HTML 5–Teachers Notes.

(Level 1)

(The material I was looking forward when I wanted to learn HTML5)

Level 1 – For Beginners To Intermediate

Learn HTML5 - UseCases.

Course Contents

Day 1:

Introduction:

Overview of HTML5
History of HTML5
The Myth of 2022 and Why It Doesn't Matter
Who Is Developing HTML5?
A New Vision
Compatibility and Paving the Cow Paths
Utility and the Priority of Constituencies
Interoperability Simplification
Universal Access
A Plugin–Free Paradigm
What's In and What's Out?
What's New in HTML5?
New DOCTYPE and Character Set
New and Deprecated Elements
Semantic Markup
Simplifying Selection Using the Selectors API
JavaScript Logging and Debugging window JSON
DOM Level 3
Using WAI-ARIA with HTML5 for Accessibility

The first thing to realize about HTML5 is, it's not a single entity. It's comprised of many elements, including the fifth revision of HTML, CSS3 and many JavaScript API's.

It allows you to use the multimedia experience of the desktop on the web. Prior to this technology, experiences of this kind could only work on the desktop.

With HTML5, developers can create apps and websites that function like desktop applications, which allows you to use the web platform to reach all of your users at once. Users no longer have to download apps for multiple devices, They can start an app by clicking on a link or button and not have to worry about having the latest update.

An interesting aspect of HTML5 is that it will allow you to create apps that function even when not connected, or when your system is offline. The trick is being able to store the assets and content locally. When this is done, the app works, regardless of where you go or if your system is online.

Another one of the positive aspects of the offline features is being able to store data in the cache or in such a way that allows the data to be retained even if the page is reloaded.

With HTML5, you can make use of a wide variety of graphics elements, such as animation, games, movies, etc. Even intense graphics effects such as lightning and shadows, 3D, special effects, vector graphics and so on are supported.

A major advance is with the JavaScript engines, which are fast enough to run these applications in real time.

Hardware accelerated rendering is being used in modern browsers to create smooth rendering and transitions. What this means is that browsers are using the GPU (Graphics Processing Unit) to speed up computing tasks, which will improve the user experience.

HTML5 tags are backwards compatible so that HTML 4-based content won't destroy HTML5 content. The former HTML structure and formatting is retained.

Several new tags have been added, some of which are: <audio> is used for sound, <video> for video playback, <canvas> for dynamic graphics.

HTML5 allows SVG and MathML graphics to be embedded inline or linked within the HTML content. The result is that developers can include complex imagery without images, nor do they need to rely on third party platforms. All the content can be in one HTML file. It's also possible work with SVG imagery using JavaScript and manipulate aspects of the images, as well.

With HTML5 ,you can add audio directly to a web page. Be aware that while you can control the element with HTML or JavaScript, the specification doesn't cover which the types of codecs supported and this will vary with each browser. Before beginning your HTML5 development, this needs to be checked.

HTML5 allows you to embed video directly into a web page. Be aware that while the video element has been standardized, how the video codec is supported varies with each browser, so it's important to check that as part of your development process.

New types of form elements have been included, which allow you to display a given type of input then rely upon the browser to execute it. Some of the input options are: Telephone number, email, URL, date, time, color along with different variations.

The Canvas element gives you dynamic script-able graphics rendering in the

browser. With the use of JavaScript, you have the ability to control colors, vectors and pixels on your screen and this works with simple illustrations all the way up to complex visual effects.

Semantic markup tags are another advance which allow to structure your content so that the structure has meaning, sometimes known as semantics. Examples of this are the: <article> , <section> , <header> , <footer> , <aside> , <nav> , and <figure> tags. This allows you to place important elements into sections. It's also useful for search engines and aggregators.

The DOCTYPE has been simplified. The result is that HTML5 documents are valid XML structures. HTML5 content is XML based which means that it must follow XML formatting rules.

It's important to realize that while HTML5 isn't standardized it is in use on devices such as the Apple iOS, Android, BlackBerry 6, and HP/Palm WebOS devices. All of the major browsers (Safari, Chrome, Opera, Internet Explorer) support HTML5, but it's not equal across platforms, so it's important to test out the features on the different devices, platforms, etc. To make sure that you're obtaining the results you seek, it would be wise to get feedback from users during the course of development.

Accessible Rich Internet Applications specification [WAI-ARIA-1.1],
which defines a way to make Web content and Web applications more accessible to people with disabilities.
https://www.w3.org/TR/using-aria/ rules to be followed .

CANVAS API:

Overview of HTML5 Canvas
Canvas Coordinates& registering the Canvas dimensions
Drawing on Cavas with paths,curves etc.
Working with Solid colors,Gradients & Transparancy
Importing External Images & Setting the background
Working with Color & Geometrical transformations
Creating text,graphs & charts
Animating a Vertical Bar-Chartwith fine tuning.
Working with Pixel Data
CSS and Canvas
Create High-Res, Retina-Display-Ready Media with Canvas
Clipping Cavas drawings & saving them to a file.
When Not to Use Canvas
Fallback Content
Implementing Canvas Security
Implementing techniquesfor Backward compatibility.

What is Canvas?

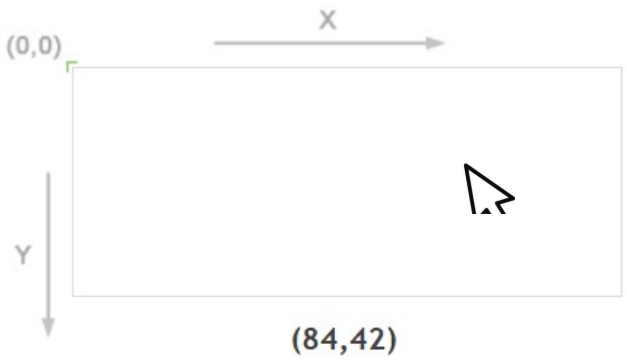
The HTML5 canvas element can be used to draw graphics on the webpage via

JavaScript. The canvas was originally introduced by Apple for the Mac OS dashboard widgets and to power graphics in the Safari web browser. Later it was adopted by the Firefox, Google Chrome and Opera. Now the canvas is a part of the new HTML5 specification for next generation web technologies.

By default the <canvas> element has 300px of width and 150px of height without any border and content. However, custom width and height can be defined using the CSS height and width property whereas the border can be applied using the CSS border property.

Understanding Canvas Coordinates

The canvas is a two-dimensional rectangular area. The coordinates of the top-left corner of the canvas are (0, 0) which is known as origin, and the coordinates of the bottom-right corner are (canvas width, canvas height). Here's a simple demonstration of canvas default coordinate system.

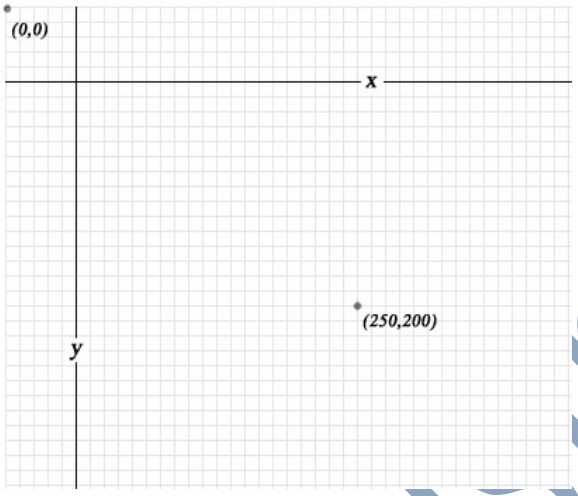


SVG	Canvas
Vector based (composed of shapes)	Raster based (composed of pixel)
Multiple graphical elements, which become the part of the page's DOM tree	Single element similar to <img> in behavior. Canvas diagram can be saved to PNG or JPG format
Modified through script and CSS	Modified through script only
Good text rendering capabilities	Poor text rendering capabilities
Give better performance with smaller number of objects or larger surface, or both	Give better performance with larger number of objects or smaller surface, or both

Better scalability. Can be printed with high quality at any resolution. Pixelation does not occur	Poor scalability. Not suitable for printing on higher resolution. Pixelation may occur

A webpage may contain multiple canvas elements. Each canvas may have an id using which you may target a specific canvas through JavaScript. Each canvas element has a 2D Context. This again has objects, properties, and methods. Tinkers these, you may draw your stuff. To draw on a canvas, you need to reference the context of the canvas. The context gives you access to the 2D properties and methods that We'll dive deeper into the context later.

Every canvas element has x and y coordinates. X being the horizontal coordinate and y being the vertical coordinate. The following image shows these coordinates on a canvas.



Canvas and Hardware Acceleration

With Hardware Acceleration enabled browsers, as a developer, you will be pleased enough. Because that renders the image/animations with the speed you desire your users should experience. Let's dive a bit deep.

Modern desktops have a GPU (Graphical Processing Unit) along with a CPU(Central Processing Unit). When it comes to delivering fast image/animation, the operation is taken care by GPU, CPU continues server for the rest of the task, resulting in accelerated graphics performance.

Chrome 27, Firefox 22, IE10, And Opera Next supports hardware acceleration and shows significant improve in rendering graphics.

Since JavaScript is the workhorse behind the Canvas, several performance tweaks can be used to the user experience by rendering image/animation faster.

Canvas API

Each canvas has its own context, so if your page contains multiple canvas elements; you must have a reference to each individual context that you want to work with.

Aside from getContext, there are plenty of other functions (functions of an object are called methods in

JavaScript) at your disposal in the canvas 2D API. Some of the notable ones are outlined below.

Transformation Functions :

- scale - allows you to scale the current context.
- rotate - allows you to rotate the x and y coordinates of the current context.

State Functions :

- save - allows you to save the current state of the context.
- restore - allows you to restore the state of the context from a previously saved state.

Text Functions

- font - gets or sets the font for the current context.
- fillText - renders filled text to the current canvas.
- measureText - measures the current width of the specified text.

Browser support and polyfills

Like so many other features of HTML5, you need to test whether Canvas is supported in the rendering browser. As of this writing, IE 9.0, 10.0 and 11.0, Firefox 2.0 to 26.0, Chrome 4.0 to 31.0, Safari 3.1 to 7.0, Opera 9.0 to 17.0, iOS Safari 3.2 to 7.0, Opera mini 5.0-7.0, Android Browser 2.1-4.3, Blackberry 7.0 and 10.0 and IE Mobile 10.0 supports Canvas basic. That covers most of the modern browsers you may understand well. If you think that you may have visitors from browsers not supporting Canvas, you may use any of the several **polyfills** available. Following is a list of name usage (typically you add those codes within head section of your HTML page) of some of the most used of those

flashcanvas

```
<!--[if lt IE 9]>

<script type="text/javascript" src="path/to/flashcanvas.js"></script>

<![endif]-->
```

Copy
You may download flashcanvas from http://flashcanvas.net/download

explorercanvas

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

Copy
You may download explorercanvas from http://code.google.com/p/explorercanvas/downloads/list

slcanvas

```
<script src="http://silverlightcanvas.appspot.com/static/slcanvas.js" ></script>
```

Copy
You may add this to the bottom of your HTML document.

You may download slcanvas from http://slcanvas.codeplex.com/releases/view/41025

fxcanvas

```
<script type="text/javascript" src="/public/path/jooscript.js"></script>

  <script type="text/javascript" src="/public/path/fxcanvas.js"></script>

  <!--[if IE]><script type="text/javascript"
src="/public/path/flash_backend.js"></script><![endif]-->

  <comment><script type="text/javascript"
src="/public/path/canvas_backend.js"></script></comment>
```

Copy
You may download fxcanvas from http://code.google.com/p/fxcanvas/downloads/detail?name=fxcanvas-0.2(beta4)-supersonic.zip

kineticjs

You may explore how to use kineticjs from http://kineticjs.com/docs/

You may download kineticjs from

```
<!DOCTYPE html>

<html>

<head>

<title>HTML5 Canvas Demo</title>

<style>

#FirstCanvas{

width: 500px;

height: 300px;

border: 3px solid green;

background-color: orange;

}

</style>

</head>
```

```
<body>

<canvas id="FirstCanvas"></canvas>

</body>

</html>
```

Copy

Ex1 - TEXT

```
<!DOCTYPE HTML>
<html>
<head>
<title>HTML5 Canvas Text example</title>
<script>
function CreateText() {
    var canvas=document.getElementById("rCanvas");
    var context=canvas.getContext("2d");

    var x = 100;
    var y = 100;
        context.font = 'bold 22pt Arial';//sets font style, size and type
        context.fillStyle = 'orange';//sets font color
    context.fillText("Welcome to html5-Canvas - CITI!!", x, y);//sets text to be rendered
}

</script>
</head>
<body onload="CreateText()">
    <canvas id="rCanvas" width="700" height="150"></canvas>
</body>
</html>
```

Ex 2:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML5 Canvas with transparency example</title>
<script>
  window.onload=function() {
    var rcanvas=document.getElementById("rCanvasTag");
    var rcontext=rcanvas.getContext('2d');
    rcontext.fillStyle='rgb(255,60,10,.50)';      //Sets the color used for filling an area
                                                  //the last parameter sets the transparency level
    rcontext.fillRect(0,0,100,100);
     //Fills a rectangle positioned at x and y, with a width and height of w and h.
    rcontext.fillStyle='rgb(0,0,255)';
    rcontext.fillRect(50,50,300,300);
    rcontext.fillStyle='rgb(0,255,0,0.5)';
    rcontext.fillRect(160,160,300,300);
  }
</script>
</head>
<body>
<canvas id="rCanvasTag" width="700" height="700"></canvas>
</body>
</html>
```

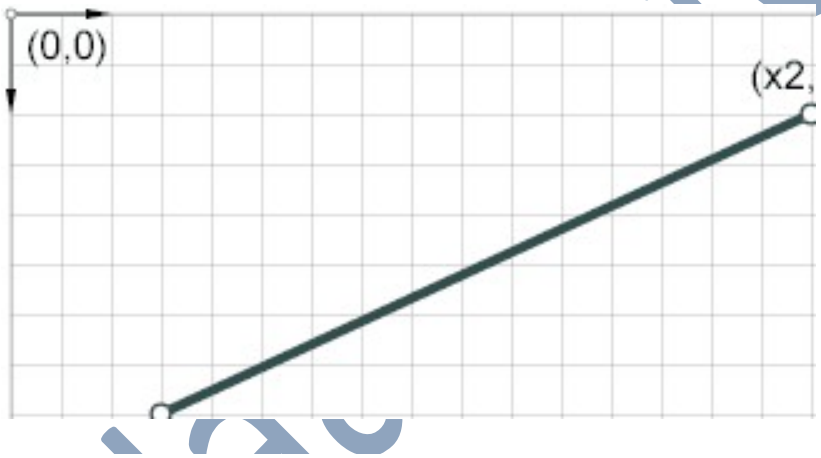
Ex 3 – Gradient

```
<!DOCTYPE html>
<html>
<head>
<title>HTML5 Canvas gradient example</title>
<script>
//Gradient 1 - RadialGradient
  window.onload=function() {
    var rcanvas=document.getElementById("rCanvasTag");
    var rcontext=rcanvas.getContext('2d');
    var rgradient=rcontext.createRadialGradient(300,300,0,300,300,300);
    rgradient.addColorStop("0","magenta");// Adds a color stop to a gradient. A color stop is a position in the
gradient where a color                          change occurs. The offset must be between 0 and 1.
    rgradient.addColorStop("1.0","red");
    rgradient.addColorStop(".75","yellow");
    rgradient.addColorStop(".50","green");
    rgradient.addColorStop(".25","blue");
    rcontext.fillStyle=rgradient;
    rcontext.fillRect(0,0,400,400);
    var mycontext1=document.getElementById("myCanvasTag1").getContext('2d');

     // Gradient 1 – LinearGradient
    var mygradient1=mycontext1.createLinearGradient(30,30,90,90);
    mygradient1.addColorStop(0,"#FF0000");
    mygradient1.addColorStop(1,"#00FF00");
    mycontext1.fillStyle=mygradient1;
    mycontext1.fillRect(0,0,100,100);
  }
</script>
</head>
<body>
<canvas id="rCanvasTag" width="400" height="400"></canvas>
<canvas id="myCanvasTag1" width="100" height="100" style="border: 10px blue solid">
</canvas>
</body>
</html>
```

Ex 4 – Image Drawing

```
<!DOCTYPE HTML>
<html>
  <head>
<title>Drawing image with Canvas</title>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
    </style>
  </head>
  <body>
    <canvas id="rCanvas" width="500" height="400"></canvas>
    <script>
      var canvas = document.getElementById('rCanvas');
      var context = canvas.getContext('2d');
      var imgObj = new Image();

      imgObj.onload = function() {
        context.drawImage(imgObj, 150, 78);
      };
      imgObj.src = 'https://via.placeholder.com/150/0000FF?Text=Digital.com';
    </script>
  </body>
</html>
```

Ex 5 – Draw Lines




```
<html>
<head>
<meta charset=utf-8 />
<title>Draw a line</title>
</head>
<body>
<canvas id="DemoCanvas" width="500" height="200"></canvas>
<script>
var canvas = document.getElementById('DemoCanvas');
//Always check for properties and methods, to make sure your code doesn't break in other browsers.
if (canvas.getContext)
 {
  var context = canvas.getContext('2d');
  // Reset the current path
  context.beginPath();
  // Staring point (10,45)
   context.moveTo(10,0); //x1,y1
  // End point (180,47)
```

```
  context.lineTo(0,147);//x2,y2
  // Make the line visible
context.lineWidth=3;
           context.strokeStyle = '#808000';
           context.lineCap = 'round';
  context.stroke();
   }
</script>
</body>
</html>
```

Ex 5 –

A Demo for creating a Net using lines example.

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>Draw a line</title>
</head>
<body>
<canvas id="DemoCanvas" width="400" height="400"></canvas>
<script>
var canvas = document.getElementById('DemoCanvas');
if (canvas.getContext)
 {

     for (i = 20; i < 200; i += 40)
         {
           var ctx = canvas.getContext("2d");
           ctx.moveTo(10, i);
           ctx.lineTo(canvas.width/2, i);
           ctx.lineWidth=14;
           ctx.strokeStyle='#0000FF';
           ctx.lineCap = 'round';
           ctx.stroke();
          }
     for (i = 20; i <200; i += 40)
         {
           var ctx = canvas.getContext("2d");
           ctx.moveTo(i, 10);
           ctx.lineTo(i,canvas.width/2);
           ctx.lineWidth=11;
           ctx.strokeStyle = '#808000';
             ctx.lineCap = 'round';
           ctx.stroke();
          }
  }
</script>
</body>
</html>


</html>
```

Ex 6- Drawing Lines

```
<html>
 <head>
 <meta charset=utf-8 />
 <title>Line Joining</title>
 </head>
 <body>
 <canvas id="DemoCanvas" width="400" height="400"></canvas>
```

```
<script>
var canvas = document.getElementById('DemoCanvas');
if (canvas.getContext)
{
   var ctx = canvas.getContext("2d");
   var lStart = 50;
   var lEnd = 200;
   var yStart = 20;
   ctx.beginPath();
   ctx.lineWidth = 25;
// Use a bevel corner.
 //  ctx.lineJoin = "bevel";
    ctx.lineJoin = "miter";
//    ctx.lineJoin = "round";
   ctx.moveTo(30, 20);
   ctx.lineTo(250, 20);
   ctx.lineTo(150, 120);
   ctx.stroke();
 }
</script>
</body>
</html>
```

Ex 7 – Drawing Arcs

```
<!DOCTYPE html>
<html>
<head>
<title>Sample arcs example</title></head>
<body>
<canvas id="demoCanvas" width="340" height="340"> canvas</canvas>
<script>
var canvas = document.getElementById('demoCanvas');
var ctx = canvas.getContext('2d');
var x = 150;
var y = 180;
var radius = 45;
//var startAngle = 1.1 *3.14;
//var endAngle = 1.9 * 3.14;
var startAngle=0;
var endAngle=2*3.14;
var counterClockwise = false;
ctx.beginPath();
ctx.arc(x, y, radius, startAngle, endAngle, counterClockwise);
ctx.lineWidth = 10;
// line color
ctx.strokeStyle = 'green';
ctx.stroke();
</script>
</body>
</html>
```

Ex 8 –

arcTo Method: arcTo(x1, y1, x2, y2, radius)

The arcTo() method creates an arc of radius between two tangents. The first tangent is defined by an imaginary line that is drawn through the last point in a path and the point (x1, y1). The second tangent is defined by an

imaginary line that is drawn through the point (x1, y1) and the point (x2, y2).

Syntax :
arcTo(x1, y1, x2, y2, radius);

```
<html>
<head>
<title>Sample arcs example</title></head>
<body>
<canvas id="myCanvas" width="300" height="600"></canvas>
<script>
var canvas = document.getElementById("myCanvas");
if (canvas.getContext)
{
var ctx = canvas.getContext("2d");
// Draw the imaginary tangents in blue.
ctx.beginPath();
ctx.lineWidth = "3";
ctx.strokeStyle = "black";
ctx.moveTo(80, 100);
ctx.lineTo(240, 100);
ctx.moveTo(200, 60);
ctx.lineTo(200, 220);
ctx.stroke();
// Draw it.
// Create two lines that have a connecting arc that could be used as a start to a rounded rectangle.
ctx.beginPath();
ctx.strokeStyle = "red";
ctx.lineWidth = "5";
ctx.moveTo(120, 100);   // Create a starting point.
ctx.arcTo(200, 100, 200, 220, 75); // Create an arc.
ctx.stroke();
}
</script>
</body>
</html>
```

Ex 9 - Drawing Bezier Curves

Originally developed by Pierre Bézier in the 1970's for CAD/CAM operations. You can draw bezier curves in the same way as you draw lines, but instead of using the lineTo() method, you use either the bezierCurveTo() method or quadraticCurveTo() method. bezierCurveTo() method connects the endpoints with a cubic bezier curve using a specified pair of control points and the quadraticCurveTo() method connects the endpoints with a quadratic bezier curve using a single specified control point.

bezierCurveTo() method

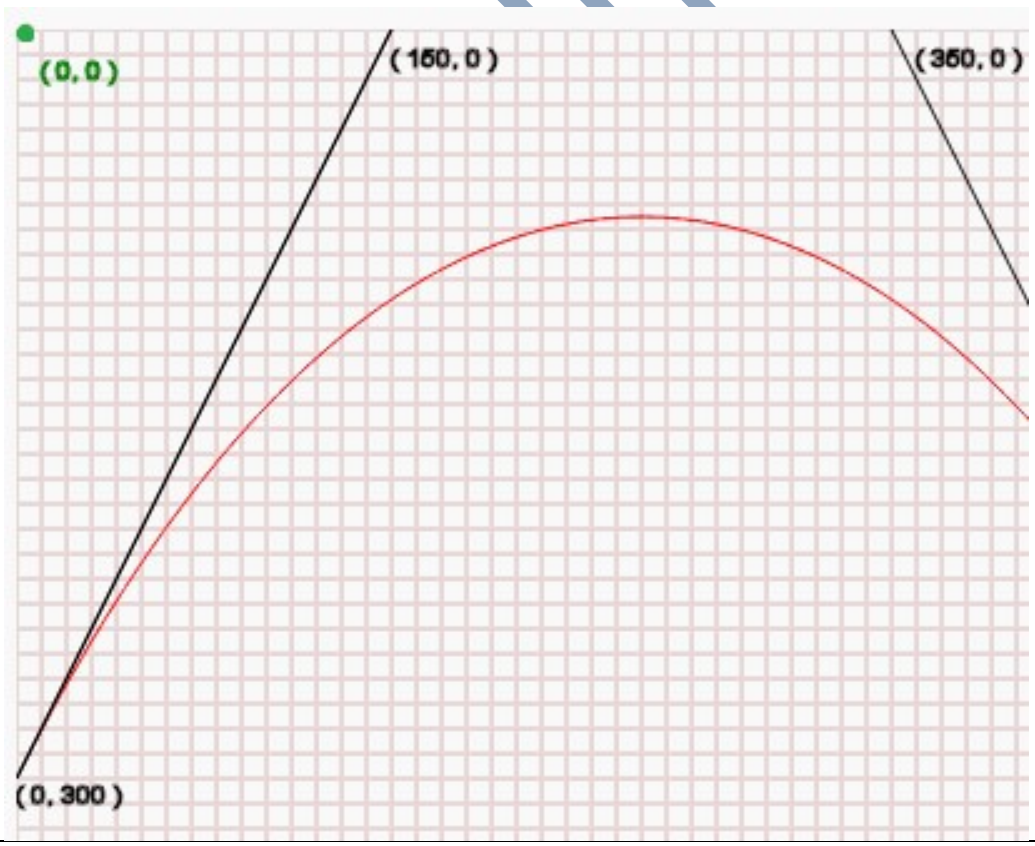
A cubic Bézier curve must include three points. The first two points (cp1x,cp1y) and (cp2x, cp2y) are control points that are used in the cubic Bézier calculation and the last point (x,y) is the ending point for the curve.

Syntax:
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

Parameters	Type	Description
cp1x	number	The x-coordinate of the first Bézier control point.
cp1y	number	The y-coordinate of the first Bézier control point.
cp2x	number	The x-coordinate of the second Bézier control point.
cp2y	number	The y-coordinate of the second Bézier control point.
x	number	The x-coordinate of the point to add to the current path.
y	number	The y-coordinate of the point to add to the current path.

Note : The first point on the curve is the last point in the existing current subpath. If a path does not exist, use the beginPath and moveTo methods to create a starting point.

See the following diagram :



In the above diagram :

- (0, 0) is the top-left position of the canvas.
- (0, 400) is the bottom-left position of the canvas.
- (0, 500) is the top-right position of the canvas.
- (400, 500) is the bottom-right position of the canvas.
- (0, 300) is the starting point of the curve.
- (150, 0) i.e. (cp1x, cp1y) is the first control position of the curve.
- (350, 0) i.e. (cp2x, cp2y) is the second control position of the curve.
- (500, 300) i.e. (x, y) is the ending point of the curve.

```
<!DOCTYPE html>
<html>
<head>
<title>Sample arcs example</title></head>
<body>
<canvas id="DemoCanvas" width="500" height="400"></canvas>
<script>
var canvas = document.getElementById("DemoCanvas");
var ctx = canvas.getContext("2d");
ctx.lineWidth = 7;
ctx.lineCap = "round";
ctx.clearRect(0, 0, canvas.width, canvas.height);
ctx.beginPath();
ctx.moveTo(30, 200);
ctx.lineTo(30, 50);
ctx.lineTo(65, 50);
ctx.moveTo(30, 120);
ctx.lineTo(65, 120);
ctx.lineTo(100, 200);
ctx.strokeStyle = "black";
ctx.stroke();
ctx.beginPath();
ctx.moveTo(65, 50);
ctx.bezierCurveTo(120, 50, 120, 120, 65, 120);
ctx.strokeStyle = "green";
ctx.stroke();
</script>
</body>
</html>
```

Ex 10 - quadraticCurveTo() method

A quadratic Bézier curve requires two points. The first point is a control point that is used in the quadratic Bézier calculation and the second point is the ending point for the curve.

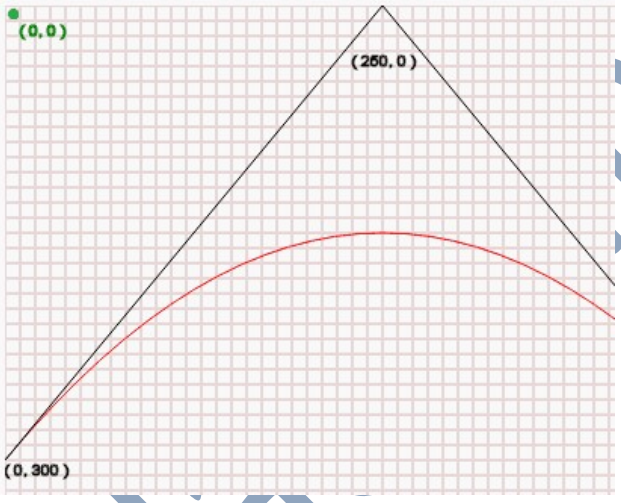
Syntax :

quadraticCurveTo(cp1x, cp1y, x, y);

Parameters	Type	Description
cp1x	number	The x-coordinate of the Bézier control point.
cp1y	number	The y-coordinate of the Bézier control point.
x	number	The x-coordinate of the point to add to the current path.
y	number	The y-coordinate of the point to add to the current path.

Note : The starting point for the curve is the last point in the existing current subpath. If a path does not exist, use beginPath() and moveTo() methods to set a starting point.

See the following diagram :



In the above diagram :

- (0, 0) is the top-left position of the canvas.
- (0, 400) is the bottom-left position of the canvas.
- (0, 500) is the top-right position of the canvas.
- (400, 500) is the bottom-right position of the canvas.
- (0, 300) is the starting point of the curve.
- (250, 0) i.e. (cp1x, cp1y) is the control position of the curve.

- (500, 300) i.e. (x, y) is the ending point of the curve.

```
<!DOCTYPE html>
<html>
<head>
<title>Sample arcs example</title></head>
<body>
<canvas id="DemoCanvas" width="500" height="400"></canvas>
<script>
var canvas = document.getElementById('DemoCanvas');
  if (canvas.getContext)
  {
    var ctx = canvas.getContext('2d');
    // Draw shapes
    ctx.beginPath();
    ctx.moveTo(75,25);
    ctx.quadraticCurveTo(25,25,25,62.5);
    ctx.quadraticCurveTo(25,100,50,100);
    ctx.quadraticCurveTo(50,120,30,125);
    ctx.quadraticCurveTo(60,120,65,100);
    ctx.quadraticCurveTo(125,100,125,62.5);
    ctx.quadraticCurveTo(125,25,75,25);
    ctx.lineWidth = "3";
    ctx.strokeStyle = "green";
    ctx.stroke();
  } else {
    alert('Not supported in this browser.');
  }
</script>
</body>
</html>
```

Ex: 11 – PATHS

Introduction

The canvas paths allow you to draw custom shapes. In HTML5 canvas, a path consists of a list of zero or more subpaths. There are one or more points in each subpath that are connected by straight lines or curves. To create a custom shape perform the following steps :

- Use beginPath() method to start drawing the path.
- Draw the path that makes a shape using lines, curves and other primitives.
- After creating the path, call fill() method to fills subpaths by using the current fill style or stroke() method to render the strokes of the current subpath by using the current stroke styles. Your shape will not be visible until you call fill() or stroke() methods.
- Now call closePath() method to close the current subpath and starts a new subpath that has a start point that is equal to the end of the closed subpath.

Note : If a subpath has fewer than two points, it is ignored when the path is painted.

Code:

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>Sample arcs example</title>
</head>
<body>
<canvas id="myCanvas" width="300" height="600"></canvas>
<script>
var canvas = document.getElementById("myCanvas");
if (canvas.getContext)
{
 var ctx = canvas.getContext('2d');
 ctx.beginPath();     // Start a new path.
 ctx.lineWidth = "3";
 ctx.strokeStyle = "green";  // This path is green.
 ctx.moveTo(0, 0);
 ctx.lineTo(160, 160);
 ctx.lineTo(200, 160);
 ctx.stroke();
 ctx.beginPath();
 ctx.strokeStyle = "blue";  // This path is blue.
 ctx.moveTo(0, 0);
 ctx.lineTo(50, 170);
 ctx.stroke();
 ctx.closePath(); // Close the current path.
}
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Sample arcs example</title>
</head>
<body>
<canvas id="DemoCanvas" width="300" height="600"></canvas>
<script>
var ctx = document.getElementById('DemoCanvas').getContext('2d');
ctx.fillStyle = "blue";
ctx.beginPath();
ctx.moveTo(30, 30);
ctx.lineTo(150, 150);
ctx.bezierCurveTo(60, 70, 60, 70, 70, 180);
ctx.lineTo(30, 30);
ctx.fill();
</script>
</body>
</html>

<!DOCTYPE html>
<html>
<head>
<title>Sample arcs example</title>
</head>
<body>
<canvas id="DemoCanvas" width="500" height="600"></canvas>
<script>
var canvas = document.getElementById('DemoCanvas');
if (canvas.getContext)
{
var ctx = canvas.getContext('2d');
```

```
ctx.lineWidth = 5;
ctx.beginPath();
ctx.moveTo(100, 20);
// line 1
ctx.lineTo(200, 160);
ctx.strokeStyle = 'red';
ctx.stroke();
// quadratic curve
ctx.beginPath();
ctx.moveTo(200,160);
ctx.quadraticCurveTo(230, 200, 260, 100);
ctx.strokeStyle = 'blue';
ctx.stroke();
// bezier curve
ctx.beginPath();
ctx.moveTo(260,100);
ctx.bezierCurveTo(290, -40, 300, 190, 400, 150);
ctx.strokeStyle = 'green';
ctx.stroke();
ctx.closePath();
}
</script>
</body>
</html>
```

Ex: 12- Saving and Cropping

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8" />
<title>Canvas Test</title>
</head>
<body>
<header> </header>
<nav> </nav>
<section>

<div>
<canvas id="canvas" width="320" height="200">
This text is displayed if your browser does not support HTML5 Canvas.
</canvas>
</div>

<script type="text/javascript">
var canvas;
var ctx;

function init() {
canvas = document.getElementById("canvas");
ctx = canvas.getContext("2d");
draw();
}


function draw() {

ctx.fillStyle = '#FA6900';
ctx.shadowOffsetX = 5;
ctx.shadowOffsetY = 5;
```

```
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(0,0,15,150);
ctx.save();

ctx.fillStyle = '#E0E4CD';
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(30,0,30,150);
ctx.save();

ctx.fillStyle = '#A7DBD7';
ctx.shadowOffsetX = 15;
ctx.shadowOffsetY = 15;
ctx.shadowBlur    = 4;
ctx.shadowColor   = 'rgba(204, 204, 204, 0.5)';
ctx.fillRect(90,0,45,150);
ctx.save();

ctx.restore();
ctx.beginPath();
ctx.arc(185, 75, 22, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();

ctx.restore();
ctx.beginPath();
ctx.arc(260, 75, 15, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();

ctx.restore();
ctx.beginPath();
ctx.arc(305, 75, 8, 0, Math.PI*2, true);
ctx.closePath();
ctx.fill();
}

init();
</script>
</section>
<aside> </aside>
<footer> </footer>
</body>
</html>
```

Security Concern of Canvas

There's a common point of confusion regarding when one can use HTML5 Canvas `getImageData()` and `toDataUrl()` methods. Certain operations will cause these methods to throw a security error instead of functioning normally.

The rules for what one can and cannot do are laid out in the spec, though the reasoning behind them isn't so obvious. The most typical violation is when a programmer uses `drawImage()` with an image that is from a different domain (than the page that the canvas is on) or an image that is on the local file system. When `drawImage()` is used in one of

these two ways, the canvas internally sets its *origin-clean* flag to false.

From the moment a canvas has its *origin-clean* flag set to false one is not allowed to use the `getImageData()` and `toDataUrl()` methods of that canvas, instead the security error will be thrown. There are a few more cases where the origin-clean flag will be set to false, you can read about them in the spec here.

The reason for this security is to prevent something called information leakage. To see why this is a security issue, consider the following hypothetical situation:

Say you are on a work network and so you have access to internal, private company sites and your (private!) hard-drive. You can accessThe private sites might be something like `www.internal.myCompany.com` and your hard drive would be accessible from urls like `file:///C:/SomeOfMyPhotos/`.

Now suppose you visited a website with a hidden canvas and while you were browsing the site that canvas was constantly calling drawImage() onto that canvas with urls that it was guessing might exist. These urls would be things like an image on the private subdomain:

```
www.internal.myCompany.com/secret/secret-plans.jpg
```

Or an image on your hard drive:

```
file:///C:/SomeOfMyPhotos/thatEmbarassingPhoto.png
```

The malicious site could keep trying different combinations of private-to-you urls until it found one that was actually a file. Then it would draw it to the canvas. Then it would get the imageData from the canvas and send it off to the server.

Voila! The malicious site owner now has your secret plans and your embarrassing photos, much without your consent.

Now we know that the above scenario is not very probable: In the real world, secret plans are almost always in PNG format whereas embarassing photos are almost universally in JPG format! But it stands that situations like the above could happen and so the security implications of canvas must take this into account.

Building an Application with HTML5 Canvas

SVG API :

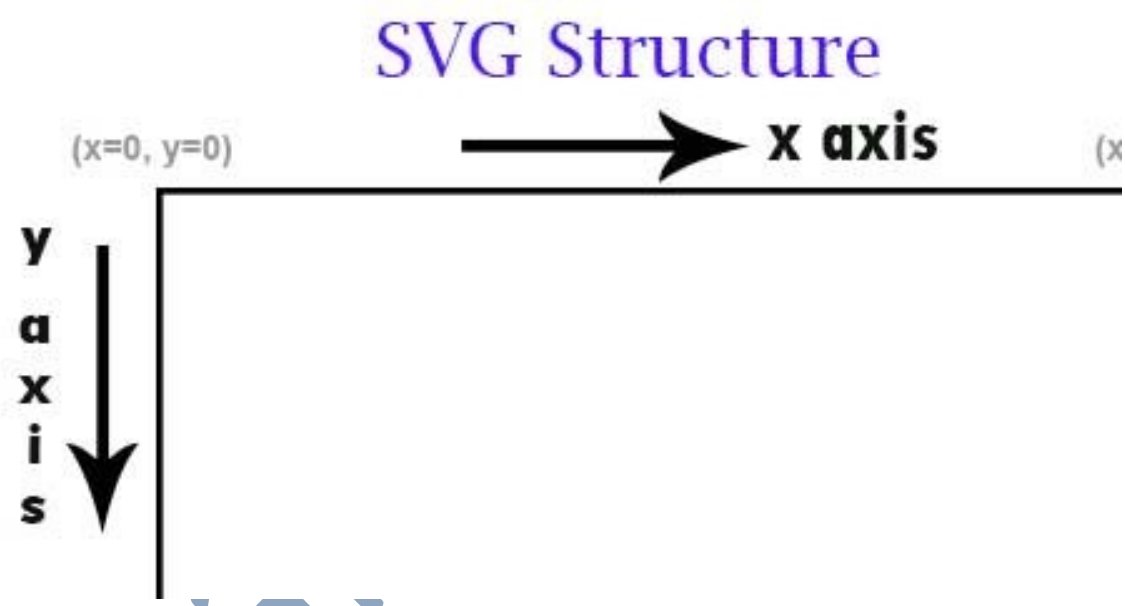
Understanding SVG
Creating 2D Graphics with SVG
Adding SVG to a Page
Simple Shapes & Text
Transforming SVG Elements
Reusing Content
SVG paths
Patterns and Gradients
Building an Interactive Application with SVG

Adding the CSS Styles
Implementing techniques for Backward compatibility.

SVG

HTML5 SVG*(Scalable Vector Graphics)* is the new way to add graphics on your Webpage. **SVG** can create **Vector based** drawing and objects like **lines**, **rectangle**,**circle**, **polygons**, **text** so on.

Unlike bitmap images(jpg, png and gif), they never stretch up or shrink down. They are even light weighted as compared to bitmap images.



SVG stands for Scalable Vector Graphics
Scalable
To be scalable means to increase or decrease uniformly. In terms of graphics, scalable means not being limited to a single, fixed, pixel size.
Vector
Vector graphics contain geometric objects such as lines and curves.
Graphics
Most existing XML grammars represent either textual information, or represent raw data such as financial information

The W3C has produced or is developing four inter related graphics and multimedia specifications.

a. Scalable Vector Graphics, SVG
b. Synchronized Multimedia Integration Language, SMIL,

SMIL Animation
d. SMIL version 2
e. VML

Salient Points.
A GIF image is limited to 256 colours or the 216 colours of the "web safe" palette. With SVG we are not limited in that way. If the monitor and software allows the display of 16.7 millions colors, SVG will provide that if we so choose.

1. SVG graphics are always stored as a set of SVG elements, just like any other XML document.
2. A SVG viewer is actually a rendering engine that interprets the vector description of an image, and renders it as a bitmap for display on screen.
3. The fact that the rendering of an SVG image can be carried out on a client machine dynamically, also means that it is possible to adjust certain parameters of the rendered image at the client's choice.
4. For example a SVG image can typically be zoomed in or out on screen and can be scrolled and panned to view particular areas in detail.

Definitions
Basic shape
Standard shapes which are predefined in SVG as a convenience for common graphical operations. Specifically:
Rect, circle, ellipse, line, polyline, polygon.
canvas
A surface onto which graphics elements are drawn, which can be real physical media such as a display or paper or an abstract surface such as a allocated region of computer memory
clipping path
A combination of path, text and basic shapes serve as the outline.

Drawing Basic Shapes

Ex 1

```
<svg width="400" height="400" style="border:1px solid #ccc" >
    <circle cx="55" cy="75" r="20" fill="purple"/>
    Ellipse
   <ellipse cx="75" cy="25" rx="20" ry="10" fill="orange"/>
    Line
   <line x1="55" y1="5" x2="45" y2="45" stroke="black"/> Rectangle
   <rect x="115" y="5" width="40" height="40" fill="cyan"/> Rounded Rectangle
   <rect x="225" y="5" rx="5" ry="5" width="40" height="40" fill="yellow"/>
    Use
       <defs>
           <rect id="rect" width="15" height="15" fill="blue"/>
       </defs>
       <use x="5"  y="5"  xlink:href="#rect"/>
       <use x="30" y="30" xlink:href="#rect"/>

   </svg>
```

Ex 2

Writing Text

```
Writing Text

Left Justification
<text x="0" y="13" fill="red" text-anchor="start">Text</text> Center
<text x="25" y="13" fill="red" text-anchor="middle">Text</text>
Right Justification
<text x="50" y="13" fill="red" text-anchor="end">Text</text> Multiple Lines
<text x="0" y="13" fill="red">    <tspan>Line 1</tspan>    <tspan x="0" dy="1em">Line 2</tspan></text>
Text on a Path
<defs>    <path id="textPath" d="M10 50 C10 0 90 0 90 50"/></defs><text fill="red">    <textPath
xlink:href="#textPath">Text on a Path</textPath></text>
```

Ex 3
Animation

```
Horizontal Animation
<rect y="45" width="10" height="10" fill="red">    <animate attributeName="x" from="0" to="90" dur="10s"
repeatCount="indefinite"/></rect>

 Vertical Animation
<rect x="45" width="10" height="10" fill="red">    <animate attributeName="y" from="0" to="90" dur="10s"
repeatCount="indefinite"/></rect>

 Diagonal Animation
<rect width="10" height="10" fill="red">    <animate attributeName="x" from="0" to="90" dur="10s"
repeatCount="indefinite"/>    <animate attributeName="y" from="0" to="90" dur="10s"
repeatCount="indefinite"/></rect>

Rotation Animation
<rect x="45" width="10" height="10" fill="red">    <animateTransform attributeName="transform" type="rotate"
from="0" to="90" dur="10s" repeatCount="indefinite"/></rect>

Animation on a Path
<rect x="-5" y="-5" width="10" height="10" fill="red">    <animateMotion path="M10 50 C10 0 90 0 90 50" dur="10s"
rotate="auto" repeatCount="indefinite"/></rect>
```

Ex 4
Events

```
Click Event
<rect x="5" y="5" width="40" height="40" fill="red">    <set attributeName="fill" to="blue"
begin="click"/></rect> Mousedown Event
<rect x="5" y="5" width="40" height="40" fill="red">    <set attributeName="fill" to="blue"
begin="mousedown"/></rect>
 Mouseup Event
<rect x="5" y="5" width="40" height="40" fill="red">    <set attributeName="fill" to="blue"
begin="mouseup"/></rect>
Mouseover Event
<rect x="5" y="5" width="40" height="40" fill="red">    <set attributeName="fill" to="blue"
begin="mouseover"/></rect>
Mouseout Event
<rect x="5" y="5" width="40" height="40" fill="red">    <set attributeName="fill" to="blue"
begin="mouseout"/></rect>

Mousemove Event
<rect x="5" y="5" width="40" height="40" fill="red">    <set attributeName="fill" to="blue"
begin="mousemove"/></rect>
```

Ex 5 - Key Events

Keydown Event

```
<rect width="100%" height="100%" fill="blue"    onkeydown="press(evt)"/><text id="text" x="25" y="27" text-anchor="middle">0</text>
```

Keyup Event

```
<rect width="100%" height="100%" fill="blue"    onkeyup="press(evt)"/><text id="text" x="25" y="27" text-anchor="middle">0</text>
```

Keypress Event

```
<rect width="100%" height="100%" fill="blue"    onkeypress="press(evt)"/><text id="text" x="25" y="27" text-anchor="middle">0</text>
```

Ex 6 – Gradient

```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%">

    <defs>
        <linearGradient id="Gradient01">
          <stop offset="20%" stop-color="#39F" />
          <stop offset="90%" stop-color="#F3F" />
        </linearGradient>
      </defs>

    <rect x="1cm" y="1cm" width="6cm" height="1cm"
          fill="url(#Gradient01)"  />
      <!-- Show outline of canvas using 'rect' element -->
      <rect x=".01cm" y=".01cm" width="7.98cm" height="2.98cm"
            fill="none" stroke="blue" stroke-width=".02cm" />
    </svg>
```

Ex 7 – SVG + Canvas

```
<html>
    <style>
        svg, img, canvas {
  display: block;
}
    </style>
     <body>
    <script>
 function display(){
var svg = document.querySelector('svg');
//var svg=document.getElementById('svg');
//console.log(svg);
var img = document.querySelector('img');
console.log(img);
var canvas = document.querySelector('canvas');

// get svg data

var xml=new XMLSerializer().serializeToString(svg);
// make it base64
var svg64 = btoa(xml);
var b64Start = '';

// prepend a "header"
var image64 = b64Start + svg64;
img.src = image64;
// set it as the source of the img element
img.onload = function() {
    // draw the image onto the canvas
    canvas.getContext('2d').drawImage(img, 0, 0);
```

```
}

}
    </script>
 <button onclick="display()">Click</button>
        SVG

<svg height="40">
  <rect width="40" height="40" style="fill:rgb(255,0,255);" />
  <circle cx="55" cy="75" r="20" fill="purple"/>
    Ellipse
   <ellipse cx="75" cy="25" rx="20" ry="10" fill="orange"/>
    Line
   <line x1="55" y1="5" x2="45" y2="45" stroke="black"/> Rectangle
   <rect x="115" y="5" width="40" height="40" fill="cyan"/> Rounded Rectangle
   <rect x="225" y="5" rx="5" ry="5" width="40" height="40" fill="yellow"/>
    Use
       <defs>
           <rect id="rect" width="15" height="15" fill="blue"/>
       </defs>
       <use x="5"  y="5"  xlink:href="#rect"/>
       <use x="30" y="30" xlink:href="#rect"/>

  <image xlink:href="https://en.gravatar.com/userimage/16084558/1a38852cf33713b48da096c8dc72c338.png?size=20"
height="20px" width="20px" x="10" y="10"></image>
</svg>
<hr/><br/>

IMAGE
<img/>
<hr/><br/>

CANVAS
<canvas></canvas>
<hr/><br/>
    </body>
</html>
```

SVG – Paths

Lines

This example moves the virtual pen to the point 50,50. The next drawing command will start from that point.

Drawing a line is probably the simplest command you can give the <path> element. Drawing lines is done using the L and l (lowercase L) commands. Here is an example:

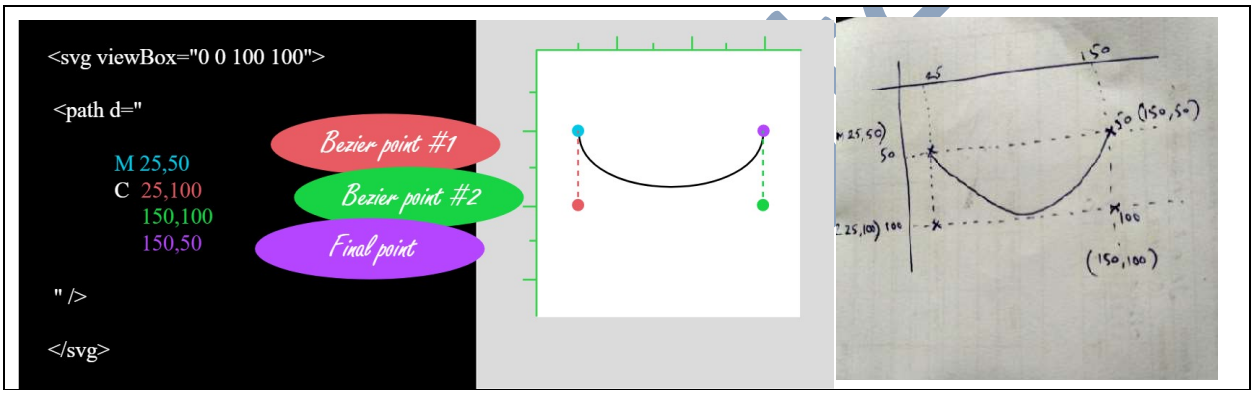
```
    <svg viewBox="0 0 500 500" xmlns="http://www.w3.org/2000/svg"
        xmlns:xlink="http://www.w3.org/1999/xlink">

        <path d="M50,50
                L100,200
                l125,0"
            style="stroke:#660000; fill:none;"/>
    </svg>
    Followed by a relative command:
```

Note: If the virtual pen is located at 50,50 and you use an l100,100 command, the line will be drawn to 50+100,50+100 = 150,150. Using an L100,100 command would have drawn the line to 100,100 exactly, regardless of the location of the virtual pen.

Ex 2

Just like the `M` and `m` commands, `L` and `l` take two numbers: either absolute or relative coordinates. There are four other commands that are essentially simpler versions of the line commands. They also draw lines, but only take one value: horizontal or vertical. When we used `l 25,0` we could have used `h 25` which means "from where the pen currently is, draw 25 to the right. More succinct, I suppose. It's bigger brother `H`, as we could guess, means to draw to the exact horizontal coordinate 25. `V` and `v` move vertically absolutely and relatively as you'd surely guess.

```
<svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">

<path d="  M 25,50 C 25,100  150,100  150,50 " style="stroke:#660000; fill:none;"/>
```




```
    <path d="M10 10 h10 v-10 h10 v10 h10 v10 h-10 v10 h-10 v-10 h-10 z"
          style="stroke:#FF0000; fill:none;"
          />
</svg>
```

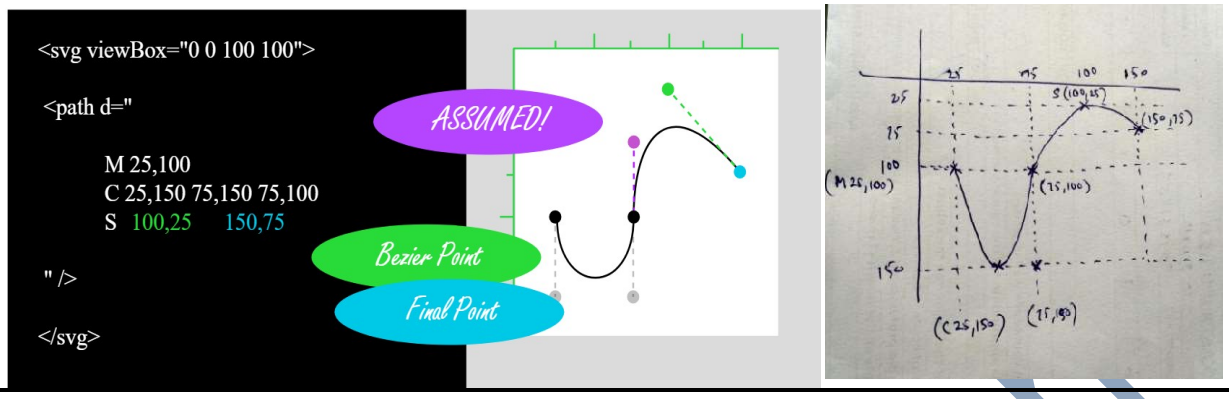
Note: See the very last character Chris used there? Z. Z (or z, it doesn't matter) "closes" the path. Like any other command, it's optional. It's a cheap n' easy way to draw a straight line directly back to the last place the "pen" was set down (probably the last M or m command). It saves you from having to repeat that first location and using a line command to get back there.

Ex 3

```
<svg viewBox="0 0 300 300">
    <path d="  M 25,50 C 25,100  150,100  150,50 " style="stroke:#660000; fill:none;"/>
<path d="  M 25,100 C 25,150 75,150 75,100 S 125,0 150,100  "
      style="stroke:#660000; fill:none;"/>
    </svg>
```

The lowercase `c` command is exactly the same, except all three points use relative values.

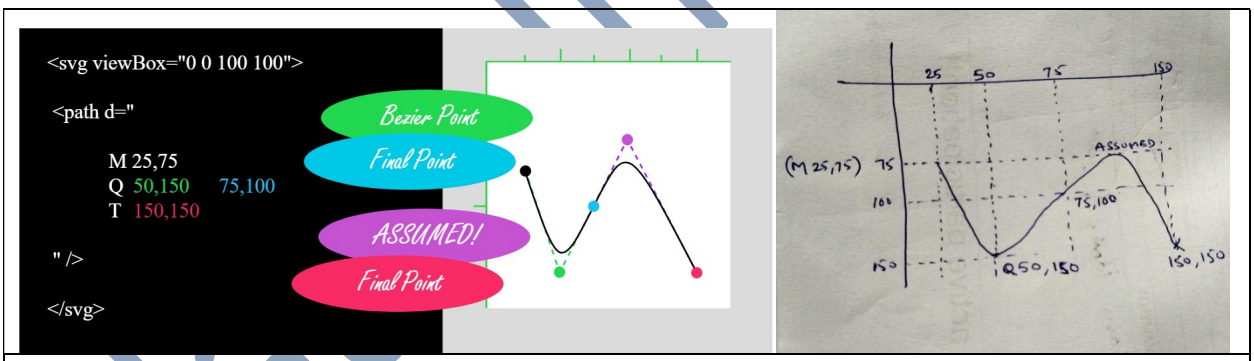
The `S` (or `s`) command is buddies with the C commands in that it only requires *two* points because it assumes that the first bezier point is a reflection of the last bezier point from the last S or C command.



Ex 4

The Q command is one of the easier ones as it only requires two points. The bezier point it wants is a "Quadratic" curve control point. It's as if both the starting and ending point share a single point for where their control handle end.

We might as well cover T at the same time. It's buddies with Q just like S is with C. When T comes after a Q, the control point is assumed to be a reflection of the previous one, so you only need to provide the final point.




```
<svg viewBox="0 0 300 300">
    <path d="  M 25,75 Q 50,150 75,100 T 150,150 " style="stroke:#660000; fill:none;"/>
 </svg>
```

Some Interesting Paths- For practicing

```
<svg viewBox="0 0 10 10" class="svg-1">
    <path d="M2,2 L8,8" />
  </svg>

  <svg viewBox="0 0 10 10" class="svg-2">
    <path d="M2,8 L5,2 L8,8" />
  </svg>
```

```
<svg viewBox="0 0 10 10" class="svg-3">
  <path d="M2,2 Q8,2 8,8" />
</svg>

<svg viewBox="0 0 10 10" class="svg-4">
  <path d="M2,5 C2,8 8,8 8,5" />
</svg>

<svg viewBox="0 0 10 10" class="svg-5">
  <path d="M2,2 L8,2 L2,5 L8,5 L2,8 L8,8" />
</svg>

<svg viewBox="0 0 10 10" class="svg-6">
  <path d="M2,5 A 5 25 0 0 1 8 8" />
</svg>

<svg viewBox="0 0 10 10" class="svg-7">
  <path d="M2,5 S2,-2 4,5 S7,8 8,4" />
</svg>

<svg viewBox="0 0 10 10" class="svg-8">
  <path d="M5,2 Q 2,5 5,8" />
</svg>

<svg viewBox="0 0 10 10" class="svg-9">
  <path d="M2,2 Q5,2 5,5 T8,8" />
</svg>
```

Building an Application with SVG GEOLOCATION:

Comparing Geolocation techniques in the past & modern day Geolocation
Understanding GPS/ IP Address/ Cell IDs/ WiFi and Bluetooth
LBS (Location based services)
Understanding Latitude,Longitude,Speed,Course & Accuracy
Getting you current location
Browser compatibility & Fallbacks.
Reverse geocoding&Mapping location
Getting Distance & Directions between two places.
Following a moving location
Combing geolocation with google maps
Triggering the Privacy Protection Mechanism
Saving Geographical information
Geolocation usage – Geo Marketing,Geo social,Geo tagging,Geo tagging & Geo applications.
Building a Real-Time Application with HTML5 Geolocation
Implementing techniques for Backward compatibility.
Alternative methods when Native Geolocation fails (Geo.js & MaxMind)

Ex 1

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Get Visitor's Location Using HTML5 Geolocation</title>
<script>
```

```
    function showPosition() {
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(function(position) {
                var positionInfo = "Your current position is (" + "Latitude: " + position.coords.latitude + ", "
+ "Longitude: " + position.coords.longitude + ")";
                document.getElementById("result").innerHTML = positionInfo;
            });
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Dealing with Errors and Rejections

There may be a situation when a user does not want to share his location data with you. To deal with such situations, you can supply two functions when you call the getCurrentLocation() function.

The first function is called if your geolocation attempt is successful, while the second is called if your geolocation attempt ends in failure. Let's check out an example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Handling the Geolocation Errors and Rejections</title>
<script>
    // Set up global variable
    var result;

    function showPosition() {
        // Store the element where the page displays the result
        result = document.getElementById("result");

        // If geolocation is available, try to get the visitor's position
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
            result.innerHTML = "Getting the position information...";
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    };

    // Define callback function for successful attempt
    function successCallback(position) {
        result.innerHTML = "Your current position is (" + "Latitude: " + position.coords.latitude + ", " +
"Longitude: " + position.coords.longitude + ")";
    }
```

```
    // Define callback function for failed attempt
    function errorCallback(error) {
        if(error.code == 1) {
            result.innerHTML = "You've decided not to share your position, but it's OK. We won't ask you again.";
        } else if(error.code == 2) {
            result.innerHTML = "The network is down or the positioning service can't be reached.";
        } else if(error.code == 3) {
            result.innerHTML = "The attempt timed out before it could get the location data.";
        } else {
            result.innerHTML = "Geolocation failed due to unknown error.";
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Ex 3 - Showing Location on Google Map

You can do very interesting things with geolocation data, like showing the user location on Google map. The following example will show your current location on Google map based the latitude and longitude data retrieved through the HTML5 geolocation feature.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Showing Geolocation on Google Map Image</title>
<script>
    function showPosition() {
        navigator.geolocation.getCurrentPosition(showMap);
    }

    function showMap(position) {
        // Get location data
        var latlong = position.coords.latitude + "," + position.coords.longitude;

        // Set Google map source url
        var mapLink =
"https://maps.googleapis.com/maps/api/staticmap?center="+latlong+"&zoom=16&size=400x300&o
utput=embed";

        // Create and insert Google map
        document.getElementById("embedMap").innerHTML = "<img alt='Map Holder' src='"+
mapLink +"'>";
    }
```

```
</script>
</head>
<body>
    <button type="button" onclick="showPosition();">Show My Position on Google Map</button>
    <div id="embedMap">
        <!--Google map will be embedded here-->
    </div>
</body>
</html>
```

Ex 4

The above example will simply show the location on the Google map using a static image. However, you can also create interactive Google maps with dragging, zoom in/out and other features that you have come across in your real life. Let's take a look at the following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Showing User's Location on Google Map</title>
<script src="https://maps.google.com/maps/api/js?sensor=false"></script>
<script>
function showPosition() {
    if(navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showMap, showError);
    } else {
        alert("Sorry, your browser does not support HTML5 geolocation.");
    }
}

// Define callback function for successful attempt
function showMap(position) {
    // Get location data
    lat = position.coords.latitude;
    long = position.coords.longitude;
    var latlong = new google.maps.LatLng(lat, long);

    var myOptions = {
        center: latlong,
        zoom: 16,
        mapTypeControl: true,
        navigationControlOptions: {
            style:google.maps.NavigationControlStyle.SMALL
        }
    }

    var map = new google.maps.Map(document.getElementById("embedMap"), myOptions);
    var marker = new google.maps.Marker({ position:latlong, map:map, title:"You are here!" });
}
function showError(error) {
    if(error.code == 1) {
        result.innerHTML = "You've decided not to share your position, but it's OK. We won't ask you again.";
    } else if(error.code == 2) {
        result.innerHTML = "The network is down or the positioning service can't be reached.";
    } else if(error.code == 3) {
        result.innerHTML = "The attempt timed out before it could get the location data.";
    } else {
        result.innerHTML = "Geolocation failed due to unknown error.";
```

```
    }
}
</script>
</head>
<body>
    <button type="button" onclick="showPosition();">Show My Position on Google Map</button>
    <div id="embedMap" style="width: 400px; height: 300px;">
        <!--Google map will be embedded here-->
    </div>
</body>
</html>
```

Ex 5- Monitoring the Visitor's Movement

All the examples we've used so far have relied on the `getCurrentPosition()` method. However, the geolocation object has another method `watchPosition()` that allow you to track the visitor's movement by returning the updated position as the location changes.

The `watchPosition()` has the same input parameters as `getCurrentPosition()`. However, `watchPosition()` may trigger the success function multiple times — when it gets the location for the first time, and again, whenever it detects a new position. Let's see how this works:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Getting Current Position</title>
<script>
    // Set global variable
    var watchID=true;

    function showPosition() {
        if(navigator.geolocation) {
            watchID = navigator.geolocation.watchPosition(successCallback);
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    }
    function successCallback(position) {
        toggleWatchBtn.innerHTML = "Stop Watching";

        // Check position has been changed or not before doing anything
        if(prevLat != position.coords.latitude || prevLong != position.coords.longitude) {

            // Set previous location
            var prevLat = position.coords.latitude;
            var prevLong = position.coords.longitude;
                    // Get current position
            var positionInfo = "Your current position is (" + "Latitude: " + position.coords.latitude + ", " +
"Longitude: " + position.coords.longitude + ")";
            document.getElementById("result").innerHTML = positionInfo;
         }
      }
    function startWatch() {
        var result = document.getElementById("result");

        var toggleWatchBtn = document.getElementById("toggleWatchBtn");

        toggleWatchBtn.onclick = function() {
```

```
            if(watchID) {
                toggleWatchBtn.innerHTML = "Start Watching";
                navigator.geolocation.clearWatch(watchID);
                watchID = false;
            } else {
                toggleWatchBtn.innerHTML = "Aquiring Geo Location...";
                showPosition();
            }
        }
    }

    // Initialise the whole system (above)
    window.onload = startWatch;
</script>
</head>
<body>
    <button type="button" id="toggleWatchBtn">Start Watching</button>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
</body>
</html>
```

Media API (video & audio):

Flash V/s HTML5 video
Adding Video & Audio to a page
Supported Audio & Video formats& Codecs
Lossy & Lossless compression
Media specific attributes Vs Global attributes
Deployment challenges on Mobiles Converting Audio & Video to supported formats using open source & commercial software
Using a Frame grabber
Custom Controls,Seekbar,Progressbar with Javascript & CSS
Applying CSS skins & transforms
Working with multiple tracks,Subtitles & Captions with Captionator,Playr & the Leanback Player
Integrating Video with Canvas & SVG
Applying Visual filters using Canvas & SVG
Debuggin,Browser support &Licensing issues.
Implementing techniques for Backward compatibility.

HTML5 Audio

Embedding Audio in HTML Document

Inserting audio onto a web page was not easy before, because web browsers did not have a uniform standard for defining embedded media files like audio.

In this chapter we'll demonstrates some of the many ways to embed sound in your webpage, from the use of a simple link to the use of the latest HTML5 `<audio>` element.

Using the HTML5 audio Element

The newly introduced HTML5 `<audio>` element provides a standard way to embed audio in web pages. However, the audio element is relatively new but it works in most of the modern web browsers.

The following example simply inserts an audio into the HTML5 document, using the browser default set of controls, with one source defined by the `src` attribute.

```
<html>
    <body>
            <audio id="myAudio" controls>
                    <source src="horse.mp3" type="audio/mpeg">
                    <source src="horse.ogg" type="audio/ogg">
                    Your browser does not support the audio element.
             </audio>
             <audio controls="controls" src="horse.mp3">
                Your browser does not support the HTML5 Audio element.
            </audio>
            <a href="horse.mp3">Track 1</a>
            <embed src="horse.mp3">
    </body>
</html>
```

Attributes in Audio

Attribute	Values	Use
Src	song.mp3	to link Audio file with player ***Compulsory***
Controls		To show **play/pause** buttons, **timeline** and **volume controller** of the player.
Autoplay	"on" or "off".	To play audio automatically.
Loop		To play audio continuously even after it ends.
controlsList	nodownload	To disable download button in chrome 55 onwards

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

<audio id="myAudio">
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
  Your browser does not support the audio element.
</audio>

<p>Click the buttons to play or pause the audio.</p>

<button onclick="playAudio()" type="button">Play Audio</button>
<button onclick="pauseAudio()" type="button">Pause Audio</button>

<script>
var x = document.getElementById("myAudio");

function playAudio() {
  x.play();
}

function pauseAudio() {
  x.pause();
}
</script>

</body>
</html>
```

Video

To embed a video, create a video tag. src is compulsory attribute for video tag. controls attribute can

add play/pause button, video timeline, mute button, volume controller, full screen, subtitles options on player

```
<html>
    <body>
            <video controls id="video1" width="420">
                    <source src="mov_bbb.mp4" type="video/mp4">
                    <source src="mov_bbb.ogg" type="video/ogg">
                    Your browser does not support HTML5 video.
            </video>
            <video src="mov_bbb.mp4" controls></video>
            <embed src="mov_bbb.mp4">
            <a href="mov_bbb.mp4">Track 2</a>
    </body>
</html>
```

Html5 Video Player

Attributes in Video Tag

Attributes	Values	Use
Src	file.mp4	to link Video file with player ***Compulsory***
Controls		To show **play /pause** buttons, **timeline**, **volume controller** and **full screen** buttons on video player.
autoplay	"on" or "off"	To play video on page load.
loop		To play video continuously even after it ends.
width	in px	Defines width of video player.
height	pixel height	Defines height of video player. (in px)
controlsList	nodownload	To disable download button in chrome 55 onwards
poster	image.jpg	shows an image(jpg or png) on pageload. Will not work if autoplay is on
audio	muted	*audio="muted"* attribute will mute audio of video.
track	sub.vtt	track tag is used to add **subtitles** of video. See Example

```
<video width="400" controls  >
    <source src="video.mp4" type="video/mp4" >
       <track kind="captions" src="sub.vtt"
```

```
srclang="en-us" label="English" >

      <track kind="captions" src="sub2.vtt"
srclang="hi" label="हिंदी" >

    </video>
```

Track will work only in *http://* or *https://* protocol.

View Sub.vtt file

Download Sub.vtt file

JavaScript functions for video tag
To add custom functionality in video, we can use javascript properties and methods. Here are some javascript functions with example.

_____
Video custom controls
To play a video file, use play function. To pause video, use paused function.
Play Pause Mute Unmute Full Screen Picture in Picture

```
<script>
  var video=document.querySelector("video");

  video.play();              // play video
  video.pause();             // pause video
  video.paused;              // true or false
  video.volume;              // volume of video
  video.currentTime;         // current time of video
  video.duration;            // video duration (in secs)
  video.requestFullscreen();     // full screen video
  video.webkitRequestFullscreen(); // full screen for safari
  video.requestPictureInPicture(); // Picture in Picture

</script>
```

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

<div style="text-align:center">
  <button onclick="playPause()">Play/Pause</button>
  <button onclick="makeBig()">Big</button>
  <button onclick="makeSmall()">Small</button>
  <button onclick="makeNormal()">Normal</button>
  <br><br>
  <video id="video1" width="420">
```

```
    <source src="mov_bbb.mp4" type="video/mp4">
    <source src="mov_bbb.ogg" type="video/ogg">
    Your browser does not support HTML5 video.
  </video>
</div>

<script>
var myVideo = document.getElementById("video1");

function playPause() {
  if (myVideo.paused)
    myVideo.play();
  else
    myVideo.pause();
}

function makeBig() {
    myVideo.width = 560;
}

function makeSmall() {
    myVideo.width = 320;
}

function makeNormal() {
    myVideo.width = 420;
}
</script>

<p>Video courtesy of <a href="https://www.bigbuckbunny.org/" target="_blank">Big Buck Bunny</a>.</p>

</body>
</html>
```

```
<html>
    <body>
            <video id="video" width="640" height="480" autoplay></video>
            <button id="snap">Snap Photo</button>
            <canvas id="canvas" width="640" height="480"></canvas>
            <script>
                // Grab elements, create settings, etc.
var video = document.getElementById('video');

// Get access to the camera!
if(navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
    // Not adding `{ audio: true }` since we only want video now
    navigator.mediaDevices.getUserMedia({ video: true }).then(function(stream) {
        //video.src = window.URL.createObjectURL(stream);//not working in chrome
        video.srcObject = stream;
        video.play();
    });
}
var canvas = document.getElementById('canvas');
var context = canvas.getContext('2d');
var video = document.getElementById('video');

// Trigger photo take
document.getElementById("snap").addEventListener("click", function() {
```

```
    context.drawImage(video, 0, 0, 640, 480);
});
            </script>
    </body>
</html>
```

WEB WORKERS API:

What are web workers ?
Possibilities & Limitations of web workers
Inline,Dedicated & Shared Workers
Creating a worker,Assign roles & Deploying the same.
Leveragin a Shared Worker
Worker support in modern browsers
Managing multiple workers
Parsing data with workers
Perform Heavy array computations
Using timers in conjunction with worker
Work with pixel manipulations
Make twitter JSONP requests
Connect to share workers at same time with multiple browser windows
Transferable objects
Debuging Your Workers
Implementing techniques for Backward compatibility.
Building an Application with HTML5 Web Workers API

HTML5 Custom Data Attributes (data-*)

Attribute Name
Attribute Value
How can I use data attributes?
What shouldn't I use data attributes for?
Using data- attributes with JavaScript
A word of warning

Web Worker

JavaScript uses the single-threaded model, which means that all tasks can only be done on one thread, and only one thing at a time.The latter tasks can only be waiting if the previous tasks have not been finished. With the increase of computing power, and especially the emergence of multi-core CPUs, single-threading brings great inconvenience and makes it impossible to fully utilize the computing power of computers.

The role of Web Workers is to create a multi-threaded environment for JavaScript, allowing the main thread to create worker threads and assigning some tasks to the latter. While the main thread is running, the Worker thread runs in the background, and they don't interfere with each other. The result will be returned to the main thread after the Worker thread has completed the calculation task. The advantage of this is that some computationally intensive or high-latency tasks are burdened by the Worker thread, and the main thread (usually responsible for UI interaction) will become smooth and won't be blocked or slowed down.

Once the Worker thread is created successfully, it will always run and won't be interrupted by activities on the main thread (such as the user clicking the button and submitting the form). It's good for responding to the communication of the main thread at any time. However, it also causes Workers to be more resource intensive, so it shouldn't be overused, and should be closed once they are used.

There are a few important points when using the Web Worker.

(1) homology restriction

The script file assigned to the Worker thread must have the same origin as the script file of the main thread.

(2) DOM restriction

The Worker thread's global object is different from the main thread's. The former can't read the DOM objects of the web page where the main thread is located, nor can it use the objects of document, window, and parent. However, the Worker thread can use the objects of navigatorand location.

(3) Communication

Worker thread and main thread are not in the same context, so they can't communicate directly and must be done through messages.

(4) Script restriction

Worker thread can't execute the alert() method and the confirm() method, but can use the XMLHttpRequest object to make AJAX requests.

(5) File restriction

Worker thread cannot read the native file, that is, it can't open the native file system (file://) . The script loaded must come from the network.

Webworker.js

```
var i = 0;
function countNumbers() {
    // if(i < 100000) {
    //     i = i + 1;
    //     postMessage(i);
    // }
    // self.addEventListener('message', function (e) {
    //     self.postMessage('You said: ' + e.data);
    //   }, false);
    // Wait for sometime before running this script again
    self.addEventListener('message', function (e) {
        var data = e.data;
        console.log("....:"+data);
        switch (data) {
          case 'start':
            self.postMessage('WORKER STARTED....: ' + data);
            break;
          case 'stop':
            self.postMessage('WORKER STOPPED: ' + data);
            self.close(); // Terminates the worker.
            break;
          default:
            self.postMessage('Unknown command..!!..: ' + data.msg+".."+e.data);
        };
      }, false);
      // Worker thread
    setTimeout("countNumbers()", 500);
}
countNumbers();
```

WebWorerDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Start/Stop Web Worker</title>
<script>
    // Set up global variable
    var worker;
    function startWorker() {
        // Initialize web worker
```

```
        worker = new Worker("webworker11.js");
        // Run update function, when we get a message from worker
        worker.onmessage = update;
        // Tell worker to get started
        worker.postMessage("start");
       // worker.postMessage('Hello World!!.......');
        //worker.postMessage({method: 'echo', args: ['Work']});
//         worker.onerror(function (event) {
//   console.log([
//     'ERROR: Line ', e.lineno, ' in ', e.filename, ': ', e.message
//   ].join(''));
// });
// or
worker.addEventListener('error', function (event) {
  // ...
});
    }
    function update(event) {
        // Update the page with current message from worker
        document.getElementById("result").innerHTML = event.data;
    }
    function stopWorker() {
        // Stop the worker
        console.log("stopping.....");
        worker.postMessage("stop");
        worker.terminate();
    }
</script>
</head>
<body>
    <h1>Web Worker Demo</h1>
    <button onclick="startWorker();" type="button">Start web worker</button>
    <button type="button" onclick="stopWorker();">Stop web worker</button>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
</html>
```

Binary data, such as File, Blob, ArrayBuffer, etc., can also be exchanged between the main thread and the Worker, and it can also be sent between threads. Here is an example.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Start/Stop Web Worker</title>
<script>
    // Set up global variable
    var worker;
    function startWorker() {
        // Initialize web worker
        worker = new Worker("webworker12.js");
        // Run update function, when we get a message from worker
        worker.onmessage = update;
        // main thread
var uInt8Array = new Uint8Array(new ArrayBuffer(10));
for (var i = 0; i < uInt8Array.length; ++i) {
```

```
  uInt8Array[i] = i * 2; // [0, 2, 4, 6, 8,...]
}
worker.postMessage(uInt8Array);

    }
    function update(event) {
        // Update the page with current message from worker
        document.getElementById("result").innerHTML = event.data;
    }
    function stopWorker() {
        // Stop the worker
        console.log("stopping.....");
        worker.postMessage("stop");
        worker.terminate();
    }
</script>
</head>
<body>
    <h1>Web Worker Demo</h1>
    <button onclick="startWorker();" type="button">Start web worker</button>
    <button type="button" onclick="stopWorker();">Stop web worker</button>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
</html>
```

```
var i = 0;
function countNumbers() {

  self.onmessage = function (e) {
    var uInt8Array = e.data;
    postMessage('Inside worker.js: uInt8Array.toString() = ' + uInt8Array.toString());
    postMessage('Inside worker.js: uInt8Array.byteLength = ' + uInt8Array.byteLength);
  };  // Worker thread
    setTimeout("countNumbers()", 500);
}
countNumbers();
```

However, copying binary data may cause performance problems. For example, the main thread sends a 500MB file to the Worker. The browser will generate a copy of the original file by default. To solve this problem, JavaScript allows the main thread to transfer binary data to the child thread directly. But once it has been transferred, in order to prevent multiple threads modifying the data at the same time, the main thread can no longer use the binary data. This method of transferring data is called Transferable Objects. It makes the main thread hand over data to the Worker quickly, which is very convenient for image processing, sound processing, 3D operations, etc., without a performance burden.

If you want to transfer control of the data directly, you should use the following method.

// Transferable Objects

```
worker.postMessage(arrayBuffer, [arrayBuffer]);

// example

var ab = new ArrayBuffer(1);

worker.postMessage(ab, [ab]);
```

Web Worker in The Same Page

Generally, Worker loads a separate JavaScript script file, but it can also load code that is on the same page as the main thread.

```
<!DOCTYPE html>
  <body>
    <script id="worker" type="app/worker">
      addEventListener('message', function (e) {
        postMessage('some message'+e.data);
      }, false);
    </script>
    <script>
      var blob = new Blob([document.querySelector('#worker').textContent]);
var url = window.URL.createObjectURL(blob);
var worker = new Worker(url);
worker.postMessage("start");
worker.onmessage = function (e) {
  console.log(e.data);
};
    </script>
  </body>
</html>
```

CROSS DOCUMENT MESSAGING API:

Understanding Origin Security
Browser Support for Cross Document Messaging
Building an Application Using the postMessage API
XMLHttpRequest Level
Cross-Origin XMLHttpRequest
Progress Events
Browser Support for HTML XMLHttpRequest Level
Building an Application Using XMLHttpRequest Structured Data&Framebusting
Implementing techniques for Backward compatibility.
Building an Application with HTML5 CDM

Messaging

Asynchronous Java script and XML

With Ajax web applications finally start feeling like desktop applications to users.

AJAX enables your web application to work behind the scenes, getting data as you need and displaying that data as you want.

When this happens, the browser does not get refreshed.

Your work does not get distrubed, because all the request are done behine the screne by AJAX that too in a Asynchronous Fashion.

XMLHttpRequest Object

xhr.open("method","url",asyncflag,"username","password");

xhr.readyState

0-uninitialized
1-loading
2-loaded
3-interactive
4-complete

```
<books><book id="a1"><name>javascript</name><author>brandan
eich</author><prize>100</prize></book><book
id="a2"><name>java</name><author>james</author><prize>500</prize></book><book
id="a3"><name>angular</name><author>misko havery</author><prize>300</prize></book><book
id="a4"><name>jquery</name><author>john resig</author><prize>300</prize></book></books>


<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<script type="text/javascript">
    var XMLHttpRequestObject = false;
    if(window.XMLHttpRequest)
    {
        XMLHttpRequestObject =new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
```

```
        XMLHttpRequestObject =
        new ActiveXObject("Microsoft.XMLHTTP");
    }
    var domobj;
    function getData(dataSource,divID)
    {
        if(XMLHttpRequestObject)
        {
        var obj=document.getElementById(divID);
            XMLHttpRequestObject.open("GET",dataSource);
            XMLHttpRequestObject.onreadystatechange=function()
                    {
                        if(XMLHttpRequestObject.readyState==4 &&
                        XMLHttpRequestObject.status ==200)
                        {
                           var response=
                           XMLHttpRequestObject.responseXML;
                           //var root=response.documentElement;
        //   console.log(response.documentElement.firstChild.firstChild.nodeValue);
            console.log(response);//alert(response);
                domobj= response;
             var ne=domobj.createElement("book");
             var te=domobj.createTextNode("blablablabla");
             var att=domobj.createAttribute("bid");
            var attvalue=domobj.createTextNode("BBBBBB");

            // att.appendChild(attvalue);
            // ne.attributes.setNamedItem(att);
             ne.appendChild(te);
             var root=domobj.documentElement;
             root.appendChild(ne);
            obj.innerHTML=response;
}
                    }
            XMLHttpRequestObject.send(null);
        }
    }

</script>
<form>
            <input type="button" value="Display message"
       onclick=
      "getData('books.xml','targetDiv')">

        <div id="targetDiv" style="color:red">

            <p>The fetched data will go here</p>
        </div>
        </form>
</body>
</html>
```

Dropdown demo

```
<script type="text/javascript">
    var XMLHttpRequestObject = false;
```

```
    if(window.XMLHttpRequest)
    {
        XMLHttpRequestObject =new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        XMLHttpRequestObject =
        new ActiveXObject("Microsoft.XMLHTTP");
    }
    function removeWhiteSpace(xml)
    {
        var loopIndex;
        for(lookIndex=0;loopIndex<xml.childNodes.length;loopIndex++)
        {
            var currentNode = xml.childNodes[loopIndex];
            if(currentNode.nodeType==1){
                removeWhiteSpace(currentNode);
            }
            if ((((/^\s+$/.test(currentNode.nodeValue))) &&
                (currentNode.nodeType==3))
                {
                    xml.removeChild(xml.childNodes[loopIndex--]);
                }
        }
    }
    function setCity(dataSource)
    {
        if(XMLHttpRequestObject)
        {
            XMLHttpRequestObject.open("GET",dataSource);
            XMLHttpRequestObject.onreadystatechange=function()
                    {
                        if(XMLHttpRequestObject.readyState==4 &&
                        XMLHttpRequestObject.status ==200)
                        {
                           var response=
                           XMLHttpRequestObject.responseXML;
                           removeWhiteSpace(response);
        var countrylist=response.getElementsByTagName("country");
  var countryselected=document.getElementById("countryList").value;
                           for(var i=0;i<countrylist.length;i++)
                                {
                                var c=countrylist[i].attributes.
                                getNamedItem("name").nodeValue;
                                if(c==countryselected){
                                    push(countrylist[i]);   }
                                }
                        }
                    }
            XMLHttpRequestObject.send(null);
        }
    }
   function push(countrylist)
   {
   var target=document.getElementById("target");
   for(j=0;j<countrylist.childNodes.length;j++){
```

```
    target.options[j]=new Option(countrylist.childNodes[j].
    firstChild.nodeValue);
    }
   }
</script>

        <body>

        <form>
        <select size="1" id="countryList" onchange="setCity('city.xml')">
            <option value="india">India</option>
            <option value="iraq">IRAQ</option>
        </select>
         <select size="1" id="target">
            <option value="india">cities</option>

        </select>
        </form>
        </body>

<cities>
<country name="india">
    <city>chennai</city>
    <city>mumbai</city>
</country>
<country name="iraq">
    <city>baghdad</city>
    <city>basra</city>
</country>
</cities>
```

Reading JSON

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<script type="text/javascript">
    var XMLHttpRequestObject = false;
    if(window.XMLHttpRequest)
    {
        XMLHttpRequestObject =new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        XMLHttpRequestObject =
        new ActiveXObject("Microsoft.XMLHTTP");
    }
    var domobj;
    function getData(dataSource,divID)
    {
        if(XMLHttpRequestObject)
        {
        var obj=document.getElementById(divID);
```

```
            XMLHttpRequestObject.open("GET",dataSource);
            XMLHttpRequestObject.onreadystatechange=function()
                    {
                        if(XMLHttpRequestObject.readyState==4 &&
                        XMLHttpRequestObject.status ==200)
                        {
                           var response=
                           XMLHttpRequestObject.responseText;

              // obj.innerHTML=response;
               var obj = JSON.parse(response);

    // Accessing individual value from JS object
    document.write(obj.name + "<br>"); // Prints: Peter
    document.write(obj.age + "<br>"); // Prints: 22
    document.write(obj.country); // Prints: United States
}
                     }
            XMLHttpRequestObject.send(null);
        }
    }

</script>
<form>
            <input type="button" value="Display message"
       onclick=
      "getData('name.json','targetDiv')">

        <div id="targetDiv" style="color:red">

            <p>The fetched data will go here</p>
        </div>
        </form>
</body>
</html>
```

Name.json

```
{"name": "haaris", "age": 22, "country": "India"}
```

Server Sent Events (SSE)

Possible Applications
Overview of the API
new EventSource(url)
Properties of Server-Sent Events
Message Format
Typical Server
Polyfills and Tweaks to the Server
Why Not Use WebSockets

1. What's SSE?

Strictly speaking, the HTTP protocol doesn't allow the server to push information actively. However, there is a

workaround: The server declares to the client what will be sent next is streaming.

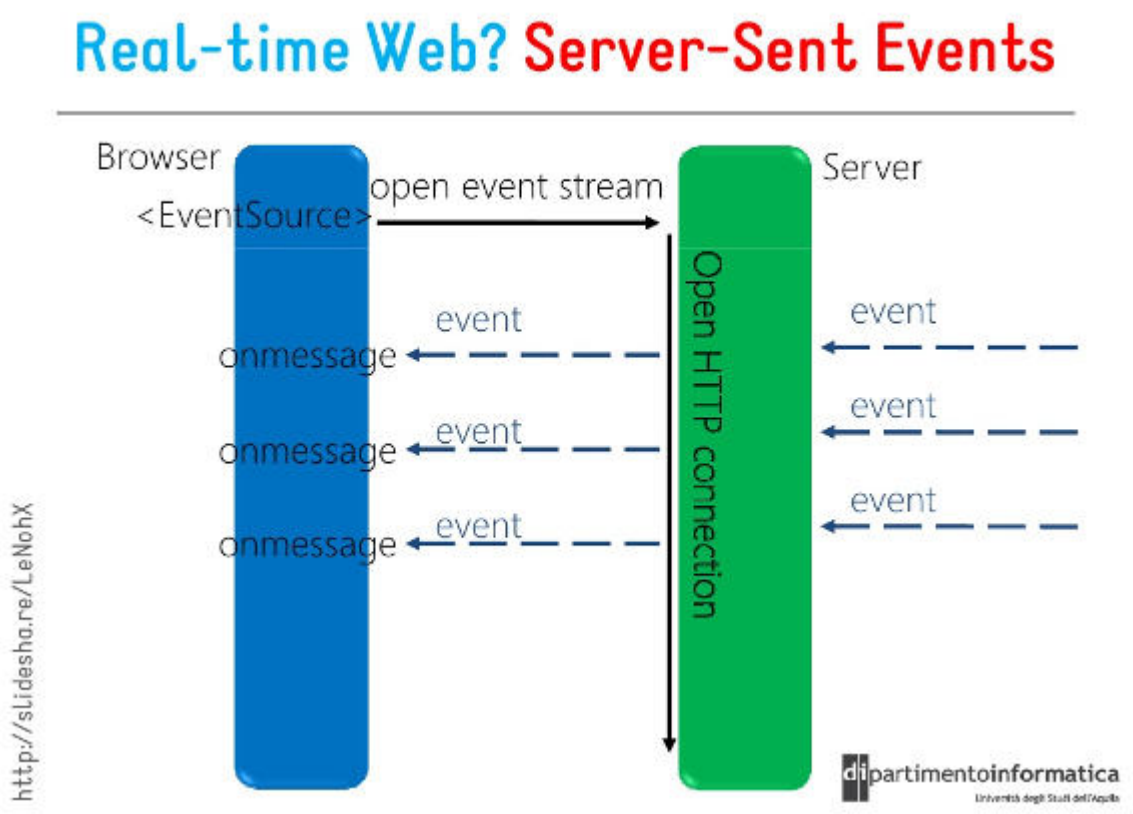
In other words, what will be sent continuously is a stream of data instead of a one-off packet. At this time, the client won't close the connection, but will wait for the new data stream sent by the server. Video playback is an example of this. Essentially, this kind of communication is to achieve a long-time download by the way of streaming.

And SSE uses this mechanism to make streaming push information to the browser. It is based on the HTTP protocol and is supported by other browsers currently except for IE/Edge.

The Features of SSE

SSE is similar to WebSocket for that they both establish a communication channel between the browser and the server, and then the server push information to the browser.

Overall, WebSockets is more powerful and flexible. WebSockets is a full-duplex channel, so it can communicate in both directions; SSE is a one-way channel and can only be sent by the server to the browser, because the streaming is a download essentially. And if the browser sends information to the server, it then becomes another HTTP request.



However, SSE has its own advantages.

- SSE uses the HTTP protocol and is supported by all the existing servers ; and WebSocket is a standalone protocol.
- SSE is lightweight and easy to use; and the WebSocket protocol is relatively complex.
- SSE supports reconnection by default; and WebSocket needs to be implemented by itself.
- SSE is generally only used to transfer text, and binary data needs to be encoded first and then transmitted; and WebSocket supports binary data transfer by default.
- SSE supports the message types for being customized and then sent.

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

<h1>Getting server updates</h1>
<div id="result"></div>

<script>
if(typeof(EventSource) !== "undefined") {
    var source = new EventSource("http://localhost:2020/html5/dispatcher");
    source.onmessage = function(event) {
        document.getElementById("result").innerHTML += event.data + "<br>";
    };
} else {
    document.getElementById("result").innerHTML = "Sorry, your browser does not support server-sent
events...";
}
</script>

</body>
</html>
```

Servlet

```
package ssepack;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.ResultSet;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SSEServlet extends HttpServlet {
	@Override
	public void init() throws ServletException {
		System.out.println("servlet loaded...");
	}
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
			response.setContentType("text/event-stream");
			response.setCharacterEncoding("UTF-8");
			PrintWriter writer = response.getWriter();
			writer.write("Now the names will get printed...."+"\n\n");
			writer.write("data: "+ "hello:"+System.currentTimeMillis() +"\n\n");
			writer.flush();
	}
}
```

Ex 2:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
<div id="example"></div>
<script>
  var source = new EventSource('http://127.0.0.1:8844/stream');
  var div = document.getElementById('example');

  source.onopen = function (event) {
    div.innerHTML += '<p>Connection open ...</p>';
  };

  source.onerror = function (event) {
    div.innerHTML += '<p>Connection close.</p>';
  };

  source.addEventListener('connecttime', function (event) {
    div.innerHTML += ('<p>Start time: ' + event.data + '</p>');
  }, false);

  source.onmessage = function (event) {
    div.innerHTML += ('<p>Ping: ' + event.data + '</p>');
  };

</script>
```

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

Server.js

```
var http = require("http");

http.createServer(function (req, res) {
  var fileName = "." + req.url;

  if (fileName === "./stream") {
    res.writeHead(200, {
      "Content-Type":"text/event-stream",
      "Cache-Control":"no-cache",
      "Connection":"keep-alive",
      "Access-Control-Allow-Origin": '*',
    });
    res.write("retry: 10000\n");
    res.write("event: connecttime\n");
    res.write("data: " + (new Date()) + "\n\n");
    res.write("data: " + (new Date()) + "\n\n");

    interval = setInterval(function () {
      res.write("data: " + (new Date()) + "\n\n");
    }, 1000);

    req.connection.addListener("close", function () {
      clearInterval(interval);
    }, false);
  }
}).listen(8844, "127.0.0.1");
```

To start this server – issue the following command

>node server.js

WEB SOCKET API:

Understanding WebSocket
WebSocket API
WebSocket Protocol
Writing a Simple Echo WebSocket Server
Using the WebsocketAPI
Checking for Browser Support
Building a WebSocket Application
Adding the Geolocation Code
Combining Geolocation & Web sockets together.
Building Instant Messaging and Chat over WebSocket with XMPP
Using Messaging over WebSocket with STOMP
VNC with the Remote Framebuffer Protocol
WebSocket Security
Deployment Considerations
Inspecting WebSocket Traffic
Implementing techniques for Backward compatibility.

Web sockets are defined as a two-way communication between the servers and the clients, which mean both the parties, communicate and exchange data at the same time. This protocol defines a full duplex communication from the ground up. Web sockets take a step forward in bringing desktop rich functionalities to the web browsers. It represents an evolution, which was awaited for a long time in client/server web technology.

The key points of Web Sockets are **true concurrency** and **optimization of performance**, resulting in more responsive and rich web applications.

However, none of the early solutions offered a truly standardized cross browser solution to real-time bi-directional communication between a server and a client.

This gave rise to the need of Web Sockets Protocol. It gave rise to full-duplex communication bringing desktop-rich functionality to all web browsers.

Web Socket is an independent TCP-based protocol, but it is designed to support any other protocol that would traditionally run only on top of a pure TCP connection.

- **Reduce unnecessary network traffic and latency** using full duplex through a single connection (instead of two).
- **Streaming through proxies and firewalls**, with the support of upstream and downstream communication simultaneously.

The steps for establishing the connection of Web Socket are as follows −

- The client establishes a connection through a process known as Web Socket handshake.
- The process begins with the client sending a regular HTTP request to the server.
- An Upgrade header is requested. In this request, it informs the server that request is for Web Socket connection.
- Web Socket URLs use the **ws** scheme. They are also used for secure Web Socket connections, which are the equivalent to HTTPs.
-

Ex1:

```
<!DOCTYPE html>
<html>
   <meta charset = "utf-8" />
   <title>WebSocket Test</title>

   <script language = "javascript" type = "text/javascript">
      var wsUri = "ws://echo.websocket.org/";
      var output;

      function init() {
         output = document.getElementById("output");
         testWebSocket();
      }

      function testWebSocket() {
         websocket = new WebSocket(wsUri);

         websocket.onopen = function(evt) {
            onOpen(evt)
         };
      }
```

```
      function onOpen(evt) {
         console.log("connected....");
      }

      window.addEventListener("load", init, false);

   </script>

   <h2>WebSocket Test</h2>
   <div id = "output"></div>

</html>
```

Ex 2:

```
<!DOCTYPE html>
<html>
   <meta charset = "utf-8" />
   <title>WebSocket Test</title>

   <script language = "javascript" type = "text/javascript">
      var wsUri = "ws://echo.websocket.org/";
      var output;

      function init() {
         output = document.getElementById("output");
         testWebSocket();
      }

      function testWebSocket() {
         websocket = new WebSocket(wsUri);

         websocket.onopen = function(evt) {
            onOpen(evt)
         };

         websocket.onclose = function(evt) {
            onClose(evt)
         };

         websocket.onerror = function(evt) {
            onError(evt)
         };
         websocket.onmessage = function(evt) {
            onMessage(evt)
         };
      }

      function onOpen(evt) {
         writeToScreen("CONNECTED");
         doSend("WebSocket rocks");
      }
```

```
      function onClose(evt) {
         writeToScreen("DISCONNECTED");
      }

      function onError(evt) {
         writeToScreen('<span style = "color: red;">ERROR:</span> ' + evt.data);
      }

      function doSend(message) {
         writeToScreen("SENT: " + message);
         websocket.send(message);
      }
      function onMessage(evt) {
         writeToScreen('<span style = "color: blue;">RESPONSE: ' +
            evt.data+'</span>');
            websocket.close();
      }
      function writeToScreen(message) {
         var pre = document.createElement("p");
         pre.style.wordWrap = "break-word";
         pre.innerHTML = message; output.appendChild(pre);
      }

      window.addEventListener("load", init, false);
   </script>

   <h2>WebSocket Test</h2>
   <div id = "output"></div>

</html>
```

Chat Application Example

ChatServer.js

```
// http://ejohn.org/blog/ecmascript-5-strict-mode-json-and-more/
"use strict";

// Optional. You will see this name in eg. 'ps' or 'top' command
process.title = 'node-chat';

// Port where we'll run the websocket server
var webSocketsServerPort = 1337;

// websocket and http servers
var webSocketServer = require('websocket').server;
var http = require('http');

/**
 * Global variables
 */
// latest 100 messages
var history = [ ];
// list of currently connected clients (users)
var clients = [ ];
```

```
/**
 * Helper function for escaping input strings
 */
function htmlEntities(str) {
    return String(str).replace(/&/g, '&amp;').replace(/</g, '&lt;')
                      .replace(/>/g, '&gt;').replace(/"/g, '&quot;');
}

// Array with some colors
var colors = [ 'red', 'green', 'blue', 'magenta', 'purple', 'plum', 'orange' ];
// ... in random order
colors.sort(function(a,b) { return Math.random() > 0.5; } );

/**
 * HTTP server
 */
var server = http.createServer(function(request, response) {
    // Not important for us. We're writing WebSocket server, not HTTP server
});
server.listen(webSocketsServerPort, function() {
    console.log((new Date()) + " Server is listening on port " + webSocketsServerPort);
});

/**
 * WebSocket server
 */
var wsServer = new webSocketServer({
    // WebSocket server is tied to a HTTP server. WebSocket request is just
    // an enhanced HTTP request. For more info http://tools.ietf.org/html/rfc6455#page-6
    httpServer: server
});

// This callback function is called every time someone
// tries to connect to the WebSocket server
wsServer.on('request', function(request) {
    console.log((new Date()) + ' Connection from origin ' + request.origin + '.');

    // accept connection - you should check 'request.origin' to make sure that
    // client is connecting from your website
    // (http://en.wikipedia.org/wiki/Same_origin_policy)
    var connection = request.accept(null, request.origin);
    // we need to know client index to remove them on 'close' event
    var index = clients.push(connection) - 1;
    var userName = false;
    var userColor = false;

    console.log((new Date()) + ' Connection accepted.');

    // send back chat history
    if (history.length > 0) {
        connection.sendUTF(JSON.stringify( { type: 'history', data: history} ));
    }

    // user sent some message
    connection.on('message', function(message) {
```

```
        if (message.type === 'utf8') { // accept only text
            if (userName === false) { // first message sent by user is their name
                // remember user name
                userName = htmlEntities(message.utf8Data);
                // get random color and send it back to the user
                userColor = colors.shift();
                connection.sendUTF(JSON.stringify({ type:'color', data: userColor }));
                console.log((new Date()) + ' User is known as: ' + userName
                            + ' with ' + userColor + ' color.');

            } else { // log and broadcast the message
                console.log((new Date()) + ' Received Message from '
                            + userName + ': ' + message.utf8Data);

                // we want to keep history of all sent messages
                var obj = {
                    time: (new Date()).getTime(),
                    text: htmlEntities(message.utf8Data),
                    author: userName,
                    color: userColor
                };
                history.push(obj);
                history = history.slice(-100);

                // broadcast message to all connected clients
                var json = JSON.stringify({ type:'message', data: obj });
                for (var i=0; i < clients.length; i++) {
                    clients[i].sendUTF(json);
                }
            }
        }
    });

    // user disconnected
    connection.on('close', function(connection) {
        if (userName !== false && userColor !== false) {
            console.log((new Date()) + " Peer "
                + connection.remoteAddress + " disconnected.");
            // remove user from the list of connected clients
            clients.splice(index, 1);
            // push back user's color to be reused by another user
            colors.push(userColor);
        }
    });

});
```

>npm install websocket
>node chatserver.js

Client

Websocketchatfrontend.js

```
$(function () {
```

```
"use strict";

// for better performance - to avoid searching in DOM
var content = $('#content');
var input = $('#input');
var status = $('#status');

// my color assigned by the server
var myColor = false;
// my name sent to the server
var myName = false;

// if user is running mozilla then use it's built-in WebSocket
window.WebSocket = window.WebSocket || window.MozWebSocket;

// if browser doesn't support WebSocket, just show some notification and exit
if (!window.WebSocket) {
    content.html($('<p>', { text: 'Sorry, but your browser doesn\'t '
                                + 'support WebSockets.'} ));
    input.hide();
    $('span').hide();
    return;
}

// open connection
var connection = new WebSocket('ws://127.0.0.1:1337');

connection.onopen = function () {
    // first we want users to enter their names
    input.removeAttr('disabled');
    status.text('Choose name:');
};

connection.onerror = function (error) {
    // just in there were some problems with conenction...
    content.html($('<p>', { text: 'Sorry, but there\'s some problem with your '
                                + 'connection or the server is down.' } ));
};

// most important part - incoming messages
connection.onmessage = function (message) {
    // try to parse JSON message. Because we know that the server always returns
    // JSON this should work without any problem but we should make sure that
    // the massage is not chunked or otherwise damaged.
    try {
        var json = JSON.parse(message.data);
    } catch (e) {
        console.log('This doesn\'t look like a valid JSON: ', message.data);
        return;
    }

    // NOTE: if you're not sure about the JSON structure
    // check the server source code above
    if (json.type === 'color') { // first response from the server with user's color
        myColor = json.data;
        status.text(myName + ': ').css('color', myColor);
```

```
            input.removeAttr('disabled').focus();
            // from now user can start sending messages
        } else if (json.type === 'history') { // entire message history
            // insert every single message to the chat window
            for (var i=0; i < json.data.length; i++) {
                addMessage(json.data[i].author, json.data[i].text,
                           json.data[i].color, new Date(json.data[i].time));
            }
        } else if (json.type === 'message') { // it's a single message
            input.removeAttr('disabled'); // let the user write another message
            addMessage(json.data.author, json.data.text,
                       json.data.color, new Date(json.data.time));
        } else {
            console.log('Hmm..., I\'ve never seen JSON like this: ', json);
        }
    };

    /**
     * Send mesage when user presses Enter key
     */
    input.keydown(function(e) {
        if (e.keyCode === 13) {
            var msg = $(this).val();
            if (!msg) {
                return;
            }
            // send the message as an ordinary text
            connection.send(msg);
            $(this).val('');
            // disable the input field to make the user wait until server
            // sends back response
            input.attr('disabled', 'disabled');

            // we know that the first message sent from a user their name
            if (myName === false) {
                myName = msg;
            }
        }
    });

    /**
     * This method is optional. If the server wasn't able to respond to the
     * in 3 seconds then show some error message to notify the user that
     * something is wrong.
     */
    setInterval(function() {
        if (connection.readyState !== 1) {
            status.text('Error');
            input.attr('disabled', 'disabled').val('Unable to comminucate '
                                                   + 'with the WebSocket server.');
        }
    }, 3000);

    /**
     * Add message to the chat window
     */
```

```
    function addMessage(author, message, color, dt) {
        content.prepend('<p><span style="color:' + color + '">' + author + '</span> @ ' +
             + (dt.getHours() < 10 ? '0' + dt.getHours() : dt.getHours()) + ':'
             + (dt.getMinutes() < 10 ? '0' + dt.getMinutes() : dt.getMinutes())
             + ': ' + message + '</p>');
    }
});
```

Frontend.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>WebSockets - Simple chat</title>

        <style>
        * { font-family:tahoma; font-size:12px; padding:0px; margin:0px; }
        p { line-height:18px; }
        div { width:500px; margin-left:auto; margin-right:auto;}
        #content { padding:5px; background:#ddd; border-radius:5px; overflow-y: scroll;
                   border:1px solid #CCC; margin-top:10px; height: 160px; }
        #input { border-radius:2px; border:1px solid #ccc;
                 margin-top:10px; padding:5px; width:400px;  }
        #status { width:88px; display:block; float:left; margin-top:15px; }
        </style>
    </head>
    <body>
        <div id="content"></div>
        <div>
            <span id="status">Connecting...</span>
            <input type="text" id="input" disabled="disabled" />
        </div>

        <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <script src="./websocketchatfrontend.js"></script>
    </body>
</html>
```

Web RTC API:

Introduction to Web Real-time Communication (WebRTC)
Introduction
History of Real-time Communication on the web
What you can do with WebRTC
Where WebRTC is supported

Architecture of a WebRTC Application

Introduction
Security with WebRTC
The Full WebRTC Environment

Understanding Server Technologies for WebRTC

Why Would We Need Servers?
Introducing ICE, STUN and TURN
Signaling Options for WebRTC
Options for Server Setup and Hosting
Recap of the Module

Introducing the WebRTC API

Overview of the WebRTC APIs
Accessing Webcam and Microphone with MediaStream (getUserMedia)
Establishing a Peer Connection with RTCPeerConnection Understanding Data Communication with RTCDataChannel
Recap of the Module

Setting Up Your Development Environment

Introduction to Setting Up Your Environment
Software and tools for WebRTC development
Recap of the module

Create a Two-person Video Chat Using Peer.js

Introduction to Peer.js Library
Set Up the HTML for Two-person Video Chat
Write JavaScript Calls to Peer.js
Test the Two-person Chat Application
Recap of the Module

Create a Multi-person Chat Application Using SimpleWebRTC

Introduction to SimpleWebRTC Framework
Set up the HTML for Multi-person Video Chat
Write JavaScript Calls to SimpleWebRTC
Test the Multi-person Chat Application

With Web Real-Time Communication (WebRTC), modern web applications can easily stream audio and video content to millions of people.

WebRTC allows you to set up peer-to-peer connections to other web browsers quickly and easily. To build such an application from scratch, you would need a wealth of frameworks and libraries dealing with typical issues like data loss, connection dropping, and NAT traversal. With WebRTC, all of this comes built-in into the browser out-of-the-box. This technology doesn't need any plugins or third-party software. It is open-sourced and its source code is freely available at http://www.webrtc.org/.

The WebRTC API includes media capture, encoding and decoding audio and video, transportation layer, and session management.

Use Cases

The real-time web opens the door to a whole new range of applications, including text-based chat, screen and file sharing, gaming, video chat, and more. Besides

communication you can use WebRTC for other purposes like −

- real-time marketing
- real-time advertising
- back office communications (CRM, ERP, SCM, FFM)
- HR management
- social networking
- dating services
- online medical consultations
- financial services
- surveillance
- multiplayer games
- live broadcasting
- e-learning

File IO

File System Directories - Create, List, Delete, Move & Copy :

Introduction
Demo Create and Read Directory
Demo Create Sub Directories
Demo List Directory Contents
Demo Delete and Recursive Delete
Demo Move, Copy and Rename
Summary

File System Building an Abstraction Layer over Directories :

Introduction
Demo localFileSystem Module - Error Handling
Demo localFileSystem Module - Request File System
Demo localFileSystem Module - Create Directory
Demo localFileSystem Module - Directory Exists
Demo localFileSystem Module - Get Directory Entries
Demo localFileSystem Module - Delete
Demo localFileSystem Module - Move, Rename and Copy
Summary

File System Files - Create, Read, Write, Delete, Move & Copy :

Introduction
Demo Create and Get File
Demo Read, Write and Update File
Demo Delete, Move, Rename and Copy File
Demo File Abstractions Overview
Demo localFileSystem Module - Create File
Demo localFileSystem Module - Get and File Exists
Demo localFileSystem Module - Read, Prepend and Append

Demo localFileSystem Module - Delete and Replace File
Demo localFileSystem Module - Move, Rename and Copy File
Summary

File System Testing Capacity Limits & Implementing a File Editor:

Introduction
Demo File System Capacity Limits
Demo File Editor Demonstration
Demo File Editor Markup
Demo File Editor View Model
Summary

The FileSystem-API allows the creation of files and folders as well as their local storage using JavaScript. Files can be simple text files, but even more complex files such as images are possible. Modern Webkit browsers with HTML5 support are already able to handle the FileSystem-API. We show you how you can benefit from the new possibilities.

Ex:

```
<html>
    <body>
        <script>
window.requestFileSystem  = window.requestFileSystem || window.webkitRequestFileSystem;
window.requestFileSystem(window.TEMPORARY, 5*1024*1024, initFS, errorHandler);
//window.PERSISTENT
function initFS(fs){
  alert("Welcome to Filesystem! It's showtime :)"); // Just to check if everything is OK :)
  // place the functions you will learn bellow here
  console.log(fs.root);
  fs.root.getDirectory(
       'documents',
       {create: true},
       function(dirEntry)
        {
            alert('You have just created the ' + dirEntry.name + ' directory.');
        },
       errorHandler
    );
    createDir(fs.root, 'documents/Images/Nature/Sky/'.split('/'));
    createDir(fs.root, 'documents/Music/'.split('/'));
    browse(fs);
    removeDir(fs);
    removeAll(fs);
    workWithFiles(fs);
}
function createDir(rootDir, folders) {
  rootDir.getDirectory(
      folders[0],
      {create: true},
      function(dirEntry) {
        if (folders.length) {
        createDir(dirEntry, folders.slice(1));
        }
    }
```

```
    , errorHandler);
  };

  function browse(fs){
    fs.root.getDirectory('documents', {}, function(dirEntry){
  var dirReader = dirEntry.createReader();
  dirReader.readEntries(function(entries) {
    for(var i = 0; i < entries.length; i++) {
      var entry = entries[i];
      if (entry.isDirectory){
        console.log('Directory: ' + entry.fullPath);
      }
      else if (entry.isFile){
        console.log('File: ' + entry.fullPath);
      }
    }

  }, errorHandler);
}, errorHandler);
};
function removeDir(fs){
    fs.root.getDirectory('documents/Music', {}, function(dirEntry) {
  dirEntry.remove(function(){
    console.log('Directory successfully removed.');
  }, errorHandler);
}, errorHandler);
};
function removeAll(fs){
    fs.root.getDirectory('documents/Music', {}, function(dirEntry) {
  dirEntry.removeRecursively(function(){
    console.log('Directory successufully removed.');
  }, errorHandler);
}, errorHandler);
};
function workWithFiles(fs){
    fs.root.getFile('test.txt', {create: true, exclusive: true}, function(fileEntry) {
  alert('A file ' + fileEntry.name + ' was created successfully.');
}, errorHandler);
/*
Having an empty file is not very useful, though; so let's add some
content inside. We can use the FileWriter object for this.
*/
fs.root.getFile('test.txt', {create: false}, function(fileEntry) {
  fileEntry.createWriter(function(fileWriter) {
    //window.BlobBuilder = window.BlobBuilder || window.WebKitBlobBuilder;
    //var bb = new BlobBuilder();
    //bb.append('Filesystem API is awesome!');
    //fileWriter.write(bb.getBlob('text/plain'));
    var blob = new Blob(["Lorem Ipsum"], {type: "text/plain"});
      fileWriter.write(blob);
  }, errorHandler);
}, errorHandler);

fs.root.getFile('test.txt', {}, function(fileEntry) {
  fileEntry.file(function(file) {
    var reader = new FileReader();
```

```
    reader.onloadend = function(e) {
      alert(this.result);
    };
    reader.readAsText(file);
  }, errorHandler);
}, errorHandler);
}
function errorHandler(err){
 var msg = 'An error occured: ';

  switch (err.code) {
    case FileError.NOT_FOUND_ERR:
      msg += 'File or directory not found';
      break;

    case FileError.NOT_READABLE_ERR:
      msg += 'File or directory not readable';
      break;

    case FileError.PATH_EXISTS_ERR:
      msg += 'File or directory already exists';
      break;

    case FileError.TYPE_MISMATCH_ERR:
      msg += 'Invalid filetype';
      break;

    default:
      msg += 'Unknown Error';
      break;
  };

 console.log(msg);
};
</script>
</body>
</html>
```

Drag and Drop

The Drag and Drop Event is the most fabulous properties of HTML5. With the help of Drag and Drop features We can move an object from one place to another place. The all major browsers support the dragging except of Safari 5.1.2. In the process of dragging an object we have to just grab an object from its original place and leave it or drag it in the specified area. All the process is done with the help of Java Script. The Drag and Drop Event may be a little bit complex but we have tried to make it very simple just take a look at below of the page for detail descriptions.

Ex 1:

```
<html>
    <head>
            <style>
                    #div1 {
```

```
                        width:550px;height:320px;padding:10px;
                        border:1px solid #aaaaaa;}
            </style>
    </head>

<body>

<script>
    function dragstart(e){
        e.dataTransfer.setData("mydraggedobj", e.target.id);
        console.log("drag started...."+e);
    }
    function dragend(e){
        console.log("drag ended..:"+e)
    }
    function drop(e){
        e.preventDefault();
        console.log("drop.....");
        var obj = e.dataTransfer.getData("mydraggedobj");
        e.target.appendChild(document.getElementById(obj));
    }
    function dragover(e){
        e.preventDefault();
        console.log("drag......");
    }
</script>
<div id="div1" ondrop="drop(event)" ondragover="dragover(event)">

</div>
<img src='https://en.js.cx/clipart/soccer-gate.svg'
    height="300" width="300"
    draggable="true"
    ondragstart="dragstart(event)"
    ondragend="dragend(event)"
    id="gate" >

<img id="ball" src="https://en.js.cx/clipart/ball.svg"
    draggable="true"
    ondragstart="dragstart(event)"
    ondragend="dragend(event)"
    height="150" width="150">



</body>
```

Ex 2:

```
<html>
    <head>
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
            <div class="empty">
                    <div class="fill" draggable="true"></div>
                </div>
        <div class="empty"></div>
        <div class="empty"></div>
```

```
        <div class="empty"></div>
        <div class="empty"></div>
        <script src="main.js"></script>
    </body>
</html>
```

Main.js

```
const fill=document.querySelector('.fill');
const empties=document.querySelectorAll('.empty');

fill.addEventListener('dragstart',dragStart);
fill.addEventListener('dragend',dragEnd);

for(const empty of empties){
    empty.addEventListener('dragover',dragOver);
    empty.addEventListener('drop',drop);
    empty.addEventListener('dragenter',dragEnter);
    empty.addEventListener('dragleave',dragLeave);
}

function dragStart(e){
    console.log("drag started....");
}
function dragEnd(e){
    console.log("drag end....");
    this.className='fill';
}
function dragOver(e){
    e.preventDefault();
    console.log("drag over....");
}
function dragEnter(e){
    console.log("drag enter....");
}
function dragLeave(e){
    console.log("drag leave....");
    this.className='empty';
}
function drop(e){
   // e.preventDefault();
    this.append(fill);
    console.log("drop....");
}
```

Index DB

The indexeddb is a new HTML5 concept to store the data inside user's browser. indexeddb is more power than local storage and useful for applications that requires to store large amount of the data. These applications can run more efficiency and load

faster.

Why to use indexeddb?

The W3C has announced that the Web SQL database is a deprecated local storage specification so web developer should not use this technology any more. indexeddb is an alternative for web SQL data base and more effective than older technologies.

Features

- it stores key-pair values
- it is not a relational database
- IndexedDB API is mostly asynchronous
- it is not a structured query language
- it has supported to access the data from same domain

Ex:

```
<!DOCTYPE html>

<html>
   <head>
      <meta http-equiv = "Content-Type" content = "text/html; charset = utf-8" />
      <script type = "text/javascript">

         //prefixes of implementation that we want to test
         window.indexedDB = window.indexedDB || window.mozIndexedDB ||
         window.webkitIndexedDB || window.msIndexedDB;

         //prefixes of window.IDB objects
         window.IDBTransaction = window.IDBTransaction ||
         window.webkitIDBTransaction || window.msIDBTransaction;
         window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange ||
         window.msIDBKeyRange

         if (!window.indexedDB) {
            window.alert("Your browser doesn't support a stable version of IndexedDB.")
         }

         const employeeData = [
            { id: "00-01", name: "shoiab", age: 45, email: "shoiab@hit.com" },
            { id: "00-02", name: "haaris", age: 14, email: "haaris@hit.com" }
         ];
         var db;
         var request = window.indexedDB.open("newDatabase", 1);

         request.onerror = function(event) {
            console.log("error: ");
         };

         request.onsuccess = function(event) {
            db = request.result;
```

```
            console.log("success: "+ db);
         };

         request.onupgradeneeded = function(event) {
            var db = event.target.result;
            var objectStore = db.createObjectStore("employee", {keyPath: "id"});

            for (var i in employeeData) {
               objectStore.add(employeeData[i]);
            }
         }

         function read() {
            var transaction = db.transaction(["employee"]);
            var objectStore = transaction.objectStore("employee");
            var request = objectStore.get("00-03");

            request.onerror = function(event) {
               alert("Unable to retrieve daa from database!");
            };

            request.onsuccess = function(event) {
               // Do something with the request.result!
               if(request.result) {
                  alert("Name: " + request.result.name + ",Age: " + request.result.age + ", Email: "
+ request.result.email);
               } else {
                  alert("hashim couldn't be found in your database!");
               }
            };
         }

         function readAll() {
            var objectStore = db.transaction("employee").objectStore("employee");

            objectStore.openCursor().onsuccess = function(event) {
               var cursor = event.target.result;

               if (cursor) {
alert("Name for id " + cursor.key + " is " + cursor.value.name + ",Age: " + cursor.value.age + ",
Email: " + cursor.value.email);
                  cursor.continue();
               } else {
                  alert("No more entries!");
               }
            };
         }

         function add() {
            var request = db.transaction(["employee"], "readwrite")
            .objectStore("employee")
            .add({ id: "00-03", name: "hashim", age: 12, email: "hashim@hit.com" });

            request.onsuccess = function(event) {
               alert("hashim has been added to your database.");
            };
```

```
            request.onerror = function(event) {
               alert("Unable to add data\r\nhashim  aready exist in your database! ");
            }
         }

         function remove() {
            var request = db.transaction(["employee"], "readwrite")
            .objectStore("employee")
            .delete("00-03");

            request.onsuccess = function(event) {
               alert("hashim entry has been removed from your database.");
            };
         }
      </script>

   </head>
   <body>
      <button onclick = "read()">Read </button>
      <button onclick = "readAll()">Read all </button>
      <button onclick = "add()">Add data </button>
      <button onclick = "remove()">Delete data </button>
   </body>
</html>
```