly, when we change the width of the window, the updated width will change accordingly.

Output

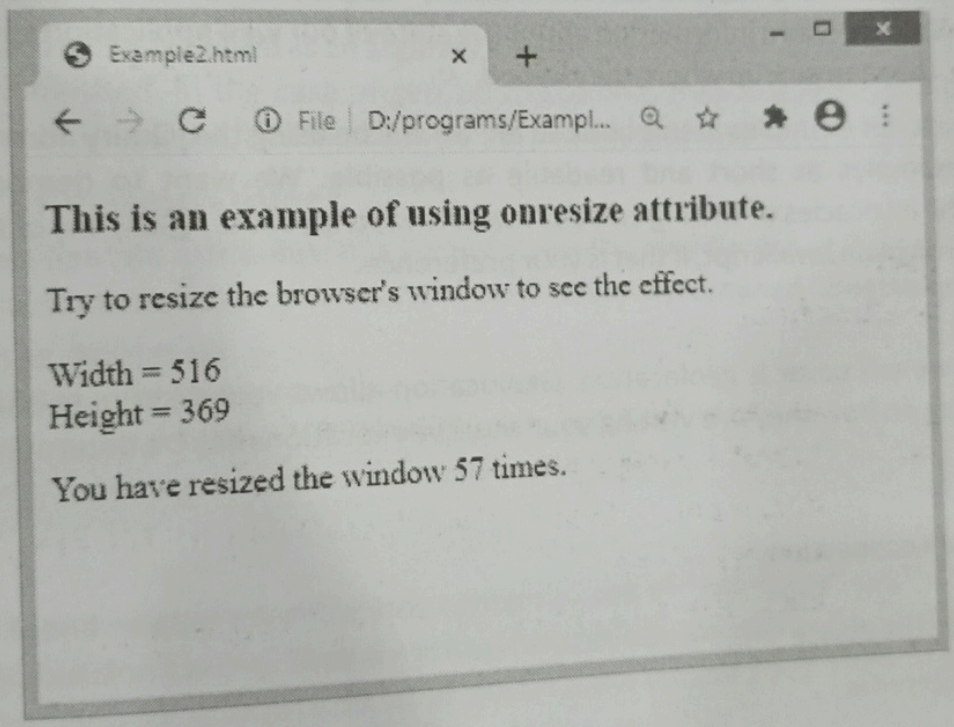
```
<html>
    <script>
        var i = 0;
        function fun()
        {
            var res = "Width = " + window.outerWidth +
            "<br>" + "Height = " + window.outerHeight;
            document.getElementById("para").innerHTML = res;
            var res1 = i += 1;
            document.getElementById("s1").innerHTML = res1;
        }
    </script>
    <body>
        <h3> This is an example of using onresize attribute. </h3>
        <p> Try to resize the browser's window to see the effect. </p>
        <p id = "para"></p>
        <p> You have resized the window
        <span id = "s1"> 0 </span> times.</p>
    </body>
</html>
```

Output

After the execution of the above code, the output will be -



When we try to resize the window, the output will be -



CHAPTER 9

GEOLOCATION

Much of what is loosely considered to be a part of "HTML5" is not, strictly speaking, HTML at all ,it is a set of additional APIs that provide a wide variety of tools to make our websites even better. An API is an interface for programs. So, rather than a visual interface where a user clicks on a button to make something happen, an API gives your code a virtual "button" to press, in the form of a method it calls that gives it access to a set of functionality. In this chapter, we will walk you through a few of the most useful of these APIs, as well as give you a brief overview of the others, and point you in the right direction should you want to learn more.

With these APIs, we can find a visitor's current location, make our website available offline as well as perform faster online, and store information about the state of our web application so that when a user returns to our site, they can pick up where they left off.

As with all the JavaScript examples in this book so far, we will be using the jQuery library in the interests of keeping the examples as short and readable as possible. We want to demonstrate the APIs themselves, not the intricacies of writing cross-browser JavaScript code. Again, any of this code can just as easily be written in plain JavaScript, if that is your preference.

Geolocation

The first new API we will cover is geolocation. Geolocation allows your visitors to share their current location. Depending on how they are visiting your site, their location may be determined by any of the following:

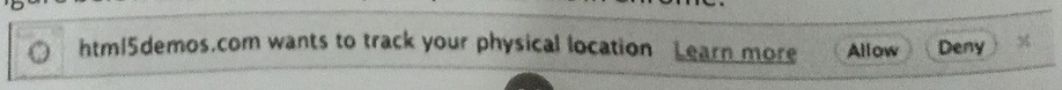
- IP address
- Wireless network connection
- Cell tower
- GPS hardware on the device

Geolocation is supported in:

- Safari 5+
- Chrome 5+
- Firefox 3.5+
- IE 9+
- Opera 10.6+
- iOS (Mobile Safari) 3.2+
- Android 2.1+

Privacy Concerns

Not everyone will want to share their location with you, as there are privacy concerns inherent to this information. Thus, your visitors must opt in to share their location. Nothing will be passed along to your site or web application unless the user agrees. The decision is made via a prompt at the top of the browser. Figure below shows what this prompt looks like in Chrome.



Geolocation Methods

With geolocation, you can determine the user's current position. You can also be notified of changes to their position, which could be used, for example, in a web application that provided real-time driving directions. These different tasks are controlled through the three methods currently available in the Geolocation API:

- Getcurrentposition
- Watchposition
- Clearposition

Retrieving the Current Position

The get Current Position method takes one, two, or three arguments. Here is a summary of the method's definition from the W3C's Geolocation API specification:

```
void getCurrentPosition(successCallback, errorCallback, options);
```

Only the first argument, successCallback, is required.successCallback is the name of the function you want to call once the position is determined.

A callback is a function that is passed as an argument to another function. A callback is executed after the parent function is finished. In the case of getCurrentPosition, thesuccessCallback will only run once getCurrentPosition is completed, and the location has been determined

Geolocation's Position Object

The Position object has two attributes: one that contains the coordinates of the position (coords), and another that contains the timestamp of when the position was determined (timestamp):

```
interface Position {
  readonly attribute Coordinates coords;
  readonly attribute DOMTimeStamp timestamp;
};
```

Interfaces

The HTML5, CSS3, and related specifications contain plenty of "interfaces" like the above. These can seem scary at first, but don't worry. They are just summarized descriptions of everything that can go into a certain property, method, or object. Most of the time the meaning will be clear and if not, they are always accompanied by textual descriptions of the attributes.

But where are the latitude and longitude stored? They are inside the Coordinates object. The Coordinates object is also defined in the W3C Geolocation spec, and here are its attributes:

```
interface Coordinates {
  readonly attribute double latitude;
  readonly attribute double longitude;
  readonly attribute double? altitude;
  readonly attribute double accuracy;
  readonly attribute double? altitudeAccuracy;
  readonly attribute double? heading;
  readonly attribute double? speed;
};
```

The question mark after double in some of those attributes simply means that there is no guarantee that the attribute will be there. If the browser cannot obtain these attributes, their value will be null. For example, very few computers or smartphones contain an altimeter so most of the time you would not receive an altitude value from a geolocation call. The only three attributes that are guaranteed to be there are latitude, longitude, and accuracy.

latitude and longitude are self-explanatory, and give you exactly what you would expect: the user's latitude and longitude. The accuracy attribute tells you, in meters, how accurate is the latitude and longitude information.

The altitude attribute is the altitude in meters, and the altitude Accuracy attribute is the altitude's accuracy, also in meters.

The heading and speed attributes are only relevant if we are tracking the user across multiple positions. These attributes would be important if we were providing real-time biking or driving directions, for example. If present, heading tells us, in degrees, the direction the user is moving in relation to true north. And speed, if present, tells us how quickly the user is moving in meters per second.

CHAPTER 10

CANVAS, SVG, AND DRAG AND DROP

Canvas

With HTML5's Canvas API, we're no longer limited to drawing rectangles on our sites. We can draw anything we can imagine, all through JavaScript. This can improve the performance of our websites by avoiding the need to download images off the network. With canvas, we can draw shapes and lines, arcs and text, gradient sand patterns. In addition, canvas gives us the power to manipulate pixels in images and even video.

The Canvas 2D Context spec is supported in:

- Safari 2.0+
- Chrome 3.0+
- Firefox 3.0+
- Internet Explorer 9.0+
- Opera 10.0+
- iOS (Mobile Safari) 1.0+
- Android 1.0+

A Bit of Canvas History

Canvas was first developed by Apple. Since they already had a framework, Quartz2D , for drawing in two-dimensional space, they went ahead and based many of the concepts of HTML5's canvas on that framework. It was then adopted by Mozilla and Opera, and then standardized by the WHATWG (and subsequently picked up by the W3C, along with the rest of HTML5).

Creating a canvas Element

The first step to using canvas is to add a canvas element to the page:

```
<canvas>
Sorry! Your browser doesn't support Canvas.
</canvas>
```

The text in between the canvas tags will only be shown if the canvas element is not supported by the visitor's browser.

Since drawing on the canvas is done using JavaScript, we will need a way to grab the element from the DOM. We'll do so by giving our canvas an id:

canvas/demo1.html *(excerpt)*

```
<canvas id="myCanvas">
Sorry! Your browser doesn't support Canvas.
</canvas>
```

All drawing on the canvas happens via JavaScript, so let us make sure we are calling a JavaScript function when our page is ready. We will add our jQuery document ready check to a script element at the bottom of the page:

canvas/demo1.html *(excerpt)*

```
<script>
$('document').ready(function(){
  draw();
});
</script>
```

The canvas element takes both a width and height attribute, which should also be set:

canvas/demo1.html *(excerpt)*

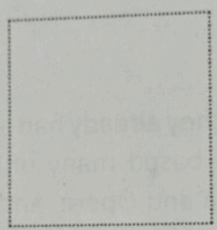
```
<canvas id="myCanvas" width="200" height="200">
Sorry! Your browser doesn't support Canvas.
</canvas>
```

Finally, let us add a border to our canvas to visually distinguish it on the page, using some CSS. Canvas has no default styling, so it is difficult to see where it is on the page unless you give it some kind of border:

css/canvas.css *(excerpt)*

```
#myCanvas {
  border: dotted 2px black;
}
```

Now that we have styled it, we can actually view the canvas container on our page ,Figure shows what it looks like.



An empty canvas with a dotted border

Drawing on the Canvas

All drawing on the canvas happens via the Canvas JavaScript API. We have called a function called draw () when our page is ready, so let us go ahead and create that function. We will add the function to our script element. The first step is to grab hold of the canvas element on our page:

canvas/demo1.html *(excerpt)*

```
<script>
:
function draw() {
  var canvas = document.getElementById("myCanvas");
}
</script>
```

Filling Our Brush with Color

On a regular painting canvas, before you can begin, you must first saturate your brush with paint. In the HTML5 Canvas, you must do the same, and we do so with the stroke Style or fill Style properties. Both strokeStyle and fillStyle a reset on a context object. And both take one of three values: a string representing a color, a CanvasGradient, or a CanvasPattern.

Let us start by using a color string to style the stroke. You can think of the stroke as the border of the shape you are going to draw. To draw a rectangle with a red border, we first define the stroke color:

canvas/demo1.html *(excerpt)*

```
function draw() {
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.strokeStyle = "red";
}
```

To draw a rectangle with a red border and blue fill, we must also define the fill color:

canvas/demo1.html *(excerpt)*

```
function draw() {
  ⋮
  context.fillStyle = "blue";
}
```

We can use any CSS color value to set the stroke or fill color, as long as we specify it as a string: a hexadecimal value like#00FFFF, a color name like red or blue, oran RGB value like rgb(0, 0, 255). We can even use RGBA to set a semi transparent stroke or fill color. Let us change our blue fill to blue with a 50% opacity:

```
}
  context.fillStyle = "rgba(0, 0, 255, 0.5)";
  ⋮
function draw() {
```

canvas\demo1.html (excerpt)

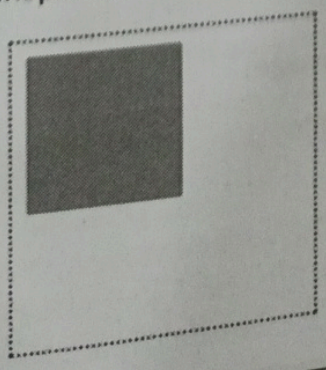
Drawing a Rectangle to the Canvas

Once we have defined the color of the stroke and the fill, we are ready to actually start drawing! Let us begin by drawing a rectangle. We can do this by calling the fillRect and strokeRect methods. Both of these methods take the X and Y coordinates where you want to begin drawing the fill or the stroke, and the width and the height of the rectangle. We will add the stroke and fill 10 pixels from the top and 10 pixels from the left of the canvas's top left corner:

canvas/demo1.html *(excerpt)*

```
function draw() {
  ⋮
  context.fillRect(10,10,100,100);
  context.strokeRect(10,10,100,100);
}
```

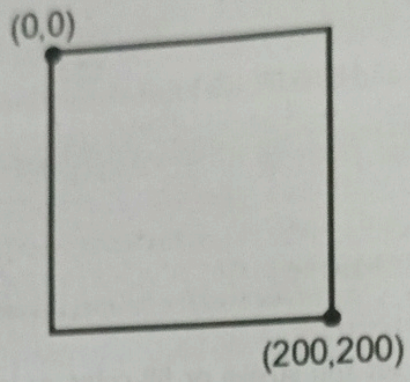
This will create a semi-transparent blue rectangle with a red border, like the one in Figure:



A simple rectangle—not bad for our first canvas drawing

The Canvas Coordinate System

As you may have gathered, the coordinate system in the canvas element is different from the Cartesian coordinate system you learned in math class. In the canvas co-ordinate system, the top-left corner is (0,0). If the canvas is 200 pixels by 200 pixels, then the bottom-right corner is (200, 200), as Figure below illustrates.



The canvas coordinate system goes top-to-bottom and left-to-right

Drawing an Image to the Canvas

We can also draw images into the canvas element. In this example, we will be redrawing into the canvas an image that already exists on the page. For the sake of illustration, we will be using the HTML5 logo4as our image for the next few examples. Let us start by adding it to our page in an img element:

canvas/demo6.html *(excerpt)*

```
<canvas id="myCanvas" width="200" height="200">
Your browser does not support canvas.
</canvas>
<img src="../images/html5-logo.png" id="myImageElem">
```

Next, after grabbing the canvas element and setting up the canvas's context, we can grab an image from our page via document.getElementById

canvas/demo6.html *(excerpt)*

```
function draw() {
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  var image = document.getElementById("myImageElem");
}
```

We will use the same CSS we used before to make the canvas element visible:

css/canvas.css *(excerpt)*

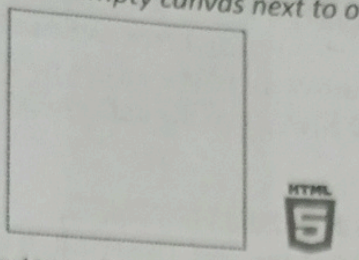
```
#myCanvas {
  border: dotted 2px black;
}
```

Let us modify it slightly to space out our canvas and our image:

css/canvas.css *(excerpt)*

```
#myCanvas {
  border: dotted 2px black;
  margin: 0px 20px;
}
```

Figure below shows our empty canvas next to our image:

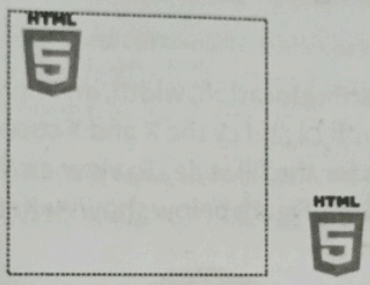


We can use canvas's drawImage method to redraw the image from our page into the canvas:

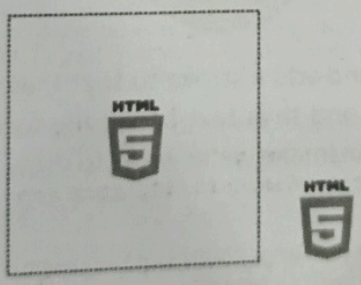
canvas/demo6.html *(excerpt)*

```
function draw() {
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  var image = document.getElementById("myImageElem");
  context.drawImage(image, 0, 0);
}
```

Because we have drawn the image to the (0,0) coordinate, the image appears in the top-left of the canvas, as you can see in Figure below :



We could instead draw the image at the center of the canvas, by changing the X and Y coordinates that we pass to drawImage. Since the image is 64 by 64 pixels, and the canvas is 200 by 200 pixels, if we draw the image to (68, 68),5 the image will be in the center of the canvas, as in Figure below



Accessibility Concerns

A major downside of canvas in its current form is its lack of accessibility. The canvas doesn't create a DOM node, is not a text-based format, and is thus essentially invisible to tools like screen readers. The HTML5 community is aware of these failings, and while no solution has been finalized, debates on how canvas could be changed to make it accessible are under-way.

SVG

SVG stands for Scalable Vector Graphics. SVG is a specific file format that allows you to describe vector graphics using XML. A major selling point of vector graphics in general is that, unlike bitmap images (such as GIF, JPEG,PNG, and TIFF), vector images preserve their shape even as you blow them up or shrink them down. We can use SVG to do many of the same tasks we can do with canvas, including drawing paths, shapes, text, gradients, and patterns.

Basic SVG, including using SVG in an HTML img element, is supported in:

- Safari 3.2+
- Chrome 6.0+
- Firefox 4.0+
- Internet Explorer 9.0+
- Opera 10.5+

There is currently no support for SVG in Android's browser.

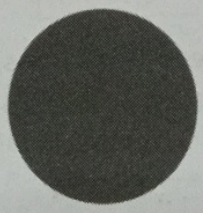
XML: XML stands for eXtensible Markup Language. Like HTML, it is a markup language, which means it is a system meant to annotate text. Just as we can use HTML tags to wrap our content and give it meaning, so can XML tags be used to describe the content of files.

Unlike canvas, images created with SVG are available via the DOM. This allows technologies like screen readers to see what is present in an SVG object through its DOM node and it also allows you to inspect SVG using your browser's developer tools. Since SVG is an XML file format, it is also more accessible to search engines than canvas.

images/circle.svg

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 400 400">
  <circle cx="50" cy="50" r="25" fill="red"/>
</svg>
```

The viewBox attribute defines the starting location, width, and height of the SVG image. The circle element defines a circle, with cx and cy the X and Y coordinates of the center of the circle. The radius is represented byr, while fill is for the fill style. To view an SVG file, you simply open it via the File menu in any browser that sup-ports SVG. Figure below shows what our circle looks like:



We can also draw rectangles in SVG, and add a stroke to them, as we did with canvas. This time, let us take advantage of SVG being an XML and thus text-based file format, and utilize the desc tag, which allows us to provide a description for the image we're going to draw:

images/rectangle.svg *(excerpt)*

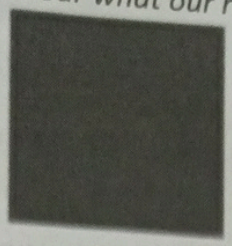
```
<svg xmlns="http://www.w3.org/2000/svg" viewbox="0 0 400 400">
  <desc>Drawing a rectangle</desc>
</svg>
```

Next, we populate the rect tag with a number of attributes that describe the rectangle. This includes the X and Y coordinate where the rectangle should be drawn, the width and height of the rectangle, the fill, the stroke, and the width of the stroke:

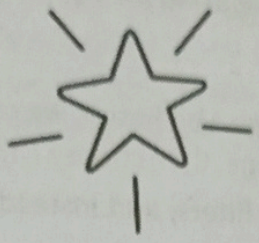
images/rectangle.svg

```
<svg xmlns="http://www.w3.org/2000/svg" viewbox="0 0 400 400">
  <desc>Drawing a rectangle</desc>
  <rect x="10" y="10" width="100" height="100"
        fill="blue" stroke="red" stroke-width="3"  />
</svg>
```

Figure below shows our what our rectangle looks like.



Unfortunately, It is not always this easy. If you want to create complex shapes, the code begins to look a little scary. Figure below shows a fairly simple-looking star image from openclipart.org



A line drawing of a star

And here are just the first few lines of SVG for this image:

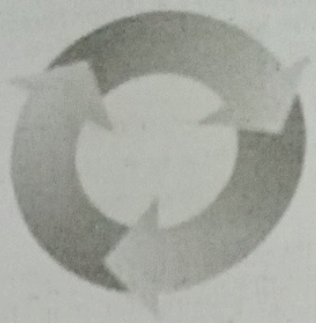
```
<svg xmlns="http://www.w3.org/2000/svg"
  width="122.88545" height="114.88568">
<g
  inkscape:label="Calque 1"
  inkscape:groupmode="layer"
  id="layer1"
  transform="translate(-242.42282,-449.03699)">
  <g
    transform="matrix(0.72428496,0,0,0.72428496,119.87078,183.8127)"
    id="g7153">
    <path
      style="fill:#ffffff;fill-opacity:1;stroke:#000000;stroke-width
➥:2.761343;stroke-linecap:round;stroke-linejoin:round;stroke-miterl
➥imit:4;stroke-opacity:1;stroke-dasharray:none;stroke-dashoffset:0"
      d="m 249.28667,389.00422 -9.7738,30.15957 -31.91999,7.5995 c -
➥2.74681,1.46591 -5.51239,2.92436 -1.69852,6.99979 l 30.15935,12.57
➥796 -11.80876,32.07362 c -1.56949,4.62283 -0.21957,6.36158 4.24212
➥,3.35419 l 26.59198,-24.55691 30.9576,17.75909 c 3.83318,2.65893 6
➥.12086,0.80055 5.36349,-3.57143 l -12.10702,-34.11764 22.72561,-13
➥.7066 c 2.32805,-1.03398 5.8555,-6.16054 -0.46651,-6.46042 l -33.5
➥0135,-0.66887 -11.69597,-27.26175 c -2.04282,-3.50583 -4.06602,-7.
➥22748 -7.06823,-0.1801 z"
      id="path7155"
      inkscape:connector-curvature="0"
      sodipodi:nodetypes="ccccccccccccccc" />
```

Using Inkscape to Create SVG Images

To save ourselves some work (and sanity), instead of creating SVG images by hand, we can use an image editor to help. One open source tool that you can use to make SVG images is Inkscape. Inkscape is an open source vector graphics editor that outputs SVG. Inkscape is available for download at http://inkscape.org.

For our progress-indicating spinner, instead of starting from scratch, we have searched the public domain to find a good image from which to begin. A good resource to know about for public domain images is http://openclipart.org, where you can find images that are copyright-free and free to use. The images have been donated by the creators for use in the public domain, even for commercial purposes, without the need to ask for permission.

We will be using an image of three arrows as the basis of our progress spinner, shown in Figure below. The original can be found at openclipart.org.10



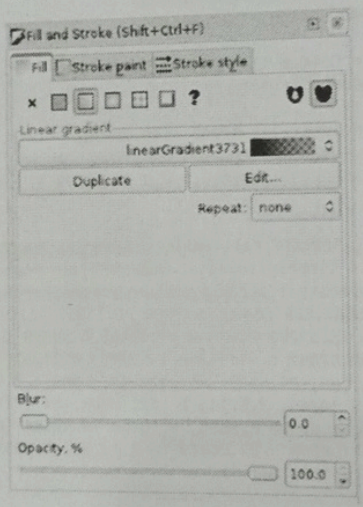
The image we'll be using for our progress indicator

SVG Filters

To make our progress spinner match our page a bit better, we can use a filter in Inkscape to make it black and white. Start by opening the file in Inkscape, then choose Filters>Color>Moonarize.

A safer approach would be to avoid using filters, and instead simply modify the color of the original image.

We can do this in Inkscape by selecting the three arrows in the spinner.svg image, and then selecting Object>Fill and Stroke. The Fill and Stroke menu will appear on the right-hand side of the screen, as seen in Figure below:



Modifying color using Fill and Stroke

From this menu, we can choose to edit the existing linear gradient by clicking the Edit button. We can then change the Red ,Green, and Blue values all to 0 to make our image black and white. We have saved the resulting SVG as spinnerBW.svg.

Canvas versus SVG

Now that we have learned about canvas and SVG, you may be asking yourself, which is the right one to use? The answer is: it depends on what you are doing. Both canvas and SVG allow you to draw custom shapes, paths, and fonts. But what is unique about each?

Canvas allows for pixel manipulation, as we saw when we turned our video from color to black and white. One downside of canvas is that it operates in what is known as immediate mode. This means that if you ever want to add more to the canvas, you cannot simply add to what is already there. Everything must be redrawn from scratch each time you want to change what is on the canvas. There is also no access to what's drawn on the canvas via the DOM. However, canvas does allow you to save the images you create to a PNG or JPEG file.

By contrast, what you draw to SVG is accessible via the DOM, because its mode is retained mode, the structure of the image is preserved in the XML document that describes it. SVG also has, at this time, a more complete set of tools to help you work with it, like the Inkscape.

However, since SVG is a file format, rather than a set of methods that allows you to dynamically draw on a surface, you cannot manipulate SVG images the way you can manipulate pixels on canvas. It would have been impossible, for example, to use SVG to convert our color video to black and white as we did with canvas.

In summary, if you need to paint pixels to the screen, and have no concerns about the ability to retrieve and modify your shapes, canvas is probably the better choice. If, on the other hand, you need to be able to access and change specific aspects of your graphics, SVG might be more appropriate. It is also worth noting that neither technology is appropriate for static images, at least not while browser support remains a stumbling block. In this chapter, we have made use of canvas and SVG for a number of such static examples, which is fine for the purpose of demonstrating what they can do. But in the real world, they are only really appropriate for cases where user interaction defines what is going to be drawn.