

# HTML5

**HTML5 is the latest and most enhanced version of HTML.**

**Technically, HTML is not a programming language, but rather a markup language.**

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

## Browser Support:

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

## New Features:

HTML5 introduces a number of new elements and attributes that helps in building a modern websites. Following are great features introduced in HTML5.

- **New Semantic Elements:** These are like <header>, <footer>, and <section>.
- **Forms 2.0:** Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- **Persistent Local Storage:** To achieve without resorting to third-party plugins.
- **WebSocket :** A a next-generation bidirectional communication technology for web applications.
- **Server-Sent Events:** HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
- **Canvas:** This supports a two-dimensional drawing surface that you can program with JavaScript.
- **Audio & Video:** You can embed audio or video on your web pages without resorting to third-party plugins.

- **Geolocation:** Now visitors can choose to share their physical location with your web application.
- **Microdata:** This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- **Drag and drop:** Drag and drop the items from one location to another location on a the same webpage.

The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML 4.

HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes,an attribute had to have a value and to close all empty elements.

But HTML5 is coming with lots of flexibility and would support the followings:

- Uppercase tag names.
- Quotes are optional for attributes.
- Attribute values are optional.
- Closing empty elements are optional.

## The DOCTYPE:

DOCTYPES in older versions of HTML were longer because the HTML language was SGML based and therefore required a reference to a DTD.

HTML 5 authors would use simple syntax to specify DOCTYPE as follows:

```
<!DOCTYPE html>
```

All the above syntax is case-insensitive.

## Character Encoding:

HTML 5 authors can use simple syntax to specify Character Encoding as follows:

```
<meta charset="UTF-8">
```

All the above syntax is case-insensitive.

## The <script> tag:

It's common practice to add a type attribute with a value of "text/javascript" to script elements as follows:

```
<script type="text/javascript" src="scriptfile.js"></script>
```

HTML 5 removes extra information required and you can use simply following syntax:

```
<script src="scriptfile.js"></script>
```

## The <link> tag:

So far you were writing <link> as follows:

```
<link rel="stylesheet" type="text/css" href="stylefile.css">
```

HTML 5 removes extra information required and you can use simply following syntax:

```
<link rel="stylesheet" href="stylefile.css">
```

## HTML5 Elements:

HTML5 elements are marked up using start tags and end tags. Tags are delimited using angle brackets with the tag name in between.

The difference between start tags and end tags is that the latter includes a slash before the tag name.

Following is the example of an HTML5 element:

```
<p>...</p>
```

HTML5 tag names are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.

Most of the elements contain some content like <p>...</p> contains a paragraph. Some elements, however, are forbidden from containing any content at all and these are known as void elements. For example, br, hr, link and meta etc.

## HTML5 Attributes:

Elements may contain attributes that are used to set various properties of an element.

Some attributes are defined globally and can be used on any element, while others are defined for specific elements only. All attributes have a name and a value and look like as shown below in the example.

Following is the example of an HTML5 attributes which illustrates how to mark up a div element with an attribute named class using a value of "example":

```
<div class="example">...</div>
```

Attributes may only be specified within start tags and must never be used in end tags.

.

## HTML5 Document:

The following tags have been introduced for better structure:

- **section:** This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- **article:** This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- **aside:** This tag represents a piece of content that is only slightly related to the rest of the page.
- **header:** This tag represents the header of a section.
- **footer:** This tag represents a footer for a section and can contain information about the author, copyright information, et cetera.
- **nav:** This tag represents a section of the document intended for navigation.
- **dialog:** This tag can be used to mark up a conversation.
- **figure:** This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

The markup for an HTML 5 document would look like the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>...</title>
</head>
<body>
  <header>...</header>
  <nav>...</nav>
  <article>
    <section>
```

```

    ...
    </section>
</article>
<aside>...</aside>
<footer>...</footer>
</body>

```

the following tags (elements) have been introduced in HTML5:

Tags (Elements)	Description
<article>	Represents an independent piece of content of a document, such as a blog entry or newspaper article
<aside >	Represents a piece of content that is only slightly related to the rest of the page.
<audio>	Defines an audio file.
<canvas>	This is used for rendering dynamic bitmap graphics on the fly, such as graphs or games.
<command>	Represents a command the user can invoke.
<datalist>	Together with the a new list attribute for input can be used to make comboboxes
<details>	Represents additional information or controls which the user can obtain on demand
<embed>	Defines external interactive content or plugin.
<figure>	Represents a piece of self-contained flow content, typically referenced as a single unit from the main flow of the document.
<footer>	Represents a footer for a section and can contain information about the author, copyright information, et cetera.
<header>	Represents a group of introductory or navigational aids.
<hgroup>	Represents the header of a section.
<keygen>	Represents control for key pair generation.
<mark>	Represents a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context.
<meter>	Represents a measurement, such as disk usage.
<nav>	Represents a section of the document intended for navigation.
<output>	Represents some type of output, such as from a calculation done

	through scripting.
<progress>	Represents a completion of a task, such as downloading or when performing a series of expensive operations.
<ruby>	Together with <rt> and <rp> allow for marking up ruby annotations.
<section>	Represents a generic document or application section
<time>	Represents a date and/or time.
<video>	Defines a video file.
<wbr>	Represents a line break opportunity.

## New types for <input> tag:

The input element's type attribute now has the following new values:

Type	Description
color	Color selector, which could be represented by a wheel or swatch picker
date	Selector for calendar date
datetime-local	Date and time display, with no setting or indication for time zones
datetime	Full date and time display, including a time zone.
email	Input type should be an email.
month	Selector for a month within a given year
number	A field containing a numeric value only
range	Numeric selector within a range of values, typically visualized as a slider
search	Term to supply to a search engine. For example, the search bar atop a browser.
tel	Input type should be telephone number.
time	Time indicator and selector, with no time zone information
url	Input type should be URL type.
week	Selector for a week within a given year

# HTML5 - Audio & Video

HTML5 features, include native audio and video support without the need for Flash.

The HTML5 `<audio>` and `<video>` tags make it simple to add media to a website. You need to set **src** attribute to identify the media source and include a controls attribute so the user can play and pause the media.

## Embedding Video:

Here is the simplest form of embedding a video file in your webpage:

```
<video src="foo.mp4" width="300" height="200" controls>
  Your browser does not support the <video> element.
</video>
```

The current HTML5 draft specification does not specify which video formats browsers should support in the video tag. But most commonly used video formats are:

1. **Ogg:** Ogg files with Theora video codec and Vorbis audio codec.
2. **mpeg4:** MPEG4 files with H.264 video codec and AAC audio codec.

You can use `<source>` tag to specify media along with media type and many other attributes. A video element allows multiple source elements and browser will use the first recognized format:

```
<!DOCTYPE HTML>
<html>
<body>
  <video width="300" height="200" controls autoplay>
    <source src="/html5/foo.ogg" type="video/ogg" />
    <source src="/html5/foo.mp4" type="video/mp4" />
    Your browser does not support the <video> element.
  </video>
</body>
</html>
```

## Video Attribute Specification:

The HTML5 video tag can have a number of attributes to control the look and feel and various functionalities of the control:

Attribute	Description
-----------	-------------

autoplay	This boolean attribute if specified, the video will automatically begin to play back as soon as it can do so without stopping to finish loading the data.
autobuffer	This boolean attribute if specified, the video will automatically begin buffering even if it's not set to automatically play.
controls	If this attribute is present, it will allow the user to control video playback, including volume, seeking, and pause/resume playback.
height	This attribut specifies the height of the video's display area, in CSS pixels.
loop	This boolean attribute if specified, will allow video automatically seek back to the start after reaching at the end.
preload	This attribute specifies that the video will be loaded at page load, and ready to run. Ignored if autoplay is present.
poster	This is a URL of an image to show until the user plays or seeks.
src	The URL of the video to embed. This is optional; you may instead use the <source> element within the video block to specify the video to embed
width	This attribut specifies the width of the video's display area, in CSS pixels.

## Embedding Audio:

HTML5 supports <audio> tag which is used to embed sound content in an HTML or XHTML document as follows.

```
<audio src="foo.wav" controls autoplay>
  Your browser does not support the <audio> element.
</audio>
```

The current HTML5 draft specification does not specify which audio formats browsers should support in the audio tag. But most commonly used audio formats are **ogg**, **mp3** and **wav**.

You can use <source> tag to specify media along with media type and many other attributes. An audio element allows multiple source elements and browser will use the first recognized format:



```
<!DOCTYPE HTML>
<html>
<body>
  <audio controls autoplay>
    <source src="/html5/audio.ogg" type="audio/ogg" />
    <source src="/html5/audio.wav" type="audio/wav" />
    Your browser does not support the <audio> element.
  </audio>
</body>
</html>
```

To learn above concept - [Do Online Practice](#) using latest version of either Safari or Opera.

### Audio Attribute Specification:

The HTML5 audio tag can have a number of attributes to control the look and feel and various functionalities of the control:

Attribute	Description
autoplay	This boolean attribute if specified, the audio will automatically begin to play back as soon as it can do so without stopping to finish loading the data.
autobuffer	This boolean attribute if specified, the audio will automatically begin buffering even if it's not set to automatically play.
controls	If this attribute is present, it will allow the user to control audio playback, including volume, seeking, and pause/resume playback.
loop	This boolean attribute if specified, will allow audio automatically seek back to the start after reaching at the end.
preload	This attribute specifies that the audio will be loaded at page load, and ready to run. Ignored if autoplay is present.
src	The URL of the audio to embed. This is optional; you may instead use the <source> element within the video block to specify the video to embed

## Handling Media Events:

The HTML5 audio and video tag can have a number of attributes to control various functionalities of the control using Javascript:

Event	Description
abort	This event is generated when playback is aborted.
canplay	This event is generated when enough data is available that the media can be played.
ended	This event is generated when playback completes.
error	This event is generated when an error occurs.
loadeddata	This event is generated when the first frame of the media has finished loading.
loadstart	This event is generated when loading of the media begins.
pause	This event is generated when playback is paused.
play	This event is generated when playback starts or resumes.
progress	This event is generated periodically to inform the progress of the downloading the media.
ratechange	This event is generated when the playback speed changes.
seeked	This event is generated when a seek operation completes.
seeking	This event is generated when a seek operation begins.
suspend	This event is generated when loading of the media is suspended.
volumechange	This event is generated when the audio volume changes.
waiting	This event is generated when the requested operation (such as playback) is delayed pending the completion of another operation (such as a seek).

Following is the example which allows to play the given video:

```
<!DOCTYPE HTML>
<head>
<script type="text/javascript">
function PlayVideo(){
    var v = document.getElementsByTagName("video")[0];
    v.play();
}
</script>
</head>
<html>
<body>
    <form>
    <video width="300" height="200" src="/html5/foo.mp4">
        Your browser does not support the <video> element.
    </video>
    <input type="button" onclick="PlayVideo();" value="Play"/>
    </form>
</body>
</html>
```

## Configuring Servers for Media Type:

Most servers don't by default serve Ogg or mp4 media with the correct MIME types, so you'll likely need to add the appropriate configuration for this.

```
AddType audio/ogg .oga
AddType audio/wav .wav
AddType video/ogg .ogv .ogg
AddType video/mp4 .mp4
```

## The <input> element in HTML5

Apart from the above mentioned attributes, HTML5 input elements introduced several new values for the **type** attribute. These are listed below.

**NOTE:** Try all the following example using latest version of **Opera** browser.

Type	Description
<a href="#">datetime</a>	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
<a href="#">datetime-local</a>	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information.

date	A date (year, month, day) encoded according to ISO 8601.
month	A date consisting of a year and a month encoded according to ISO 8601.
week	A date consisting of a year and a week number encoded according to ISO 8601.
time	A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
number	This accepts only numerical value. The step attribute specifies the precision, defaulting to 1.
range	The range type is used for input fields that should contain a value from a range of numbers.
email	This accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.
Url	This accepts only URL value. This type is used for input fields that should contain a URL address. If you try to submit a simple text, it forces to enter only URL address either in http://www.example.com format or in http://example.com format.

## HTML5 - Canvas

HTML5 element `<canvas>` gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.

Here is a simple `<canvas>` element which has only two specific attributes **width** and **height** plus all the core HTML5 attributes like id, name and class etc.

```
<canvas id="mycanvas" width="100" height="100"></canvas>
```

You can easily find that `<canvas>` element in the DOM using `getElementById()` method as follows:

```
var canvas = document.getElementById("mycanvas");
```

Let us see a simple example on using <canvas> element in HTML5 document.

```
<!DOCTYPE HTML>
<html>
<head>
<style>
#mycanvas{
    border:1px solid red;
}
</style>
</head>
<body>
    <canvas id="mycanvas" width="100" height="100"></canvas>
</body>
</html>
```

## The Rendering Context:

The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it.

The canvas element has a DOM method called **getContext**, used to obtain the rendering context and its drawing functions. This function takes one parameter, the type of context **2d**.

Following is the code to get required context along with a check if your browser supports <canvas> element:

```
var canvas = document.getElementById("mycanvas");
if (canvas.getContext){
    var ctx = canvas.getContext('2d');
    // drawing code here
} else {
    // canvas-unsupported code here
}
```

## Browser Support

The latest versions of Firefox, Safari, Chrome and Opera all support for HTML5 Canvas but IE8 does not support canvas natively.

You can use [ExplorerCanvas](#) to have canvas support through Internet Explorer. You just need to include this javascript as follows:

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

# HTML5 Canvas Examples:

This tutorial covers following examples related to HTML5 <canvas> element.

Examples	Description
Drawing Rectangles	Learn how to draw rectangle using HTML5 <canvas> element
Drawing Paths	Learn how to make shapes using paths in HTML5 <canvas> element
Drawing Lines	Learn how to draw lines using HTML5 <canvas> element
Drawing Bezier	Learn how to draw bezier curve using HTML5 <canvas> element
Drawing Quadratic	Learn how to draw quadratic curve using HTML5 <canvas> element
Using Images	Learn how to use images with HTML5 <canvas> element
Create Gradients	Learn how to create gradients using HTML5 <canvas> element
Styles and Colors	Learn how to apply styles and colors using HTML5 <canvas> element
Text and Fonts	Learn how to draw amazing text using different fonts and their size.
Pattern and Shadow	Learn how to draw different patterns and drop shadows.
Canvas States	Learn how to save and restore canvas states while doing complex drawings on a canvas.
Canvas Translation	This method is used to move the canvas and its origin to a different point in the grid.
Canvas Rotation	This method is used to rotate the canvas around the current origin.
Canvas Scaling	This method is used to increase or decrease the units in a canvas grid.
Canvas Transform	These methods allow modifications directly to the transformation matrix.
Canvas Composition	This method is used to mask off certain areas or clear sections from the canvas.
Canvas Animation	Learn how to create basic animation using HTML5 canvas and Javascript.

# HTML5 Canvas - Drawing Rectangles

There are three methods that draw rectangles on the canvas:

SN	Method and Description
1	<b>fillRect(x,y,width,height)</b> This method draws a filled rectangle.
2	<b>strokeRect(x,y,width,height)</b> This method draws a rectangular outline.
3	<b>clearRect(x,y,width,height)</b> This method clears the specified area and makes it fully transparent

Here x and y specify the position on the canvas (relative to the origin) of the top-left corner of the rectangle and *width* and *height* are width and height of the rectangle.

## Example:

Following is a simple example which makes use of above mentioned methods to draw a nice rectangle.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // Get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

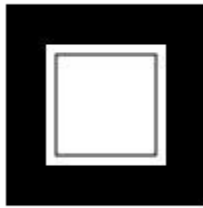
    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Draw shapes
        ctx.fillRect(25,25,100,100);
        ctx.clearRect(45,45,60,60);
        ctx.strokeRect(50,50,50,50);
    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

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

This would draw following rectangle:



## HTML5 Canvas - Drawing Paths

here are following methods required to draw paths on the canvas:

SN	Method and Description
1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	<b>fill()</b> This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>arc(x, y, radius, startAngle, endAngle, anticlockwise)</b> Adds points to the subpath such that the arc described by the circumference of the circle described by the arguments, starting at the given start angle and ending at the given end angle, going in the given direction, is added to the path, connected to the previous point by a straight line.

## Example:

Following is a simple example which makes use of above mentioned methods to draw a shape.



```

<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Draw shapes
        ctx.beginPath();
        ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
        ctx.moveTo(110,75);
        ctx.arc(75,75,35,0,Math.PI,false); // Mouth
        ctx.moveTo(65,65);
        ctx.arc(60,65,5,0,Math.PI*2,true); // Left eye
        ctx.moveTo(95,65);
        ctx.arc(90,65,5,0,Math.PI*2,true); // Right eye
        ctx.stroke();

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

The above example would draw following shape:



## HTML5 Canvas - Drawing Lines

### Line Methods:

There are following methods required to draw lines on the canvas:

SN	Method and Description
1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	<b>fill()</b> This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>lineTo(x, y)</b> This method adds the given point to the current subpath, connected to the previous one by a straight line.

## Example:

Following is a simple example which makes use of above mentioned methods to draw a triangle.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext) {

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Filled triangle
        ctx.beginPath();
        ctx.moveTo(25,25);
        ctx.lineTo(105,25);
        ctx.lineTo(25,105);
        ctx.fill();

        // Stroked triangle
```

```

    ctx.beginPath();
    ctx.moveTo(125,125);
    ctx.lineTo(125,45);
    ctx.lineTo(45,125);
    ctx.closePath();
    ctx.stroke();

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

    }
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>
```

The above example would draw following shape:



## Line Properties:

There are several properties which allow us to style lines.

SN	Property and Description
1	<b>lineWidth [ = value ]</b> This property returns the current line width and can be set, to change the line width.
2	<b>lineCap [ = value ]</b> This property returns the current line cap style and can be set, to change the line cap style. The possible line cap styles are <i>butt</i> , <i>round</i> , and <i>square</i>
3	<b>lineJoin [ = value ]</b> This property returns the current line join style and can be set, to change the line join style. The possible line join styles are <i>bevel</i> , <i>round</i> , and <i>miter</i> .

4	<b>miterLimit [ = value ]</b> This property returns the current miter limit ratio and can be set, to change the miter limit ratio.
---	---

## Example:

Following is a simple example which makes use of *lineWidth* property to draw lines of different width.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

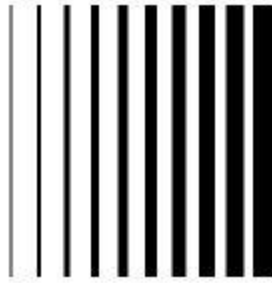
    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext) {

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        for (i=0;i<10;i++){
            ctx.lineWidth = 1+i;
            ctx.beginPath();
            ctx.moveTo(5+i*14,5);
            ctx.lineTo(5+i*14,140);
            ctx.stroke();
        }
    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```
    }
}
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>
```

The above example would draw following shape:



## HTML5 Canvas - Drawing Bezier Curves

here are following methods required to draw bezier on the canvas:

SN	Method and Description
1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	<b>fill()</b> This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)</b> This method adds the given point to the current path, connected to the previous one by a cubic Bezier curve with the given control points.

The x and y parameters in `bezierCurveTo()` method are the coordinates of the end point. `cp1x` and `cp1y` are the coordinates of the first control point, and `cp2x` and `cp2y` are the coordinates of the second control point.

### Example:

Following is a simple example which makes use of above mentioned methods to draw a Bezier curves.

```
<!DOCTYPE HTML>
```

```

<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        ctx.beginPath();
        ctx.moveTo(75,40);
        ctx.bezierCurveTo(75,37,70,25,50,25);
        ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
        ctx.bezierCurveTo(20,80,40,102,75,120);
        ctx.bezierCurveTo(110,102,130,80,130,62.5);
        ctx.bezierCurveTo(130,62.5,130,25,100,25);
        ctx.bezierCurveTo(85,25,75,37,75,40);
        ctx.fill();

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

The above example would draw following shape:



## HTML5 Canvas - Drawing Quadratic Curves

There are following methods required to draw quadratic curve on the canvas:

SN	Method and Description
----	------------------------

1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	<b>fill()</b> This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>quadraticCurveTo(cpx, cpy, x, y)</b> This method adds the given point to the current path, connected to the previous one by a quadratic Bezier curve with the given control point.

The x and y parameters in quadraticCurveTo() method are the coordinates of the end point. cpx and cpy are the coordinates of the control point.

## Example:

Following is a simple example which makes use of above mentioned methods to draw a Quadratic curves.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        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.stroke();

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

    }
}
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>

```

The above example would draw following shape:



## HTML5 Canvas - Using Images

This tutorial would show how to import and external image into a canvas and then how to draw on that image by using following methods:

SN	Method and Description
1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	<b>fill()</b> This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>drawImage(image, dx, dy)</b>



	This method draws the given image onto the canvas. Here <i>image</i> is a reference to an image or canvas object. x and y form the coordinate on the target canvas where our image should be placed.
--	--

## Example:

Following is a simple example which makes use of above mentioned methods to import an draw on an image.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Draw shapes
        var img = new Image();
        img.src = '/images/backdrop.jpg';

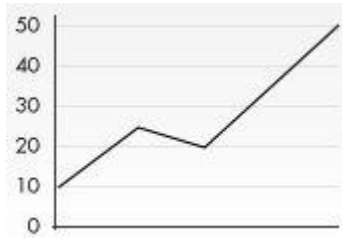
        img.onload = function(){
            ctx.drawImage(img,0,0);
            ctx.beginPath();
            ctx.moveTo(30,96);
            ctx.lineTo(70,66);
            ctx.lineTo(103,76);
            ctx.lineTo(170,15);
            ctx.stroke();
        }

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

Assuming we have following image to import *images/backdrop.jpg*.



The above example would draw following shape:



## HTML5 Canvas - Create Gradients

HTML5 canvas allows us to fill and stroke shapes using linear and radial gradients using the following methods:

SN	Method and Description
1	<b>addColorStop(offset, color)</b> This method adds a color stop with the given color to the gradient at the given offset. Here 0.0 is the offset at one end of the gradient, 1.0 is the offset at the other end.
2	<b>createLinearGradient(x0, y0, x1, y1)</b> This method returns a CanvasGradient object that represents a linear gradient that paints along the line given by the coordinates represented by the arguments. The four arguments represent the starting point (x1,y1) and end point (x2,y2) of the gradient.
3	<b>createRadialGradient(x0, y0, r0, x1, y1, r1)</b> This method returns a CanvasGradient object that represents a radial gradient that paints along the cone given by the circles represented by the arguments. The first three arguments define a circle with coordinates (x1,y1) and radius r1 and the second a circle with coordinates (x2,y2) and radius r2.

## Linear Gradient Example:

Following is a simple example which makes use of above mentioned methods to create Linear gradient.

```

<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

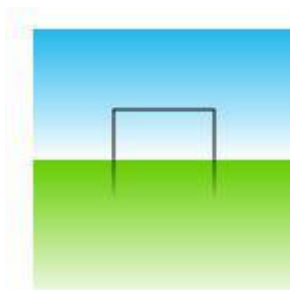
        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Create Linear Gradients
        var lingrad = ctx.createLinearGradient(0,0,0,150);
        lingrad.addColorStop(0, '#00ABEB');
        lingrad.addColorStop(0.5, 'fff');
        lingrad.addColorStop(0.5, '#66CC00');
        lingrad.addColorStop(1, 'fff');

        var lingrad2 = ctx.createLinearGradient(0,50,0,95);
        lingrad2.addColorStop(0.5, '#000');
        lingrad2.addColorStop(1, 'rgba(0,0,0,0)');

        // assign gradients to fill and stroke styles
        ctx.fillStyle = lingrad;
        ctx.strokeStyle = lingrad2;
        // draw shapes
        ctx.fillRect(10,10,130,130);
        ctx.strokeRect(50,50,50,50);
    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

The above example would produce following result



To learn above concept - [Do Online Practice](#) using latest version of either Safari or Opera.

## Radial Gradient Example:

Following is a simple example which makes use of above mentioned methods to create Radial gradient.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Create gradients
        var radgrad = ctx.createRadialGradient(45,45,10,52,50,30);
        radgrad.addColorStop(0, '#A7D30C');
        radgrad.addColorStop(0.9, '#019F62');
        radgrad.addColorStop(1, 'rgba(1,159,98,0)');

        var radgrad2 = ctx.createRadialGradient(105,105,20,112,120,50);
        radgrad2.addColorStop(0, '#FF5F98');
        radgrad2.addColorStop(0.75, '#FF0188');
        radgrad2.addColorStop(1, 'rgba(255,1,136,0)');

        var radgrad3 = ctx.createRadialGradient(95,15,15,102,20,40);
        radgrad3.addColorStop(0, '#00C9FF');
        radgrad3.addColorStop(0.8, '#00B5E2');
        radgrad3.addColorStop(1, 'rgba(0,201,255,0)');

        var radgrad4 = ctx.createRadialGradient(0,150,50,0,140,90);
        radgrad4.addColorStop(0, '#F4F201');
        radgrad4.addColorStop(0.8, '#E4C700');
        radgrad4.addColorStop(1, 'rgba(228,199,0,0)');

        // draw shapes
        ctx.fillStyle = radgrad4;
        ctx.fillRect(0,0,150,150);
        ctx.fillStyle = radgrad3;
        ctx.fillRect(0,0,150,150);
        ctx.fillStyle = radgrad2;
        ctx.fillRect(0,0,150,150);
        ctx.fillStyle = radgrad;
        ctx.fillRect(0,0,150,150);
    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

    }
}
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>

```

The above example would produce following result



## HTML5 Canvas - Styles and Colors

HTML5 canvas provides following two important properties to apply colors to a shape:

SN	Method and Description
1	<b>fillStyle</b> This attribute represents the color or style to use inside the shapes.
2	<b>strokeStyle</b> This attribute represents the color or style to use for the lines around shapes

By default, the stroke and fill color are set to black which is CSS color value #000000.

### A fillStyle Example:

Following is a simple example which makes use of above mentioned fillStyle attribute to create a nice pattern.

```

<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM

```

```

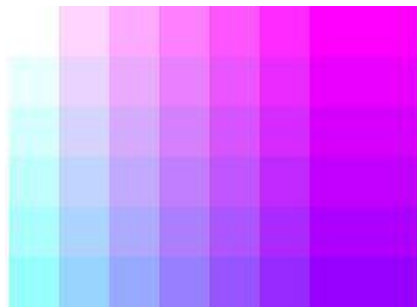
var canvas = document.getElementById('mycanvas');

// Make sure we don't execute when canvas isn't supported
if (canvas.getContext){

    // use getContext to use the canvas for drawing
    var ctx = canvas.getContext('2d');

    // Create a pattern
    for (var i=0;i<7;i++){
        for (var j=0;j<7;j++){
            ctx.fillStyle='rgb(' + Math.floor(255-20.5*i)+ ',' +
                Math.floor(255 - 42.5*j) + ',255)';
            ctx.fillRect( j*25, i* 25, 55, 55 );
        }
    }
} else {
    alert('You need Safari or Firefox 1.5+ to see this demo.');
```

The above example would produce following result



To learn above concept - [Do Online Practice](#) using latest version of either Safari or Opera.

## A strokeStyle Example:

Following is a simple example which makes use of above mentioned fillStyle attribute to create another nice pattern.

```

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

```

function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // Create a pattern
        for (var i=0;i<10;i++){
            for (var j=0;j<10;j++){
                ctx.strokeStyle='rgb(255,'+ Math.floor(50-2.5*i)+'+', '+
                    Math.floor(155 - 22.5 * j ) + ')+'';
                ctx.beginPath();
                ctx.arc(1.5+j*25, 1.5 + i*25,10,10,Math.PI*5.5, true);
                ctx.stroke();
            }
        }
    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

The above example would produce following result



## HTML5 Canvas - Text and Fonts

HTML5 canvas provides capabilities to create text using different font and text properties listed below:

SN	Property and Description
----	--------------------------

1	<b>font [ = value ]</b> This property returns the current font settings and can be set, to change the font.
2	<b>textAlign [ = value ]</b> This property returns the current text alignment settings and can be set, to change the alignment. The possible values are <b>start</b> , <b>end</b> , <b>left</b> , <b>right</b> , and <b>center</b> .
3	<b>textBaseline [ = value ]</b> This property returns the current baseline alignment settings and can be set, to change the baseline alignment. The possible values are <b>top</b> , <b>hanging</b> , <b>middle</b> , <b>alphabetic</b> , <b>ideographic</b> and <b>bottom</b>
4	<b>fillText(text, x, y [, maxWidth ] )</b> This property fills the given <i>text</i> at the given position indicated by the given coordinates x and y.
5	<b>strokeText(text, x, y [, maxWidth ] )</b> This property strokes the given <i>text</i> at the given position indicated by the given coordinates x and y.

## Example:

Following is a simple example which makes use of above mentioned attributes to draw a text:

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        ctx.fillStyle      = '#00F';
        ctx.font            = 'Italic 30px Sans-Serif';
        ctx.textBaseline    = 'Top';
        ctx.fillText        ('Hello world!', 40, 100);

        ctx.font            = 'Bold 30px Sans-Serif';
        ctx.strokeText      ('Hello world!', 40, 50);

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```




```

</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>

```

The above example would produce following result:



## HTML5 Canvas - Pattern and Shadow

### Create Pattern:

There is following method required to create a pattern on the canvas:

SN	Method and Description
1	<b>createPattern(image, repetition)</b> This method will use <i>image</i> to create the pattern. The second argument could be a string with one of the following values: <b>repeat</b> , <b>repeat-x</b> , <b>repeat-y</b> , and <b>no-repeat</b> . If the empty string or null is specified, repeat will be assumed.

### Example:

Following is a simple example which makes use of above mentioned method to create a nice pattern.

```

<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported

```

```

if (canvas.getContext){

    // use getContext to use the canvas for drawing
    var ctx = canvas.getContext('2d');

    // create new image object to use as pattern
    var img = new Image();
    img.src = 'images/pattern.jpg';
    img.onload = function(){
        // create pattern
        var ptrn = ctx.createPattern(img, 'repeat');
        ctx.fillStyle = ptrn;
        ctx.fillRect(0,0,150,150);
    }

} else {
    alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

}
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>

```

Assuming we have following pattern *images/pattern.jpg*.



The above example would draw following result:



## Create Shadows:

HTML5 canvas provides capabilities to create nice shadows around the drawings. All drawing operations are affected by the four global shadow attributes

SN	Property and Description
1	<b>shadowColor [ = value ]</b> This property returns the current shadow color and can be set, to change the shadow color.
2	<b>shadowOffsetX [ = value ]</b> This property returns the current shadow offset X and can be set, to change the shadow offset X.
3	<b>shadowOffsetY [ = value ]</b> This property returns the current shadow offset Y and can be set, change the shadow offset Y.
4	<b>shadowBlur [ = value ]</b> This property returns the current level of blur applied to shadows and can be set, to change the blur level.

## Example:

Following is a simple example which makes use of above mentioned attributes to draw a shadow.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        ctx.shadowOffsetX = 2;
        ctx.shadowOffsetY = 2;
        ctx.shadowBlur = 2;
        ctx.shadowColor = "rgba(0, 0, 0, 0.5)";

        ctx.font = "20px Times New Roman";
        ctx.fillStyle = "Black";
        ctx.fillText("This is shadow test", 5, 30);

    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>
```

The above example would produce following result:

**This is shadow test**

## HTML5 Canvas - Save and Restore States

HTML5 canvas provides two important methods to save and restore the canvas states. The canvas drawing state is basically a snapshot of all the styles and transformations that have been applied and consists of the followings:

1. The transformations such as translate, rotate and scale etc.
2. The current clipping region.
3. The current values of the following attributes: *strokeStyle*, *fillStyle*, *globalAlpha*, *lineWidth*, *lineCap*, *lineJoin*, *miterLimit*, *shadowOffsetX*, *shadowOffsetY*, *shadowBlur*, *shadowColor*, *globalCompositeOperation*, *font*, *textAlign*, *textBaseline*.

Canvas states are stored on a stack every time the **save** method is called, and the last saved state is returned from the stack every time the **restore** method is called.

SN	Method and Description
1	<b>save()</b> This method pushes the current state onto the stack..
2	<b>restore()</b> This method pops the top state on the stack, restoring the context to that state.

## Example:

Following is a simple example which makes use of above mentioned methods to show how the *restore* is called, to restore the original state and the last rectangle is once again drawn in black.

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

```

function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        // draw a rectangle with default settings
        ctx.fillRect(0,0,150,150);
        // Save the default state
        ctx.save();

        // Make changes to the settings
        ctx.fillStyle = '#66FFFF'
        ctx.fillRect( 15,15,120,120);
        // Save the current state
        ctx.save();

        // Make the new changes to the settings
        ctx.fillStyle = '#993333'
        ctx.globalAlpha = 0.5;
        ctx.fillRect(30,30,90,90);

        // Restore previous state
        ctx.restore();
        // Draw a rectangle with restored settings
        ctx.fillRect(45,45,60,60);

        // Restore original state
        ctx.restore();
        // Draw a rectangle with restored settings
        ctx.fillRect(40,40,90,90);

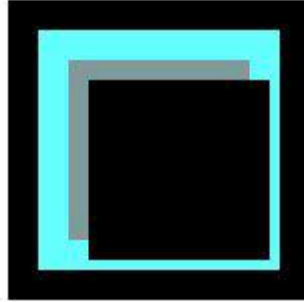
    } else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

    }
}
</script>
</head>
<body onload="drawShape();">
    <canvas id="mycanvas"></canvas>
</body>
</html>

```

The above example would produce following result:



## HTML5 Canvas - Translation

HTML5 canvas provides **translate(x, y)** method which is used to move the canvas and its origin to a different point in the grid.

Here argument x is the amount the canvas is moved to the left or right, and y is the amount it's moved up or down

### Example:

Following is a simple example which makes use of above method to draw various Spirographs:

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

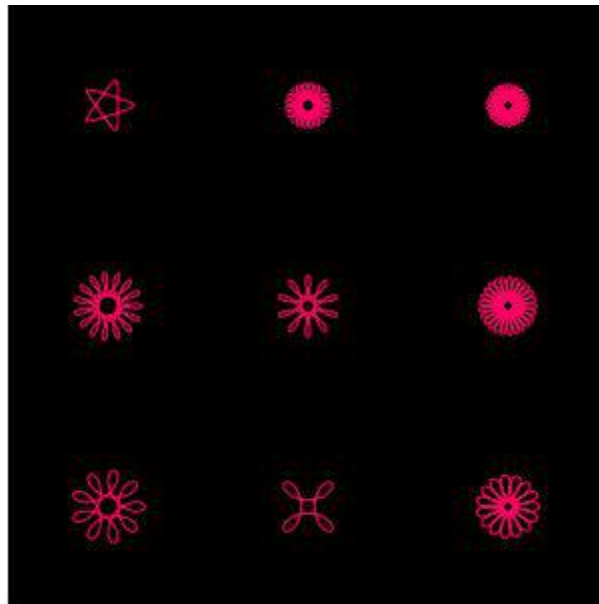
    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){
        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');
        ctx.fillRect(0,0,300,300);
        for (i=0;i<3;i++) {
            for (j=0;j<3;j++) {
                ctx.save();
                ctx.strokeStyle = "#FF0066";
                ctx.translate(50+j*100,50+i*100);
                drawSpirograph(ctx,10*(j+3)/(j+2),-2*(i+3)/(i+1),10);
                ctx.restore();
            }
        }
    }
    else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

function drawSpirograph(ctx,R,r,O){
  var x1 = R-O;
  var y1 = 0;
  var i = 1;
  ctx.beginPath();
  ctx.moveTo(x1,y1);
  do {
    if (i>20000) break;
    var x2 = (R+r)*Math.cos(i*Math.PI/72) -
              (r+O)*Math.cos(((R+r)/r)*(i*Math.PI/72))
    var y2 = (R+r)*Math.sin(i*Math.PI/72) -
              (r+O)*Math.sin(((R+r)/r)*(i*Math.PI/72))
    ctx.lineTo(x2,y2);
    x1 = x2;
    y1 = y2;
    i++;
  } while (x2 != R-O && y2 != 0 );
  ctx.stroke();
}
</script>
</head>
<body onload="drawShape();">
  <canvas id="mycanvas" width="400" height="400"></canvas>
</body>
</html>

```

The above example would produce following result:



## HTML5 Canvas - Rotation

HTML5 canvas provides **rotate(angle)** method which is used to rotate the canvas around the current origin.

This method only takes one parameter and that's the angle the canvas is rotated by. This is a clockwise rotation measured in radians.

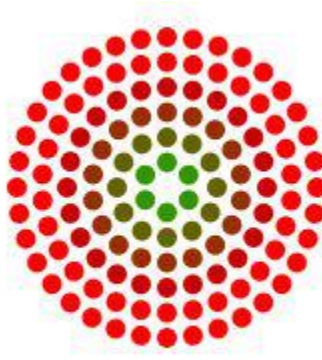
## Example:

Following is a simple example which we are running two loops where first loop determines the number of rings, and the second determines the number of dots drawn in each ring.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');
    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){
        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');
        ctx.translate(100,100);
        for (i=1; i<7; i++){
            ctx.save();
            ctx.fillStyle = 'rgb('+ (51*i)+', '+ (200-51*i)+', 0)';
            for (j=0; j < i*6; j++){
                ctx.rotate(Math.PI*2/(i*6));
                ctx.beginPath();
                ctx.arc(0,i*12.5,5,0,Math.PI*2,true);
                ctx.fill();
            }
            ctx.restore();
        }
    }else {
        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

The above example would produce following result:





## HTML5 Canvas - Scaling

HTML5 canvas provides **scale(x, y)** method which is used to increase or decrease the units in our canvas grid. This can be used to draw scaled down or enlarged shapes and bitmaps.

This method takes two parameters where x is the scale factor in the horizontal direction and y is the scale factor in the vertical direction. Both parameters must be positive numbers.

Values smaller than 1.0 reduce the unit size and values larger than 1.0 increase the unit size. Setting the scaling factor to precisely 1.0 doesn't affect the unit size.

### Example:

Following is a simple example which uses spirograph function to draw nine shapes with different scaling factors.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){
        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');
        ctx.strokeStyle = "#fc0";
        ctx.lineWidth = 1.5;
        ctx.fillRect(0,0,300,300);

        // Uniform scaling
        ctx.save()
        ctx.translate(50,50);
```

```

drawSpirograph(ctx,22,6,5);

ctx.translate(100,0);
ctx.scale(0.75,0.75);
drawSpirograph(ctx,22,6,5);

ctx.translate(133.333,0);
ctx.scale(0.75,0.75);
drawSpirograph(ctx,22,6,5);
ctx.restore();

// Non uniform scaling (y direction)
ctx.strokeStyle = "#0cf";
ctx.save()
ctx.translate(50,150);
ctx.scale(1,0.75);
drawSpirograph(ctx,22,6,5);

ctx.translate(100,0);
ctx.scale(1,0.75);
drawSpirograph(ctx,22,6,5);

ctx.translate(100,0);
ctx.scale(1,0.75);
drawSpirograph(ctx,22,6,5);
ctx.restore();

// Non uniform scaling (x direction)
ctx.strokeStyle = "#cf0";
ctx.save()
ctx.translate(50,250);
ctx.scale(0.75,1);
drawSpirograph(ctx,22,6,5);

ctx.translate(133.333,0);
ctx.scale(0.75,1);
drawSpirograph(ctx,22,6,5);

ctx.translate(177.777,0);
ctx.scale(0.75,1);
drawSpirograph(ctx,22,6,5);
ctx.restore();

}else {
    alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

}
```

```

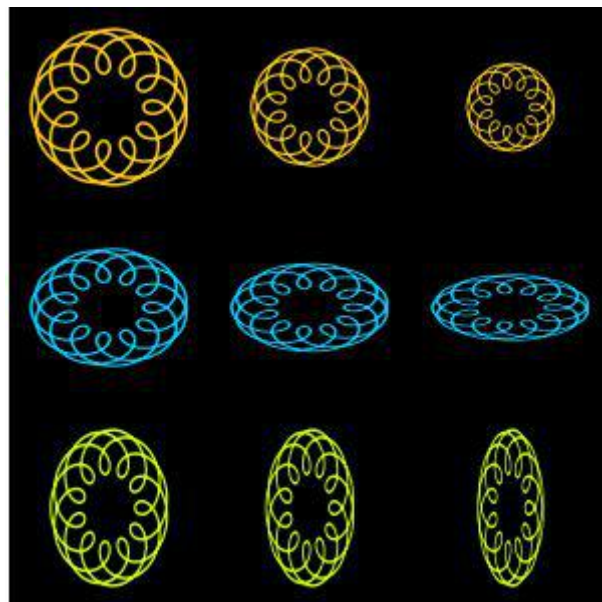
function drawSpirograph(ctx,R,r,0){
    var x1 = R-0;
    var y1 = 0;
    var i = 1;
    ctx.beginPath();
    ctx.moveTo(x1,y1);
    do {
        if (i>20000) break;
```

```

var x2 = (R+r)*Math.cos(i*Math.PI/72) -
          (r+0)*Math.cos(((R+r)/r)*(i*Math.PI/72))
var y2 = (R+r)*Math.sin(i*Math.PI/72) -
          (r+0)*Math.sin(((R+r)/r)*(i*Math.PI/72))
ctx.lineTo(x2,y2);
x1 = x2;
y1 = y2;
i++;
} while (x2 != R-0 && y2 != 0 );
ctx.stroke();
}
</script>
</head>
<body onload="drawShape();">
  <canvas id="mycanvas" width="400" height="400"></canvas>
</body>
</html>

```

The above example would produce following result:



## HTML5 Canvas - Transforms

HTML5 canvas provides methods which allow modifications directly to the transformation matrix. The transformation matrix must initially be the identity transform. It may then be adjusted using the transformation methods.

SN	Method and Description
1	<b>transform(m11, m12, m21, m22, dx, dy)</b> This method changes the transformation matrix to apply the matrix given by the

	arguments.
2	<b>setTransform(m11, m12, m21, m22, dx, dy)</b> This method changes the transformation matrix to the matrix given by the arguments .

The `transform(m11, m12, m21, m22, dx, dy)` method must multiply the current transformation matrix with the matrix described by:

```
m11    m21    dx
m12    m22    dy
0      0      1
```

The `setTransform(m11, m12, m21, m22, dx, dy)` method must reset the current transform to the identity matrix, and then invoke the `transform(m11, m12, m21, m22, dx, dy)` method with the same arguments.

## Example:

Following is a simple example which makes use of `transform()` and `setTransform()` methods:

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape() {
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){
        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        var sin = Math.sin(Math.PI/6);
        var cos = Math.cos(Math.PI/6);
        ctx.translate(200, 200);
        var c = 0;

        for (var i=0; i <= 12; i++) {
            c = Math.floor(255 / 12 * i);
            ctx.fillStyle = "rgb(" + c + "," + c + "," + c + ")";
            ctx.fillRect(0, 0, 100, 100);
            ctx.transform(cos, sin, -sin, cos, 0, 0);
        }

        ctx.setTransform(-1, 0, 0, 1, 200, 200);
        ctx.fillStyle = "rgba(100, 100, 255, 0.5)";
        ctx.fillRect(50, 50, 100, 100);
    }else {
```

```

        alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```
    }
```

```
}
```

```
</script>
```

```
</head>
```

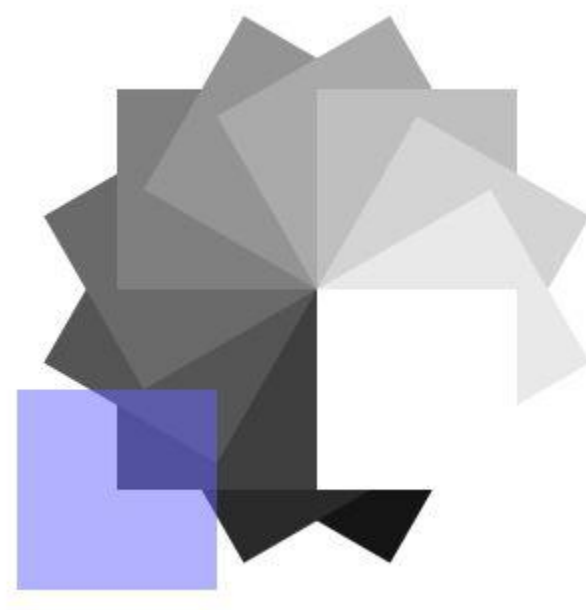
```
<body onload="drawShape();">
```

```
    <canvas id="mycanvas" width="400" height="400"></canvas>
```

```
</body>
```

```
</html>
```

The above example would produce following result:



## HTML5 Canvas - Composition

HTML5 canvas provides compositing attribute **globalCompositeOperation** which affect all the drawing operations.

We can draw new shapes behind existing shapes and mask off certain areas, clear sections from the canvas using globalCompositeOperation attribute as shown below in the example.

There are following values which can be set for globalCompositeOperation:

Attribute	Description
source-over	This is the default setting and draws new shapes on top of the existing canvas content.

source-in	The new shape is drawn only where both the new shape and the destination canvas overlap. Everything else is made transparent.
source-out	The new shape is drawn where it doesn't overlap the existing canvas content.
source-atop	The new shape is only drawn where it overlaps the existing canvas content.
lighter	Where both shapes overlap the color is determined by adding color values.
xor	Shapes are made transparent where both overlap and drawn normal everywhere else.
destination-over	New shapes are drawn behind the existing canvas content.
destination-in	The existing canvas content is kept where both the new shape and existing canvas content overlap. Everything else is made transparent.
destination-out	The existing content is kept where it doesn't overlap the new shape.
destination-atop	The existing canvas is only kept where it overlaps the new shape. The new shape is drawn behind the canvas content.
darker	Where both shapes overlap the color is determined by subtracting color values.

## Example:

Following is a simple example which makes use of globalCompositeOperation attribute to create all possible compositions:

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

var compositeTypes = [
    'source-over', 'source-in', 'source-out', 'source-atop',
    'destination-over', 'destination-in', 'destination-out',
    'destination-atop', 'lighter', 'darker', 'copy', 'xor'
];

function drawShape(){
    for (i=0;i<compositeTypes.length;i++){
        var label = document.createTextNode(compositeTypes[i]);
        document.getElementById('lab'+i).appendChild(label);
        var ctx = document.getElementById('tut'+i).getContext('2d');
```

```

        // draw rectangle
        ctx.fillStyle = "#FF3366";
        ctx.fillRect(15,15,70,70);

        // set composite property
        ctx.globalCompositeOperation = compositeTypes[i];

        // draw circle
        ctx.fillStyle = "#0066FF";
        ctx.beginPath();
        ctx.arc(75,75,35,0,Math.PI*2,true);
        ctx.fill();
    }
}
</script>
</head>
<body onload="drawShape();">
<table border="1" align="center">
<tr>
    <td><canvas id="tut0" width="125" height="125"></canvas><br/>
        <label id="lab0"></label>
    </td>
    <td><canvas id="tut1" width="125" height="125"></canvas><br/>
        <label id="lab1"></label>
    </td>
    <td><canvas id="tut2" width="125" height="125"></canvas><br/>
        <label id="lab2"></label>
    </td>
</tr>
<tr>
    <td><canvas id="tut3" width="125" height="125"></canvas><br/>
        <label id="lab3"></label>
    </td>
    <td><canvas id="tut4" width="125" height="125"></canvas><br/>
        <label id="lab4"></label>
    </td>
    <td><canvas id="tut5" width="125" height="125"></canvas><br/>
        <label id="lab5"></label>
    </td>
</tr>
<tr>
    <td><canvas id="tut6" width="125" height="125"></canvas><br/>
        <label id="lab6"></label>
    </td>
    <td><canvas id="tut7" width="125" height="125"></canvas><br/>
        <label id="lab7"></label>
    </td>
    <td><canvas id="tut8" width="125" height="125"></canvas><br/>
        <label id="lab8"></label>
    </td>
</tr>
<tr>
    <td>
    </td>
    <td><canvas id="tut9" width="125" height="125"></canvas><br/>
        <label id="lab9"></label>
    </td>
    <td><canvas id="tut10" width="125" height="125"></canvas><br/>

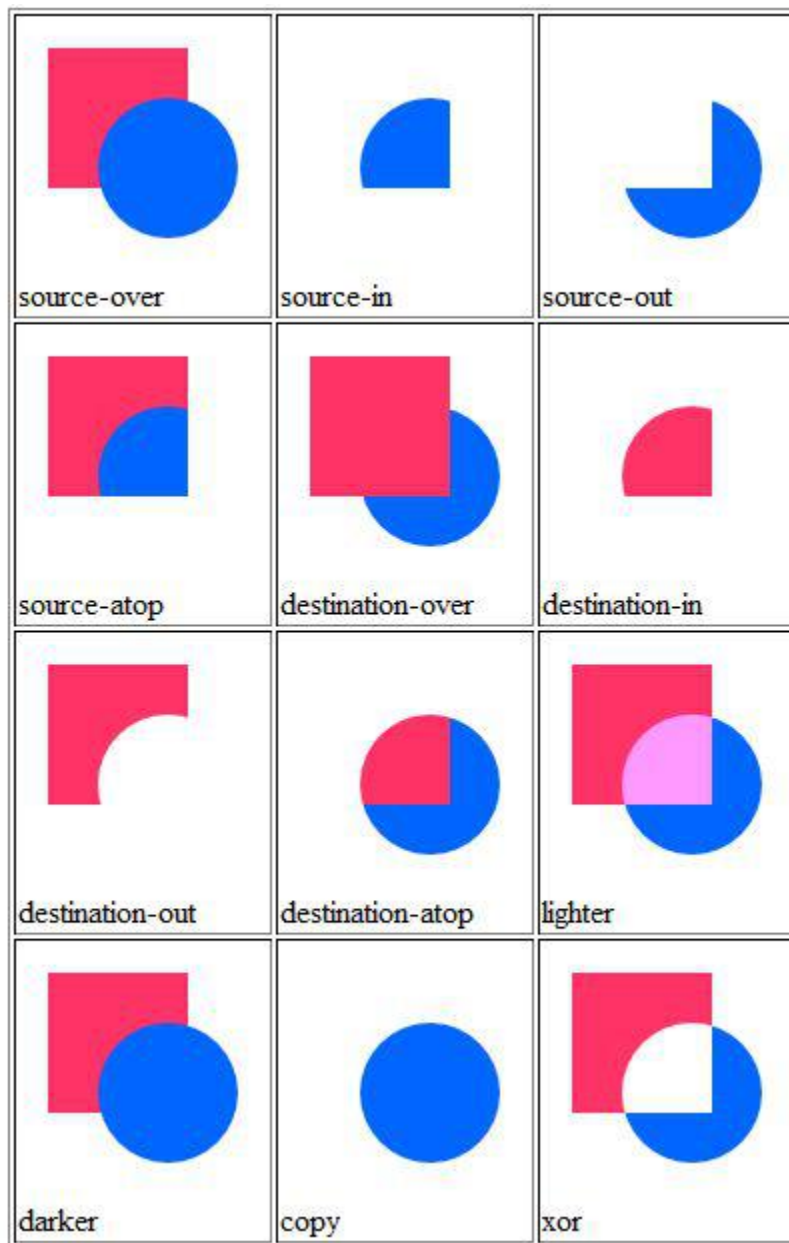
```

```

        <label id="lab10"></label>
    </td>
    <td><canvas id="tut11" width="125" height="125"></canvas><br/>
        <label id="lab11"></label>
    </td>
</tr>
</table>
</body>
</html>

```

The above example would produce following result:





# HTML5 Canvas - Animations

HTML5 canvas provides necessary methods to draw an image and erase it completely. We can take Javascript help to simulate good animation over a HTML5 canvas.

Following are the two important Javascript methods which would be used to animate an image on a canvas:

SN	Method and Description
1	<b>setInterval(callback, time);</b> This method repeatedly executes the supplied code after a given <i>time</i> milliseconds.
2	<b>setTimeout(callback, time);</b> This method executes the supplied code only once after a given <i>time</i> milliseconds.

## Example:

Following is a simple example which would rotate a small image repeatedly:

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

var pattern= new Image();

function animate(){
    pattern.src = '/images/pattern.jpg';
    setInterval(drawShape, 100);
}

function drawShape(){
    // get the canvas element using the DOM
    var canvas = document.getElementById('mycanvas');

    // Make sure we don't execute when canvas isn't supported
    if (canvas.getContext){

        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');

        ctx.fillStyle = 'rgba(0,0,0,0.4)';
        ctx.strokeStyle = 'rgba(0,153,255,0.4)';
        ctx.save();
        ctx.translate(150,150);

        var time = new Date();
        ctx.rotate( ((2*Math.PI)/6)*time.getSeconds() +
```

```

        ( (2*Math.PI)/6000)*time.getMilliseconds() );
ctx.translate(0,28.5);
ctx.drawImage(pattern,-3.5,-3.5);
ctx.restore();

} else {
    alert('You need Safari or Firefox 1.5+ to see this demo.');
```

```

}
```

```

</script>
</head>
```

```

<body onload="animate();">
```

```

    <canvas id="mycanvas" width="400" height="400"></canvas>
```

```

</body>
```

```

</html>
```

Assuming we have following pattern *images/pattern.jpg*.



The above example would produce following result:

