HTML - Canvas

HTML 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 HTML 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");
```

Example

Let us see a sample example that shows how to use <canvas> element in HTML document.

</> Open Compiler

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

Instance

Following is the code to get required context along with a check if the 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 HTML 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]-->
```

HTML Canvas - Drawing Rectangles

There are three methods that draw rectangles on the canvas –

S.No.	Method & Description
1	**fillRect(x,y,width,height)** This method draws a filled rectangle.
2	**strokeRect(x,y,width,height)** This method draws a rectangular outline.
3	**clearRect(x,y,width,height)** 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.

</> Open Compiler

```html
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>

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

HTML Canvas - Drawing Paths

We require the following methods to draw paths on the canvas –

S.No.	Method & Description

1	**beginPath()** This method resets the current path.
2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** 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	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**arc(x, y, radius, startAngle, endAngle, anticlockwise)** 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.

Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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.');
         }
      }
   </script>
</head>
<body id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Drawing Lines

Line Methods

We require the following methods to draw lines on the canvas –

S.No.	Method & Description
1	**beginPath()** This method resets the current path.
2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** 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	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**lineTo(x, y)**

	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 the above-mentioned methods to draw a triangle.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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 id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Drawing Bezier Curves

We need the following methods to draw Bezier curves on the canvas −

S.No.	Method & Description
1	**beginPath()** This method resets the current path.
2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** 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	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)** 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.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
```

```
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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.');
         }
      }
   </script>
</head>
<body id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Drawing Quadratic Curves

We require the following methods to draw quadratic curves on the canvas –

S.No.	Method & Description
1	**beginPath()** This method resets the current path.

2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** 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	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**quadraticCurveTo(cpx, cpy, x, y)** 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. The **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 curve.

Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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 id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Using Images

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

S.No.	Method & Description
1	**beginPath()** This method resets the current path.
2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** 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	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**drawImage(image, dx, dy)**

	This method draws the given image onto the canvas. Here image 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 image.

</> Open Compiler

```
<!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 = '/html/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.');
         }
      }
   </script>
</head>
```

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

HTML Canvas - Create Gradients

HTML canvas allows us to fill and stroke shapes using linear and radial gradients using the following methods −

S.No.	Method & Description
1	**addColorStop(offset, color)** 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	**createLinearGradient(x0, y0, x1, y1)** 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	**createRadialGradient(x0, y0, r0, x1, y1, r1)** 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.

Example - Linear Gradient

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

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width:100px;
         height:100px;
         margin:0px auto;
      }
   </style>
   <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.');
         }
      }
   </script>
</head>
<body id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

Example - Radial Gradient

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

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height: 100px;
         margin: 0px auto;
      }
   </style>
   <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 id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Styles and Colors

HTML canvas provides the following two important properties to apply colors to a shape –

S.No.	Method & Description
1	**fillStyle** This attribute represents the color or style to use inside the shapes.
2	**strokeStyle** 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.

Example - fillStyle

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

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>

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

Example - strokeStyle

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

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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.');
         }
      }
   </script>
</head>
<body id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Text and Fonts

HTML canvas provides capabilities to create text using different font and text properties listed below −

S.No.	Property & Description
1	**font [= value]** This property returns the current font settings and can be set, to change the font.
2	**textAlign [= value]** This property returns the current text alignment settings and can be set, to change the alignment. The possible values are start, end, left, right, andcenter.
3	**textBaseline [= value]** This property returns the current baseline alignment settings and can be set, to change the baseline alignment. The possible values are top, hanging, middle , alphabetic, ideographic and bottom
4	**fillText(text, x, y [, maxWidth])** This property fills the given text at the given position indicated by the given coordinates x and y.
5	**strokeText(text, x, y [, maxWidth])** This property strokes the given text 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 −

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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 id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Pattern and Shadow

Create Pattern

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

S.No.	Method & Description
1	**createPattern(image, repetition)** This method will use image to create the pattern. The second argument could be a string with one of the following values: repeat, repeat-x, repeat-y, and no-repeat. 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.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
```

```
    <style>
      #test {
        width:100px;
        height:100px;
        margin: 0px auto;
      }
    </style>
    <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 = '/html/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 id="test" onload="drawShape();">
    <canvas id="mycanvas"></canvas>
  </body>
</html>
```

Assuming we have following pattern –

Create Shadows

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

S.No.	Property & Description
1	**shadowColor [= value]** This property returns the current shadow color and can be set, to change the shadow color.
2	**shadowOffsetX [= value]** This property returns the current shadow offset X and can be set, to change the shadow offset X.
3	**shadowOffsetY [= value]** This property returns the current shadow offset Y and can be set, change the shadow offset Y.
4	**shadowBlur [= value]** 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.

Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>

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

HTML Canvas - Save and Restore States

HTML 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 −

- The transformations such as translate, rotate and scale etc.
- The current clipping region.
- 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.

S.No.	Method & Description
1	**save()** This method pushes the current state onto the stack..
2	**restore()** 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.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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 id="test" onload="drawShape();">
   <canvas id="mycanvas"></canvas>
</body>
</html>
```

HTML Canvas - Translation

HTML 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 –

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width:100px;
         height:100px;
         margin:0px auto;
      }
   </style>
   <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 id="test" onload="drawShape();">
   <canvas id="mycanvas" width="400" height="400"></canvas>
```

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

HTML Canvas - Rotation

HTML 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.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <style>
      #test {
         width: 100px;
         height:100px;
         margin: 0px auto;
      }
   </style>
   <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.');
         }
      }
   </script>
</head>
<body id="test" onload="drawShape();">
   <canvas id="mycanvas" width="400" height="400"></canvas>
</body>
</html>
```

HTML Canvas - Scaling

HTML 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.

</> Open Compiler

```
<!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,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>
```

HTML Canvas - Transforms

HTML 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.

S.No.	Method & Description
1	**transform(m11, m12, m21, m22, dx, dy)** This method changes the transformation matrix to apply the matrix given by the arguments.
2	**setTransform(m11, m12, m21, m22, dx, dy)** 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 –

</> Open Compiler

```
<!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>
```

HTML Canvas - Composition

HTML 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 –

S.No.	Attribute & Description
1	**source-over** This is the default setting and draws new shapes on top of the existing canvas content.
2	**source-in** The new shape is drawn only where both the new shape and the destination canvas overlap. Everything else is made transparent.
3	**source-out** The new shape is drawn where it doesn't overlap the existing canvas content.
4	**source-atop** The new shape is only drawn where it overlaps the existing canvas content.
5	**lighter** Where both shapes overlap the color is determined by adding color values.
6	**xor** Shapes are made transparent where both overlap and drawn normal everywhere else.
7	**destination-over** New shapes are drawn behind the existing canvas content.
8	**destination-in**

	The existing canvas content is kept where both the new shape and existing canvas content overlap. Everything else is made transparent.
9	**destination-out** The existing content is kept where it doesn't overlap the new shape.
10	**destination-atop** The existing canvas is only kept where it overlaps the new shape. The new shape is drawn behind the canvas content.
11	**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 −

Open Compiler

```
<!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><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>
```

HTML Canvas - Animations

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

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

S.No.	Method & Description
1	**setInterval(callback, time);** This method repeatedly executes the supplied code after a given timemilliseconds.
2	**setTimeout(callback, time);** This method executes the supplied code only once after a given time milliseconds.

Example

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

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <script type="text/javascript">
      var pattern= new Image();
      function animate(){
         pattern.src = '/html/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>
```