# UNIT-5

# CANVAS-STYLES AND COLORS

## HTML5 Canvas-Text and Fonts:

In HTML5, canvas element supports basic text rendering on a line-by-line basis. There are two methods fillText() and strokeText() to draw text on canvas. You can use font property (type : string) to specify a number of text setting such as style, weight, size, and font. The style can be normal, italic, or bold. Default style is normal.

Code example of font property :

```
ctx.font = 'italic 400 12px, sans-serif';
```

The "400" font-weight doesn't appear because that is the default value.

### a. fillText() Method

The fillText() method is used to render filled text to the canvas by using the current fill style and font.

This method draws directly to the canvas without altering the existing path, so any following fill() or stroke() calls will have no impact on it.

The text is rendered using the font and text layout design as defined by the font, textAlign, textBaseline, and direction properties.

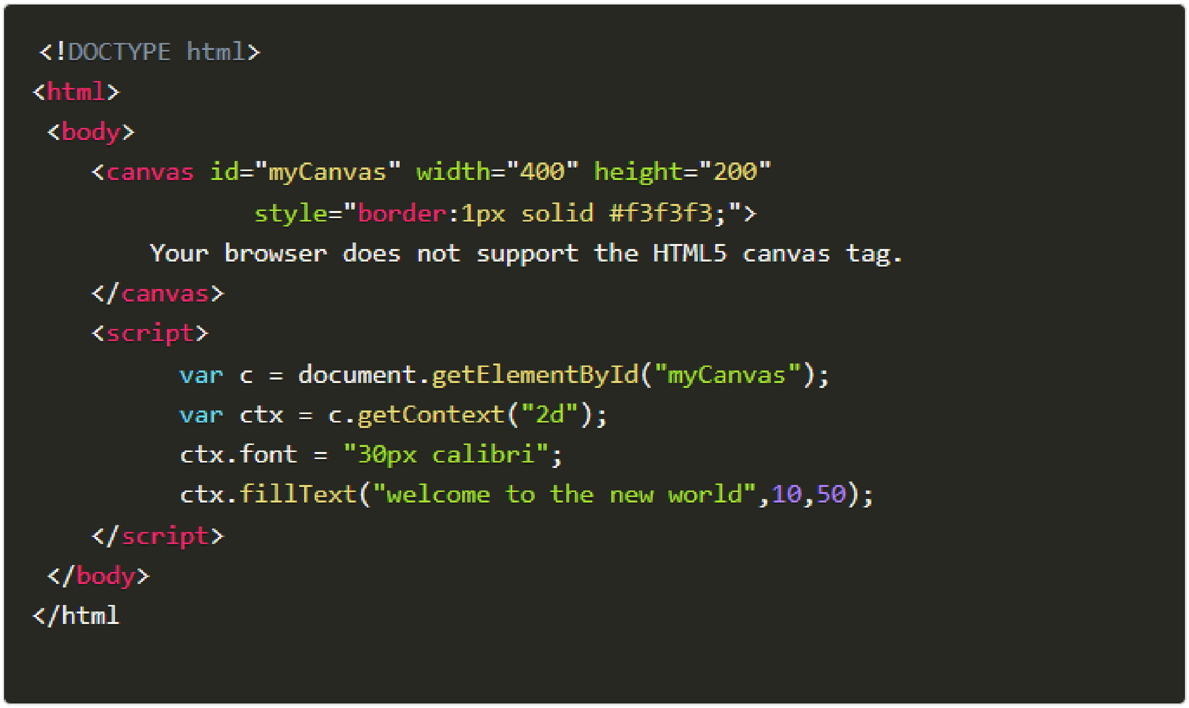
**Syntax:**

```
ctx.fillText(text, x, y, maxWidth)
```

| **Parameters** | **Type** | **Description** |
|---|---|---|
| text | string | The text characters to paint on the canvas. |
| x | number | The horizontal coordinate to start painting the text, relative to the canvas. |
| y | number | The vertical coordinate to start painting the text, relative to the canvas. |
| maxWidth | number | The maximum possible text width. |

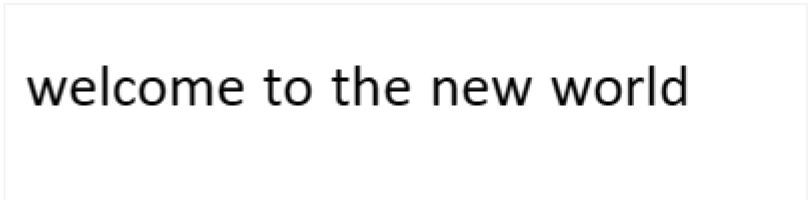
## Example -

The following example describes the fillText method.



```
 <!DOCTYPE html>
<html>
 <body>
    <canvas id="myCanvas" width="400" height="200"
               style="border:1px solid #f3f3f3;">
        Your browser does not support the HTML5 canvas tag.
    </canvas>
    <script>
          var c = document.getElementById("myCanvas");
          var ctx = c.getContext("2d");
          ctx.font = "30px calibri";
          ctx.fillText("welcome to the new world",10,50);
    </script>
 </body>
</html
```

Output:-


```
welcome to the new world
```

## b. strokeText() Method

The strokeText() method is used to render the specified text at the specified position by using the current font, lineWidth, and strokeStyle property.

This method draws directly to the canvas without altering the existing path, so any following fill() or stroke() calls will have no impact on it.

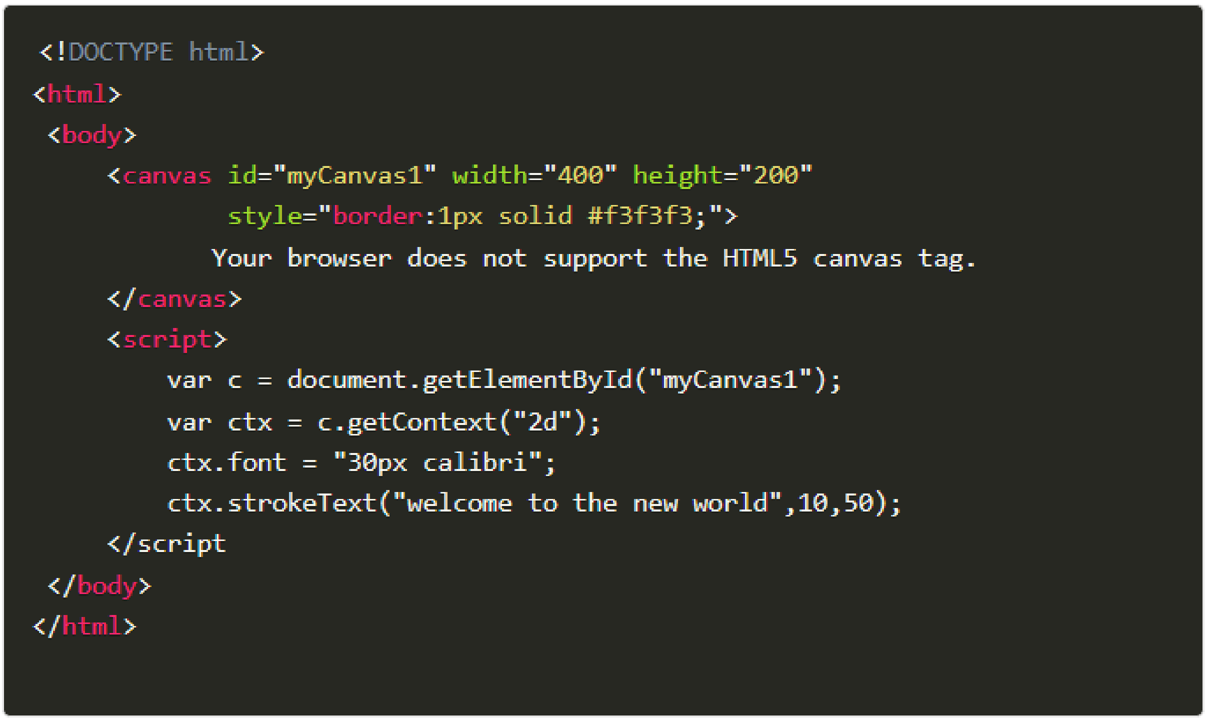
**Syntax:**

```
ctx.strokeText(text, x, y, maxWidth);
```

| **Parameters** | **Type** | **Description** |
|---|---|---|
| text | string | The text characters to paint on the canvas. |
| x | number | The horizontal coordinate to start painting the text, relative to the canvas. |
| y | number | The vertical coordinate to start painting the text, relative to the canvas. |
| maxWidth | number | The maximum possible text width. |

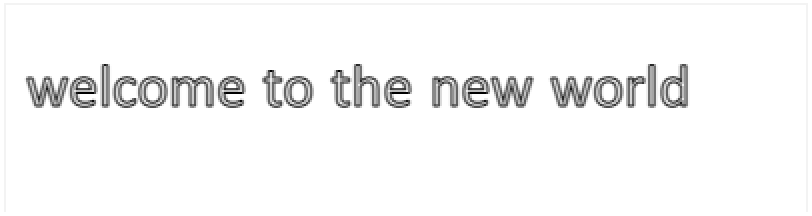
## Example -

The following example describes about the strokeText() method.



```
 <!DOCTYPE html>
<html>
 <body>
     <canvas id="myCanvas1" width="400" height="200"
             style="border:1px solid #f3f3f3;">
            Your browser does not support the HTML5 canvas tag.
     </canvas>
     <script>
         var c = document.getElementById("myCanvas1");
         var ctx = c.getContext("2d");
         ctx.font = "30px calibri";
         ctx.strokeText("welcome to the new world",10,50);
     </script
 </body>
</html>
```

## Output-



## c. Text Alignment

In HTML5 canvas textAlign property is used to set the text alignment on the canvas. Possible values are start, end, left, right, and center. Default value : start.

- start : The text is aligned at the normal start of the line (left-aligned for left-to-right locales, right-aligned for right-to-left locales).

- end : The text is aligned at the normal end of the line (right-aligned for left-to-right locales, left-aligned for right-to-left locales).
- left : The text is left-aligned.
- right : The text is right-aligned.
- center : The text is centered.

**Syntax :**

ctx.textAlign = value

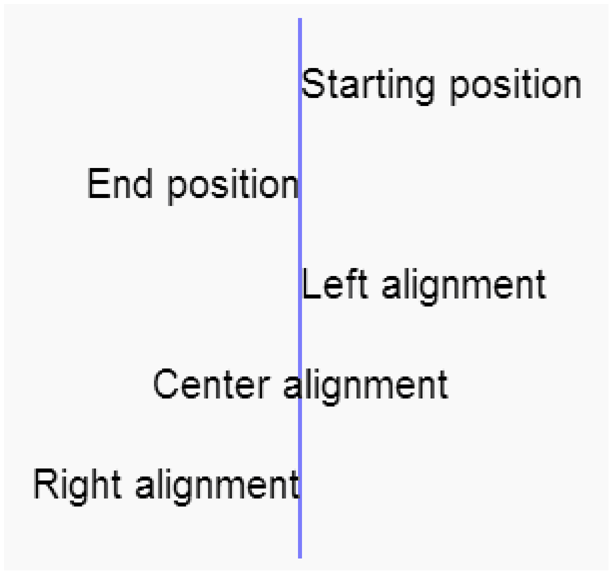
**Example: HTML5 Canvas, draw text using text alignment**

The following code example shows all the property values of textAlign property.

**Code:**

```
<!DOCTYPE html>

<html>

<head>

<title>HTML5 Canvas - Text</title>

</head>

<body>

<canvas id="DemoCanvas" width="500" height="600"></canvas>

<script>

var canvas = document.getElementById("DemoCanvas");

if (canvas.getContext)

{

var ctx = canvas.getContext('2d');

// Create a line at the anchor point.
ctx.strokeStyle = "blue";
ctx.textAlign = "center";
ctx.moveTo(150, 60);
ctx.lineTo(150, 330);
ctx.stroke();

ctx.strokeStyle = "green";
ctx.font = "20px sans-serif";
ctx.textAlign = "start";
ctx.fillText("Starting position", 150, 100);
ctx.textAlign = "end";
ctx.fillText("End position", 150, 150);
ctx.textAlign = "left";
ctx.fillText("Left alignment", 150, 200);
ctx.textAlign = "center";
ctx.fillText("Center alignment", 150, 250);
ctx.textAlign = "right";
ctx.fillText("Right alignment", 150, 300);
}
</script>
</body>
</html>
```

**Output :**



## d. Text Baseline

In HTML5 canvas textBaseline property is used to get or set the current settings for the font baseline alignment. Legal values are top, hanging, middle, alphabetic, ideographic, bottom.

- top : The top of the em square.
- hanging : The hanging baseline
- middle : The middle of the em square.
- alphabetic : Default. The alphabetic baseline.
- ideographic : The ideographic baseline.
- bottom : The bottom of the em square.

**Syntax:**

ctx.textBaseline = value

## e. Text Metrics

In HTML5 canvas measureText() method is used to get the text metrics of HTML5 Canvas text. The method returns an object that contains the width (in pixels) of the specified text.

**Syntax:**

```
var textWidth = ctx.measureText(text)
```

## HTML5 Canvas-Pattern and Shadows:

## Pattern:

createPattern() method used to create pattern by using a picture on the canvas in HTML5. It returns a pattern object, places the fillStyle property to the pattern object, and then fill the shape using fill().

The createPattern() technique requires an image object and a repeat alternative(that are repeat-x, repeat-y, or no-repeat, repeat.)

### Syntax -

```
createPattern(image, repetition)
```

This method will use picture to build 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 blank string or null is stipulated, repeat will be applied as a default.

| Parameter | Description |
|---|---|
| image | Stipulates the image, canvas, or video element of the pattern to utilize |
| repeat | Default. The pattern repeats both horizontally and vertically |
| repeat-x | The pattern repeats only horizontally |
| repeat-y | The pattern repeats only vertically |
| no-repeat | The pattern will be exhibited only once (no repeat) |

## Example:

## Shadows

In HTML5 canvas, you can add shadows on a shape, a line, text, or an image which can create a sense of third dimension. To add shadows with the HTML5 Canvas, you can use the following properties of the canvas context.

1. shadowOffsetX
2. shadowOffsetY
3. shadowColor
4. shadowBlur

### 1. shadowOffsetX() Property

The property is used to get or set the horizontal distance of a shadow from a shape. You can use positive or negative values to control the position of a shadow. The default value is zero.

**Syntax:**

```
ctx.shadowOffsetX = h_distance;
```

Where h_distance (type: number) is the horizontal distance of a shadow from a shape.

### 2. shadowOffsetY() Property

The property is used to get or set the vertical distance of a shadow from a shape. You can use positive or negative values to control the position of a shadow. The default value is zero.

**Syntax:**

```
ctx.shadowOffsetX = v_distance;
```

Where v_distance (type: number) is the vertical distance of a shadow from a shape.

## 3. shadowColor() Property

The property is used to get or set the color to use for shadows.

**Syntax:**

```
ctx.shadowColor
```

Where shadowColor (type: string) is the CSS color.

## 4. shadowBlur() Property

The property is used to get or set the current level of blur that is applied to shadows.

**Syntax:**

```
ctx.shadowBlur = blur_value
```

Where blur_value is the amount of blur (type: number) that is applied to shadows.

**Example : HTML5 Canvas adding shadow on Text**

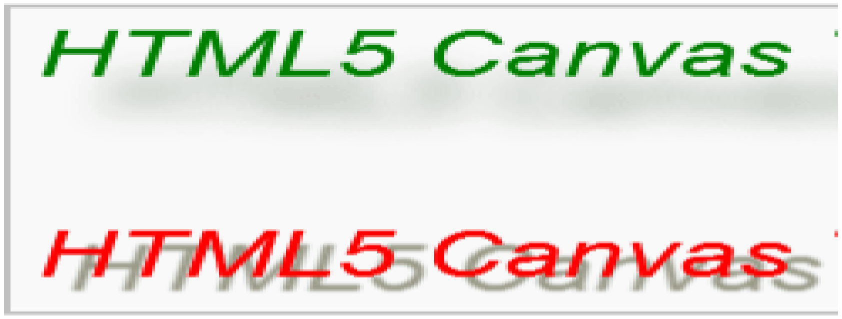
The following web document creates shadow on text.

**Code :**

```
<!DOCTYPE html>

<html>

<head><title>HTML5 Canvas - shadow</title>

</head>

<body>

<canvas id="DemoCanvas" width="500" height="600"></canvas>

<script>

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

```
if (canvas.getContext)

{

 var ctx = canvas.getContext('2d');

 ctx.fillStyle = 'green';

 ctx.shadowColor = '#898';

 ctx.shadowBlur = 20;

 ctx.shadowOffsetX = 20;

 ctx.shadowOffsetY = 20;

 //ctx.fill();

 ctx.font = 'italic 32px sans-serif';

 ctx.fillText('HTML5 Canvas ', 10, 50);


 ctx.fillStyle = 'red';

 ctx.shadowColor = '#998';

 ctx.shadowBlur = 1;

 ctx.shadowOffsetX = 9;

 ctx.shadowOffsetY = 7;

 //ctx.fill();

 ctx.font = 'italic 32px sans-serif';

 ctx.fillText('HTML5 Canvas ', 10, 150);

}

</script>

</body>

</html>
```

**Output:**



## HTML5 Canvas-States(save & restore):

The canvas using 2D Context when drawing on HTML5, the 2D Context is in a certain state. We can set that certain state by using the 2D Context properties like **fillStyle** and **strokeStyle**. All these modifications in total is called the 2D context state.

Sometimes, we need to change the state of the 2D Context when drawing on a canvas based on the requirement. For **example**, we may need one strokStyle for one line and another strokeStyle for other lines. or something else.

The canvas drawing state is essentially a snapshot of entire styles and changes that consists of the followings -

- The changes such as rotate, translate, and scale etc.
- The existing clipping region.
- The present values of the following attributes – **globalAlpha, fillStyle, strokeStyle, lineCap, lineWidth, lineJoin, miterLimit, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, globalCompositeOperation, font, textAlign, textBaseline.**

We can use the **save()** and **restore()** methods of the canvas context to save and restore different transformation states with the HTML5 Canvas.

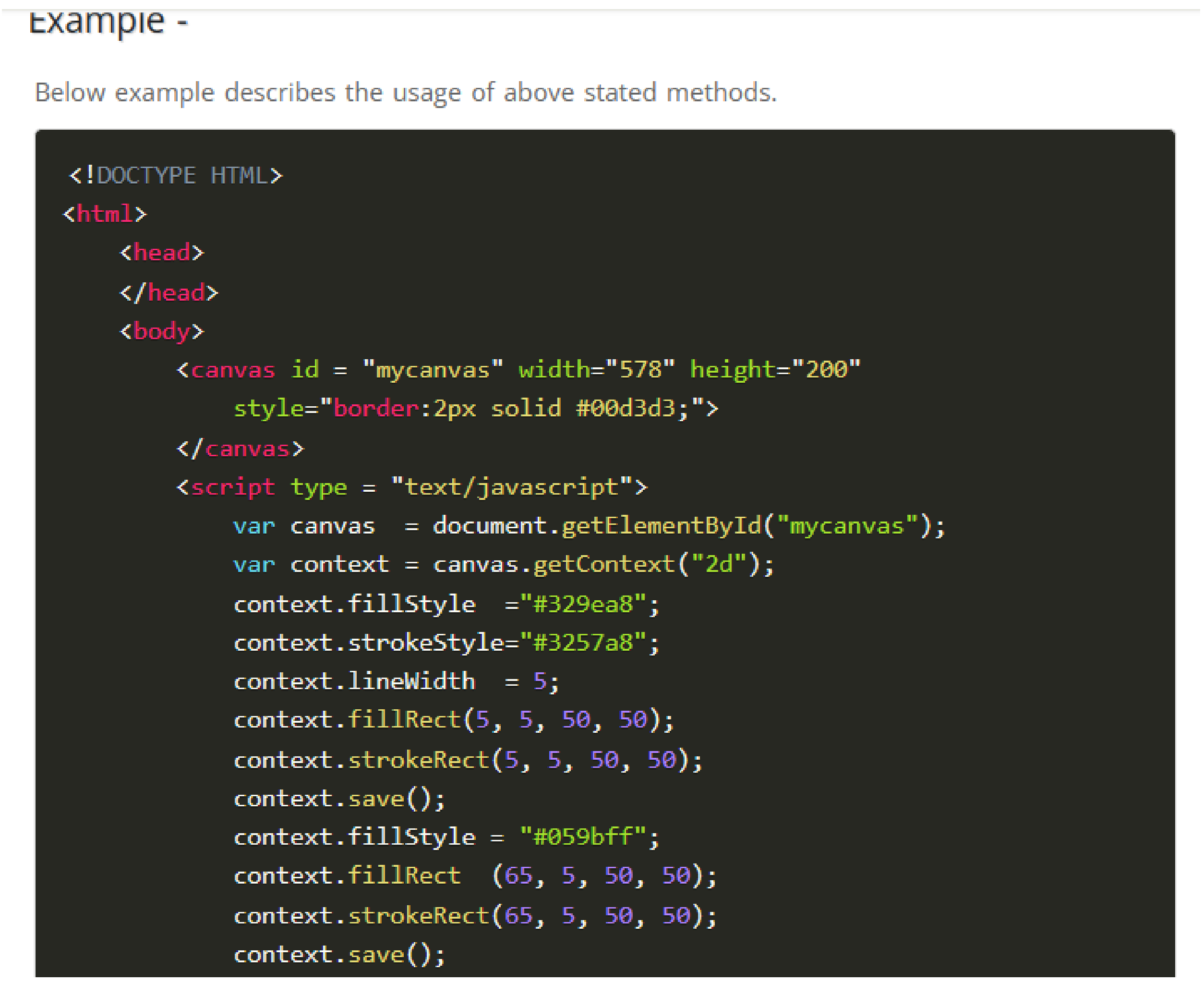
Before we look at the transformation methods, I'll introduce two other methods which are indispensable once you start generating ever more complex drawings.

save()
restore()

The canvas save and restore methods are used to save and retrieve the canvas state. The canvas drawing state is basically a snapshot of all the styles and transformations that have been applied. Both methods take no parameters.

Canvas states are stored on a stack. Every time the save method is called, the current drawing state is pushed onto the stack.

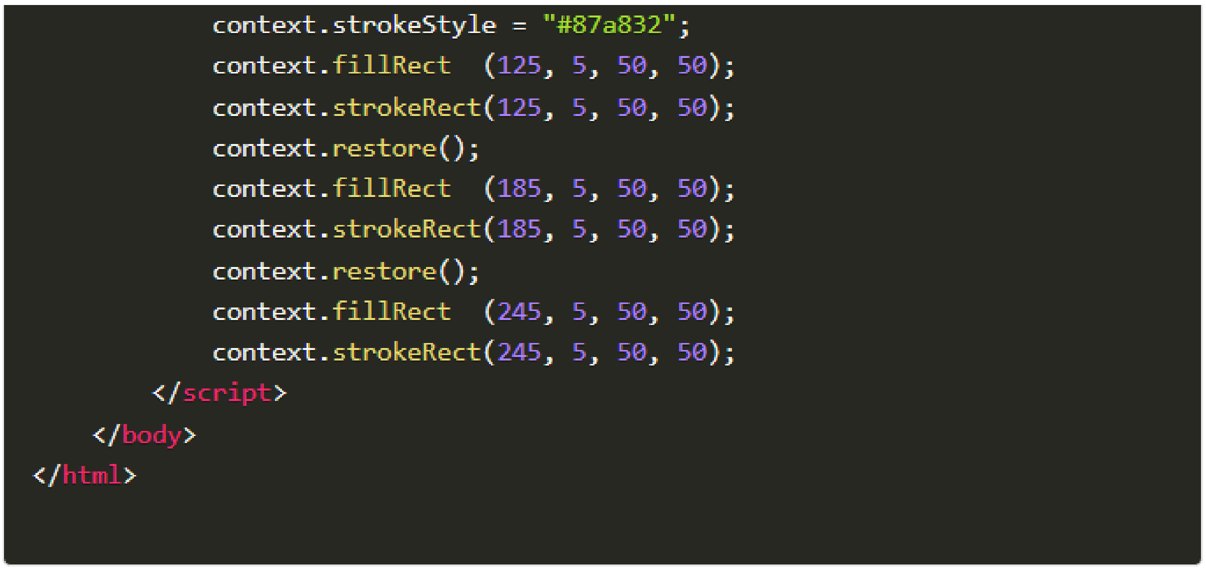
Every time the restore method is called, the last saved state is returned from the stack and all saved settings are restored.



Example -

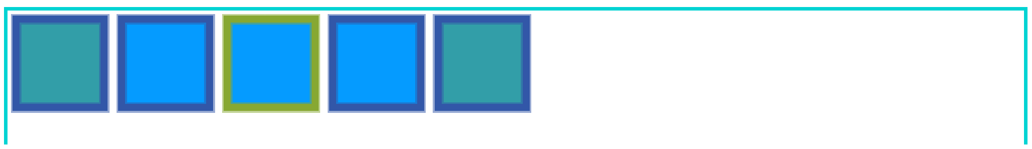
Below example describes the usage of above stated methods.

```
 <!DOCTYPE HTML>
<html>
    <head>
    </head>
    <body>
        <canvas id = "mycanvas" width="578" height="200"
            style="border:2px solid #00d3d3;">
        </canvas>
        <script type = "text/javascript">
            var canvas  = document.getElementById("mycanvas");
            var context = canvas.getContext("2d");
            context.fillStyle  ="#329ea8";
            context.strokeStyle="#3257a8";
            context.lineWidth  = 5;
            context.fillRect(5, 5, 50, 50);
            context.strokeRect(5, 5, 50, 50);
            context.save();
            context.fillStyle = "#059bff";
            context.fillRect  (65, 5, 50, 50);
            context.strokeRect(65, 5, 50, 50);
            context.save();
```



```
            context.strokeStyle = "#87a832";
            context.fillRect  (125, 5, 50, 50);
            context.strokeRect(125, 5, 50, 50);
            context.restore();
            context.fillRect  (185, 5, 50, 50);
            context.strokeRect(185, 5, 50, 50);
            context.restore();
            context.fillRect  (245, 5, 50, 50);
            context.strokeRect(245, 5, 50, 50);
        </script>
    </body>
</html>
```

Output -



## Introduction to affine transformation:

- An ***affine transform*** is composed of zero or more linear transformations (rotation, scaling, or shear) and translation (shift).
- Several linear transformations can be combined into a single matrix
- **A rotation** is a transformation that moves a rigid body around a fixed point.
- **A scaling** is a transformation that enlarges or diminishes objects. The scale factor is the same in all directions.
- **A translation** is a transformation that moves every point a constant distance in a specified direction.
- **A shear** is a transformation that moves an object perpendicular to a given axis, with greater value on one side of the axis than the other.
- There is a **transform()** method, which multiplies the current transformation with the matrix described by the arguments of the method.
- We are able to scale, rotate, move, and shear the context.
- There are also methods that perform specific transformations:

translate(), rotate(), and scale().

## HTML5 Canvas translation:

translate() transform method used to translate the HTML5 Canvas context. HTML5 canvas offers translate(x, y) process to move the canvas and its origin to a distinct point in the grid.

The translate() method remaps the (0,0) position on the canvas.

You can reposition the center (0,0) of your drawing surface by calling the translate(x,y) method. The origin of the canvas's coordinate system is moved to the point (x,y).

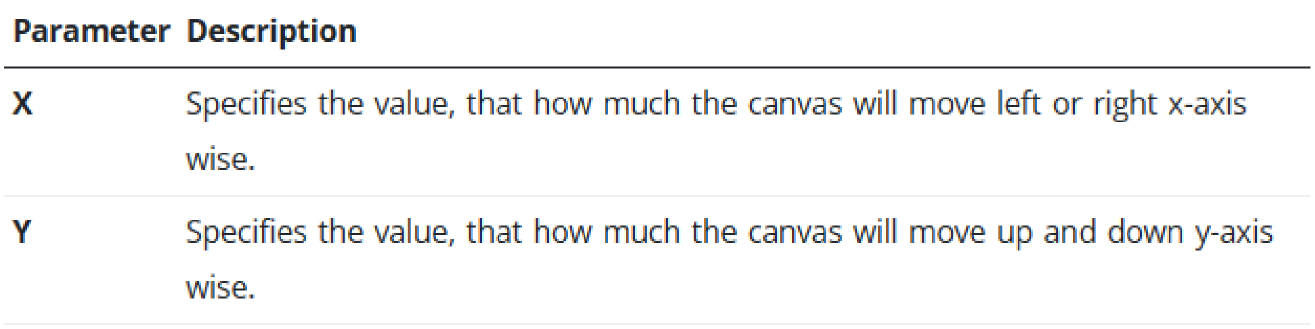
### translate(x,y) Method

The translate() method is used to move the origin point to a specified point in a canvas.

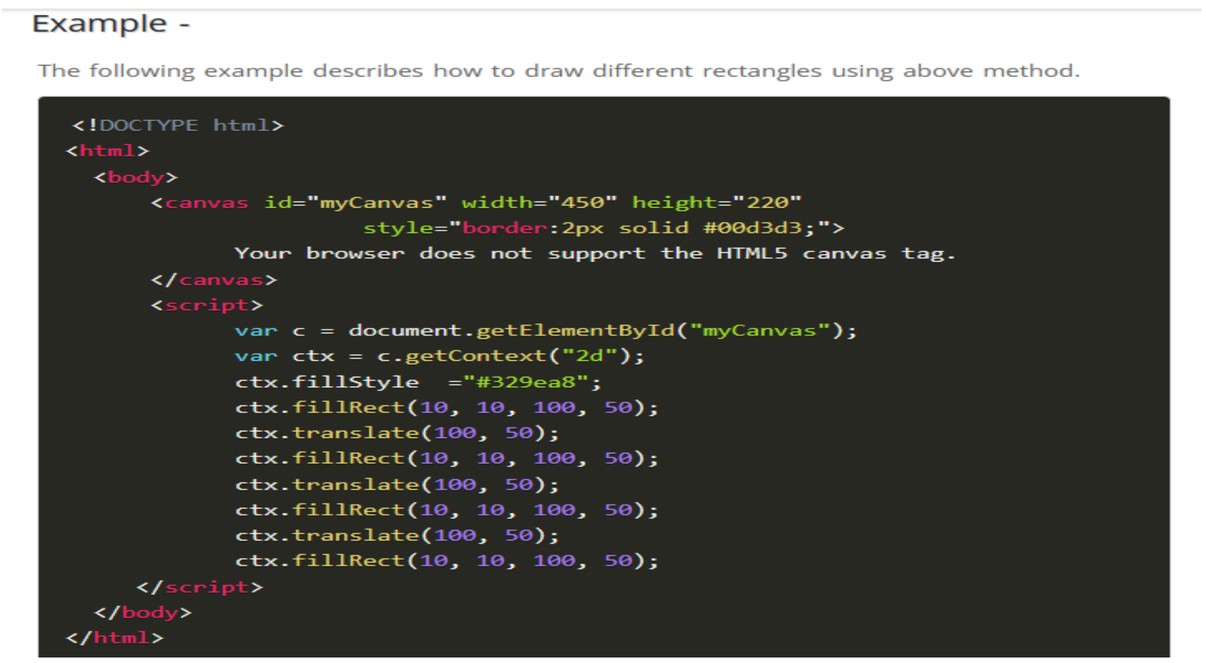
**Syntax :**

ctx.translate(x, y)

| Parameters | Type | Description |
|---|---|---|
| x | number | The value to add to horizontal (or x) coordinates. |
| y | number | The value to add to vertical (or y) coordinates. |



| Parameter | Description |
|---|---|
| X | Specifies the value, that how much the canvas will move left or right x-axis wise. |
| Y | Specifies the value, that how much the canvas will move up and down y-axis wise. |

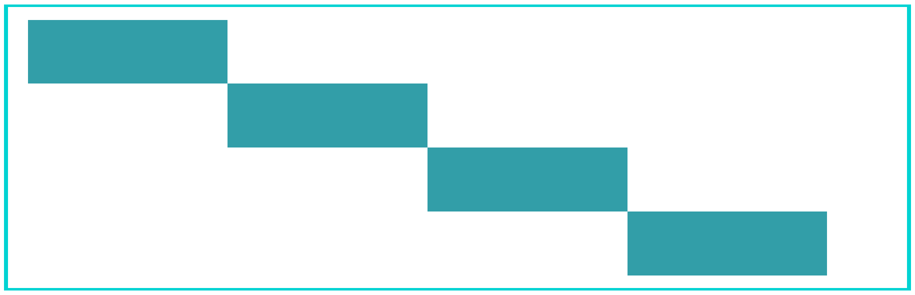


Example -

The following example describes how to draw different rectangles using above method.

```
 <!DOCTYPE html>
<html>
  <body>
      <canvas id="myCanvas" width="450" height="220"
                   style="border:2px solid #00d3d3;">
            Your browser does not support the HTML5 canvas tag.
      </canvas>
      <script>
            var c = document.getElementById("myCanvas");
            var ctx = c.getContext("2d");
            ctx.fillStyle  ="#329ea8";
            ctx.fillRect(10, 10, 100, 50);
            ctx.translate(100, 50);
            ctx.fillRect(10, 10, 100, 50);
            ctx.translate(100, 50);
            ctx.fillRect(10, 10, 100, 50);
            ctx.translate(100, 50);
            ctx.fillRect(10, 10, 100, 50);
     </script>
  </body>
</html>
```

Output -



## HTML5 Canvas Scaling:

Canvas scale used to scale the current drawing and draw enlarged shapes or scaled down shapes and bitmaps.

HTML5 canvas offers scale(x, y) method to decrease or increase the units in a canvas grid.This process brings two parameters(x, y) where **x** is the scale factor in the horizontal way and y is the scale factor in the vertical way.These two parameters should be positive numbers.

Values lesser than **1.0** reduce the unit size and values bigger than 1.0 expand the unit size.Establishing the scaling factor to exactly 1.0 doesn't affect the unit size( i.e. 1=100%, 0.5=50%, 2=200%, etc.).

### Syntax -

```
context.scale(scalewidth,scaleheight);
```

### scale(x,y) Method

The translate() method is used to scale the current context by the specified horizontal (x) and vertical (y) factors.

**Syntax :**

```
ctx.scale(x, y)
```

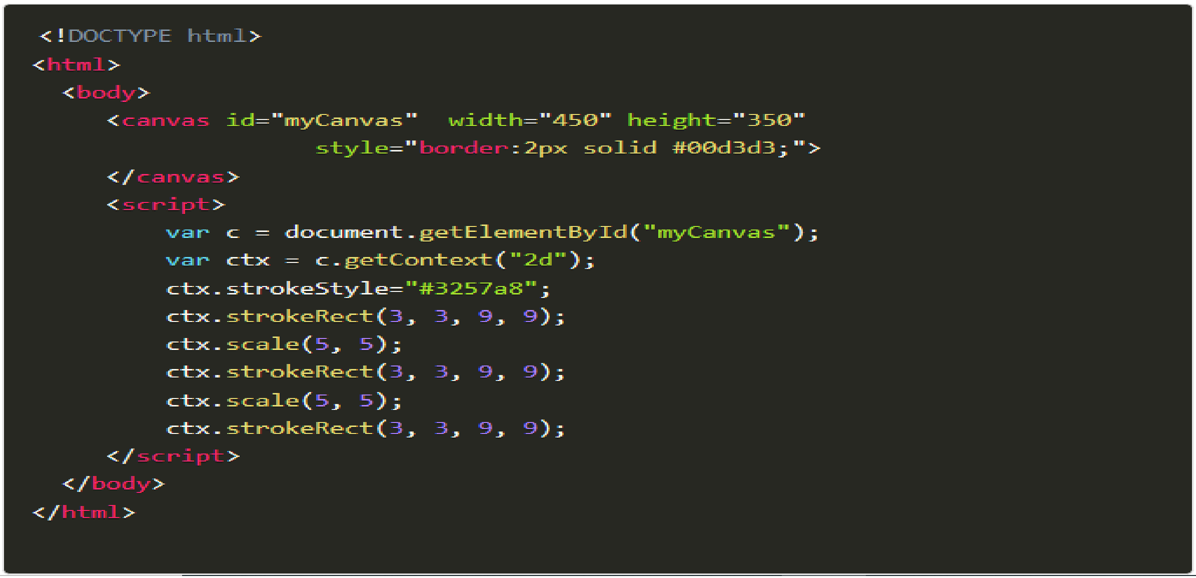
| Parameters | Type | Description |
|---|---|---|
| x | number | The horizontal scaling factor, where 1 equals unity or 100% scale. |
| y | number | The vertical scaling factor. |

| Parameter | Description |
|---|---|
| scalewidth | Scales the width of the existing drawing (1=100%, 0.5=50%, 2=200%, etc.) |
| scaleheight | Scales the height of the existing drawing (1=100%, 0.5=50%, 2=200%, etc.) |

**Example:**

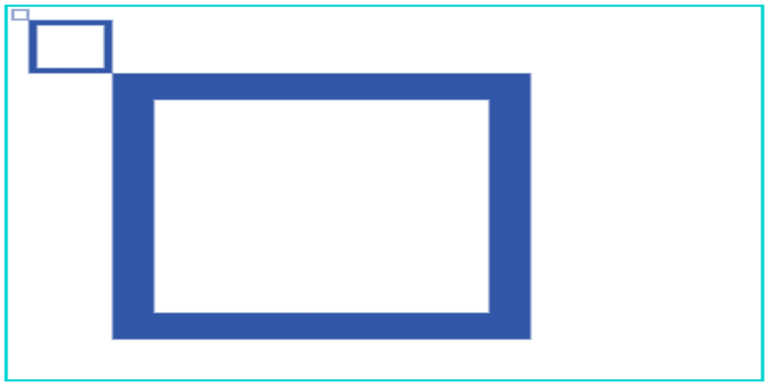
The following example to draw a square, scale to 500%, draw a square again, scale to 500%,draw square again.

-



```
 <!DOCTYPE html>
<html>
  <body>
     <canvas id="myCanvas"  width="450" height="350"
                  style="border:2px solid #00d3d3;">
     </canvas>
     <script>
         var c = document.getElementById("myCanvas");
         var ctx = c.getContext("2d");
         ctx.strokeStyle="#3257a8";
         ctx.strokeRect(3, 3, 9, 9);
         ctx.scale(5, 5);
         ctx.strokeRect(3, 3, 9, 9);
         ctx.scale(5, 5);
         ctx.strokeRect(3, 3, 9, 9);
     </script>
  </body>
</html>
```

Output -



## HTML5 Canvas Rotation:

The rotate() method rotates the present drawing. We can use the rotate() transform technique to rotate the drawing. HTML5 canvas offers rotate(angle) process to rotate the canvas around the existing origin.

This process takes one parameter **angle** and the canvas is rotated by **angle**. The rotation is a clockwise rotation calculated in radians. The rotate needs an angle in radians.

To define the rotation point, we need to provide the top left corner of the context lies on the desired rotation point.

**Note!** The rotation will only affect to the drawings after the rotation is completed.

### Syntax -

```
context.rotate(angle);
```

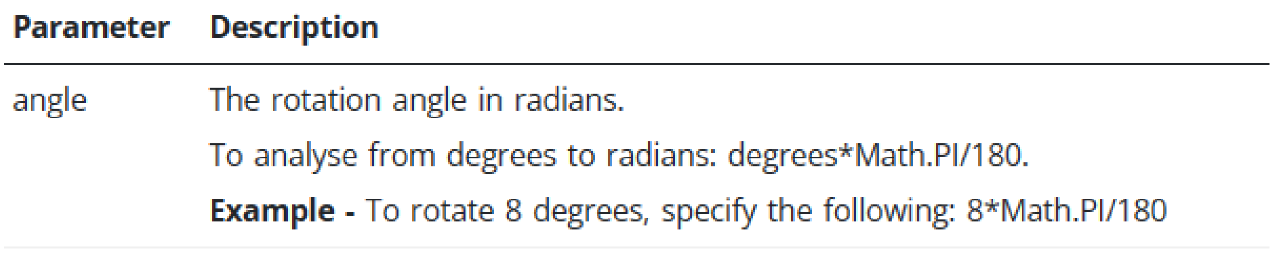
### rotate(angle) Method

The rotate() method is used to rotate the current context coordinates.

**Syntax :**

ctx.rotate(angle)

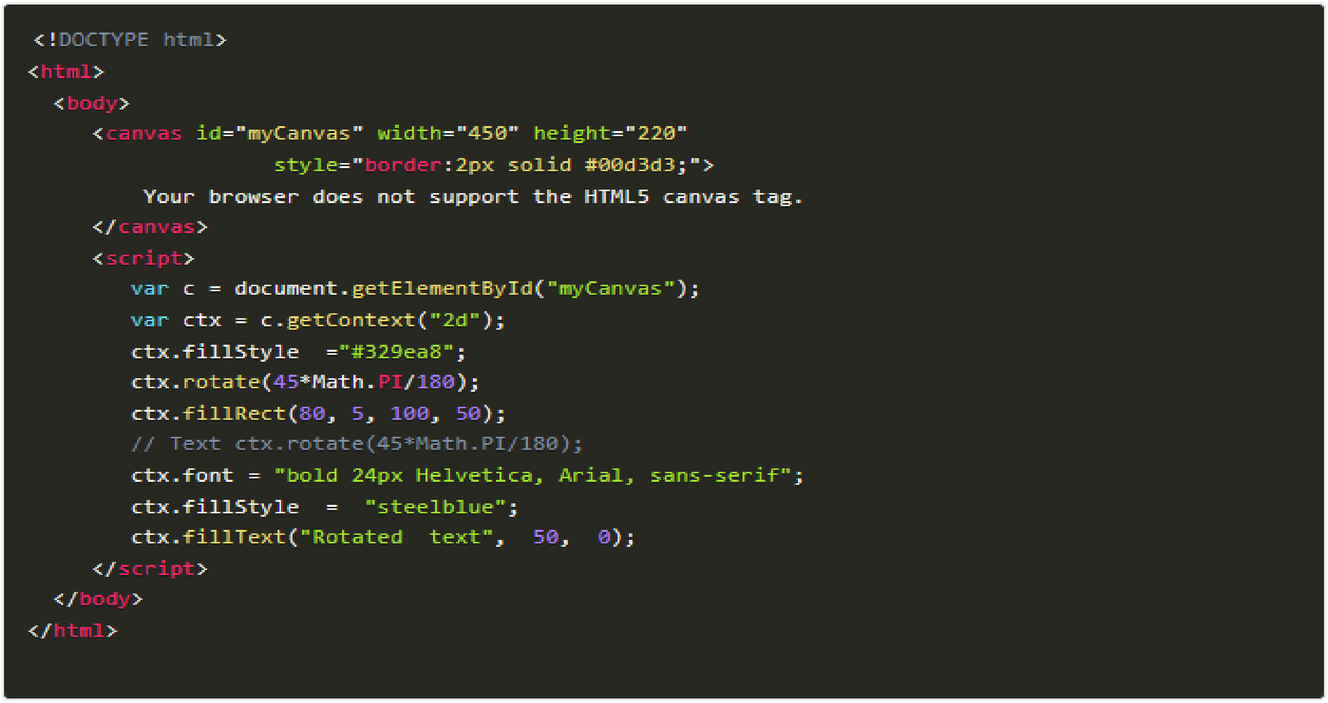
| Parameters | Type | Description |
|---|---|---|
| angle | number | The rotation angle, in radians. |



| Parameter | Description |
|---|---|
| angle | The rotation angle in radians.<br>To analyse from degrees to radians: degrees*Math.PI/180.<br>**Example -** To rotate 8 degrees, specify the following: 8*Math.PI/180 |

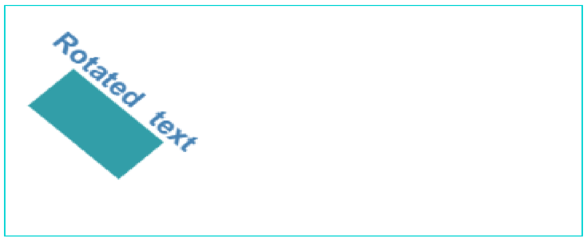
Example-

The following example describes the rotate the rectangular 45 degrees and 90 degrees rotated text.



```
 <!DOCTYPE html>
<html>
  <body>
     <canvas id="myCanvas" width="450" height="220"
                   style="border:2px solid #00d3d3;">
         Your browser does not support the HTML5 canvas tag.
     </canvas>
     <script>
        var c = document.getElementById("myCanvas");
        var ctx = c.getContext("2d");
        ctx.fillStyle  ="#329ea8";
        ctx.rotate(45*Math.PI/180);
        ctx.fillRect(80, 5, 100, 50);
        // Text ctx.rotate(45*Math.PI/180);
        ctx.font = "bold 24px Helvetica, Arial, sans-serif";
        ctx.fillStyle  =  "steelblue";
        ctx.fillText("Rotated  text",  50,  0);
     </script>
  </body>
</html>
```

Output-



## HTML5 Canvas Transforms:

In canvas transform we use 2 methods:

- transform() method
- setTransform() method

Rotation,translation,and scaling are achieved by using a transformation matrix. The transform matrix is a set of nine numbers that are used to convert a three-dimensional array, such as a bitmap using linear algebra.

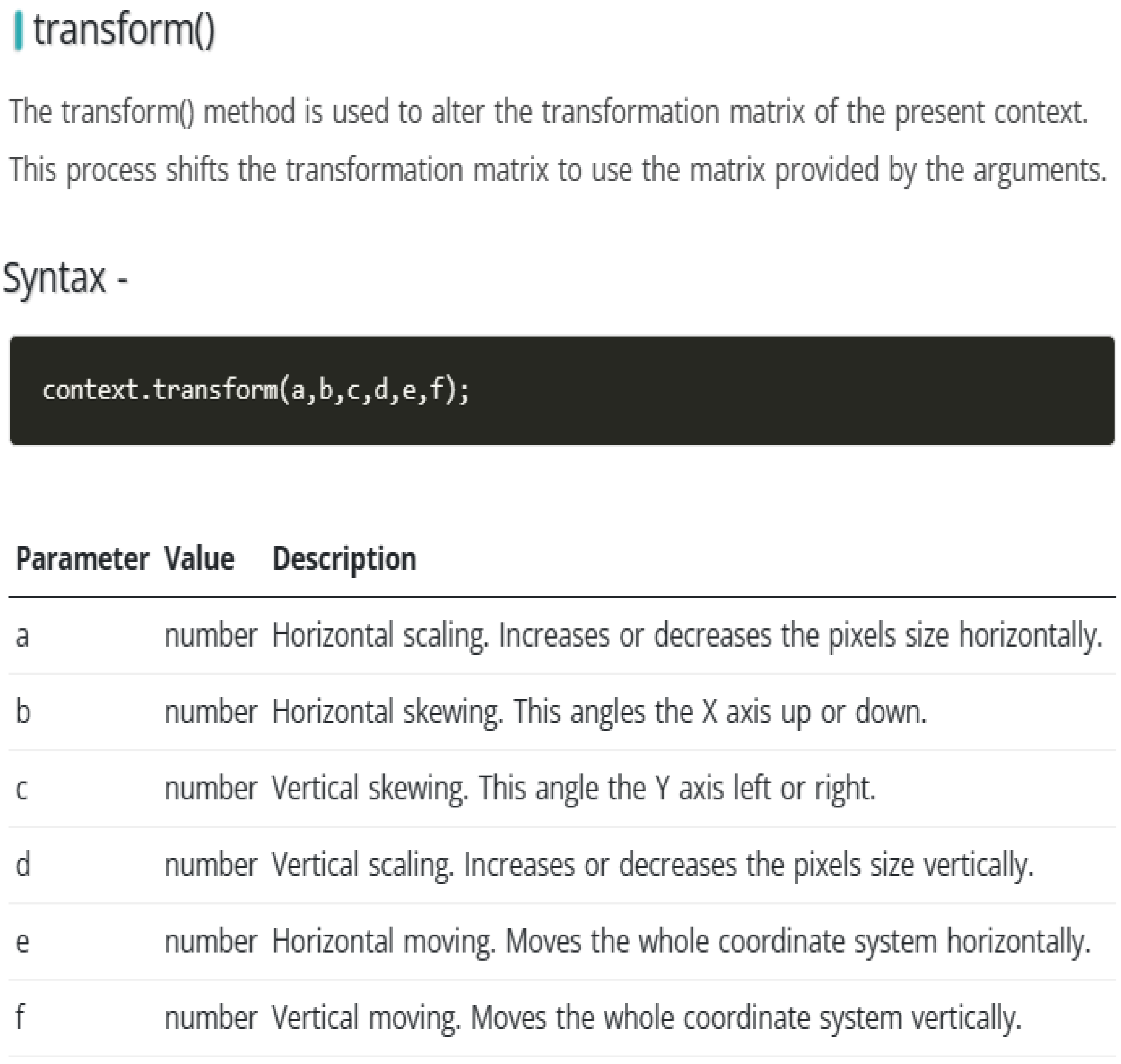
The transform()method is used to multiply that current transformation matrix with the specified matrix below–

```
 a   c   e
b   d   f
0   0   1
```

The setTransform(a, b, c, d, e, f) procedure resets the present transform to the identity matrix, and then refer to the transform(a, b, c, d, e, f) process with the similar arguments.

HTML5 canvas offers methods to modify the transformation matrix directly. The transformation matrix originally identity transform and then adjusted using the transformation procedures.

**Note!** The arguments a, b, c, d, e, and f are sometimes called m11, m12, m21, m22, dx, and dy or m11, m21, m12, m22, dx, and dy.



## transform()

The transform() method is used to alter the transformation matrix of the present context. This process shifts the transformation matrix to use the matrix provided by the arguments.

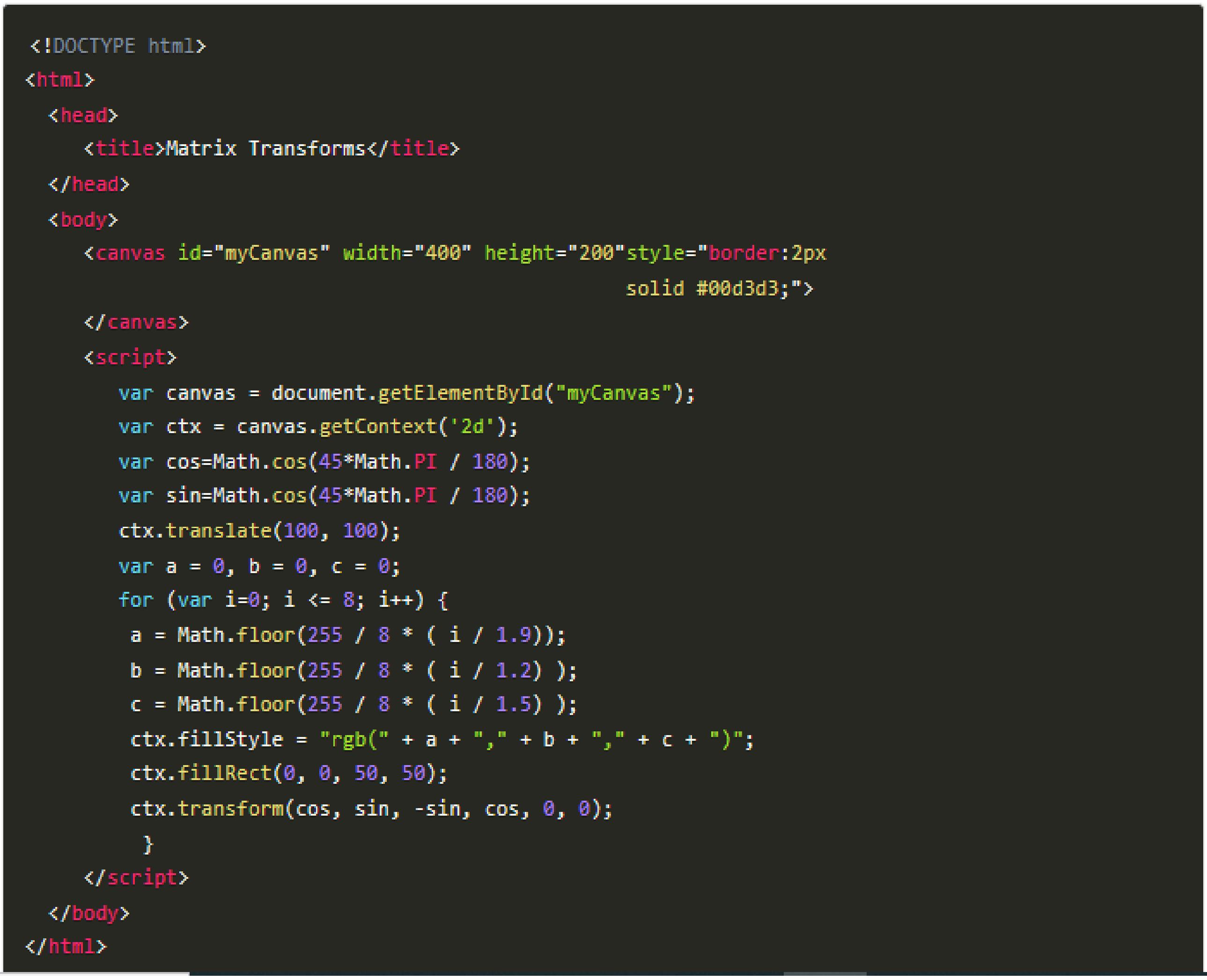
### Syntax -

```
context.transform(a,b,c,d,e,f);
```

| Parameter | Value | Description |
|---|---|---|
| a | number | Horizontal scaling. Increases or decreases the pixels size horizontally. |
| b | number | Horizontal skewing. This angles the X axis up or down. |
| c | number | Vertical skewing. This angle the Y axis left or right. |
| d | number | Vertical scaling. Increases or decreases the pixels size vertically. |
| e | number | Horizontal moving. Moves the whole coordinate system horizontally. |
| f | number | Vertical moving. Moves the whole coordinate system vertically. |

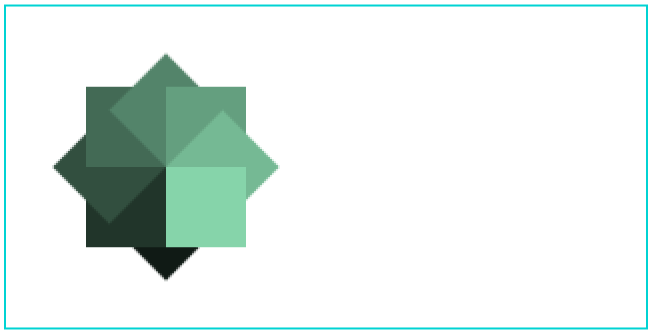
## Example -

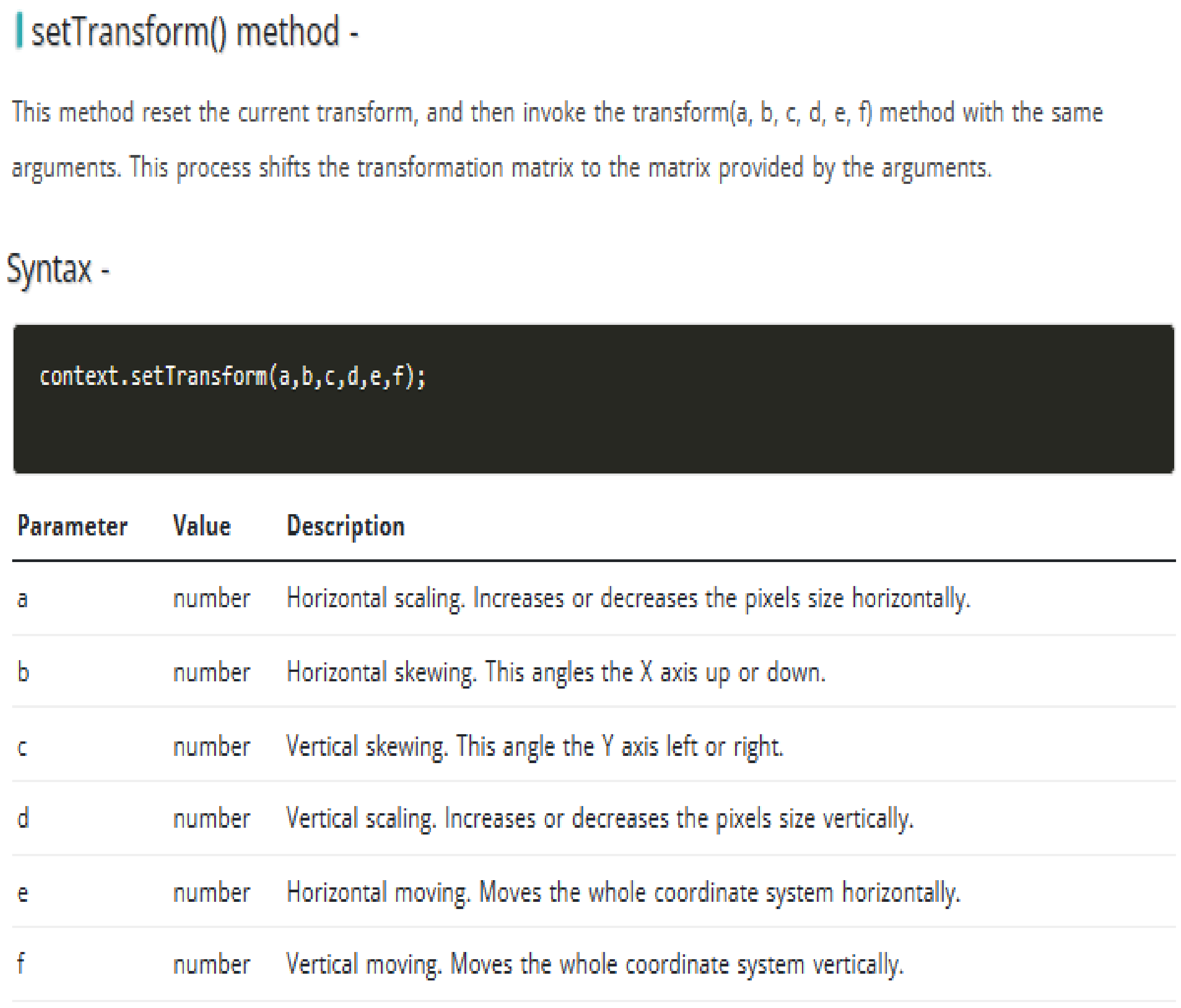
The below example shows the usage of transform() methods.



```
<!DOCTYPE html>
<html>
  <head>
     <title>Matrix Transforms</title>
  </head>
  <body>
     <canvas id="myCanvas" width="400" height="200"style="border:2px
                                                solid #00d3d3;">
     </canvas>
     <script>
        var canvas = document.getElementById("myCanvas");
        var ctx = canvas.getContext('2d');
        var cos=Math.cos(45*Math.PI / 180);
        var sin=Math.cos(45*Math.PI / 180);
        ctx.translate(100, 100);
        var a = 0, b = 0, c = 0;
        for (var i=0; i <= 8; i++) {
         a = Math.floor(255 / 8 * ( i / 1.9));
         b = Math.floor(255 / 8 * ( i / 1.2) );
         c = Math.floor(255 / 8 * ( i / 1.5) );
         ctx.fillStyle = "rgb(" + a + "," + b + "," + c + ")";
         ctx.fillRect(0, 0, 50, 50);
         ctx.transform(cos, sin, -sin, cos, 0, 0);
          }
     </script>
  </body>
</html>
```

## Output-





## setTransform() method -

This method reset the current transform, and then invoke the transform(a, b, c, d, e, f) method with the same arguments. This process shifts the transformation matrix to the matrix provided by the arguments.

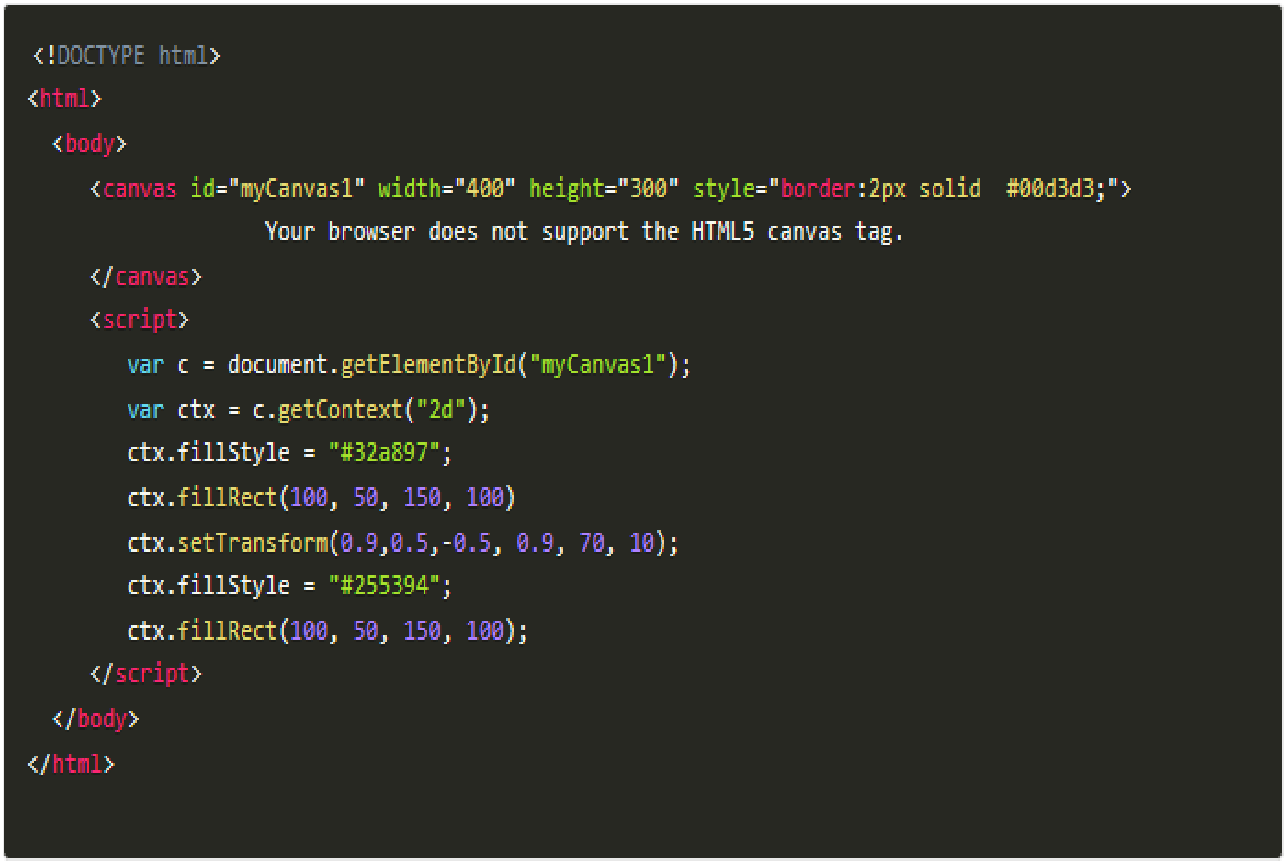
### Syntax -

```
context.setTransform(a,b,c,d,e,f);
```

| Parameter | Value | Description |
|---|---|---|
| a | number | Horizontal scaling. Increases or decreases the pixels size horizontally. |
| b | number | Horizontal skewing. This angles the X axis up or down. |
| c | number | Vertical skewing. This angle the Y axis left or right. |
| d | number | Vertical scaling. Increases or decreases the pixels size vertically. |
| e | number | Horizontal moving. Moves the whole coordinate system horizontally. |
| f | number | Vertical moving. Moves the whole coordinate system vertically. |

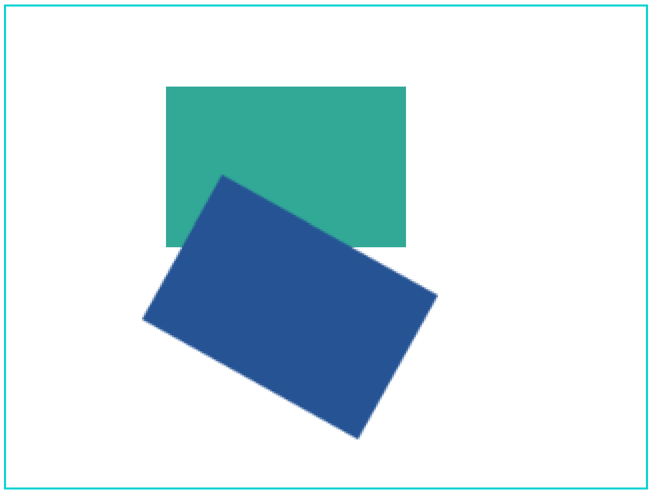
## Example -

The below example shows the usage of setTransform() methods.



```
<!DOCTYPE html>
<html>
  <body>
     <canvas id="myCanvas1" width="400" height="300" style="border:2px solid  #00d3d3;">
                    Your browser does not support the HTML5 canvas tag.
     </canvas>
     <script>
        var c = document.getElementById("myCanvas1");
        var ctx = c.getContext("2d");
        ctx.fillStyle = "#32a897";
        ctx.fillRect(100, 50, 150, 100)
        ctx.setTransform(0.9,0.5,-0.5, 0.9, 70, 10);
        ctx.fillStyle = "#255394";
        ctx.fillRect(100, 50, 150, 100);
     </script>
  </body>
</html>
```

## Output-



## HTML5 Canvas Composition:

HTML5 canvas offers compositing attribute **globalCompositeOperation** to alter entire the drawing functions. **globalCompositeOperation** sets or returns how a new drawings/images are drawn on an existing drawings/images.

We can draw fresh shapes on/behind current shapes and mask off some areas, clear sections from the canvas utilizing **globalCompositeOperation** attributes.

### Syntax -

```
context.globalCompositeOperation="source-attribute";
```

### Note: -

**source image/shape =** new drawings that are about to place onto the canvas.

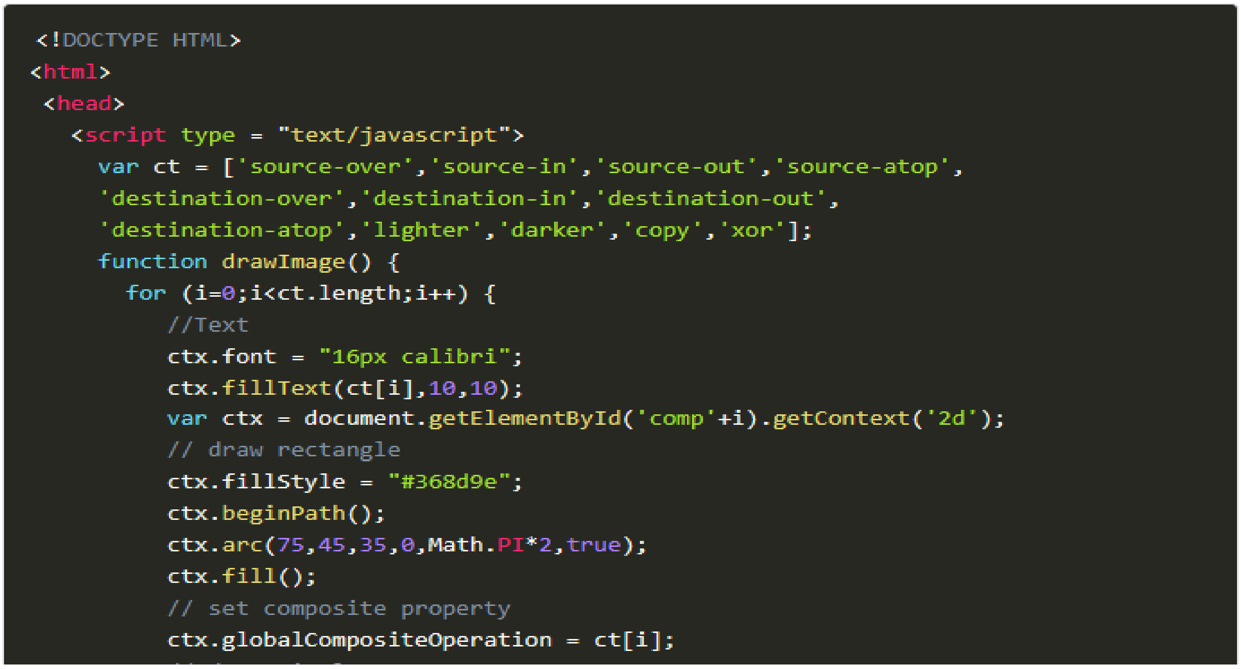
**destination image/shape =** old drawings that are already placed onto the canvas.

The following are the attributes for **globalCompositeOperation** –

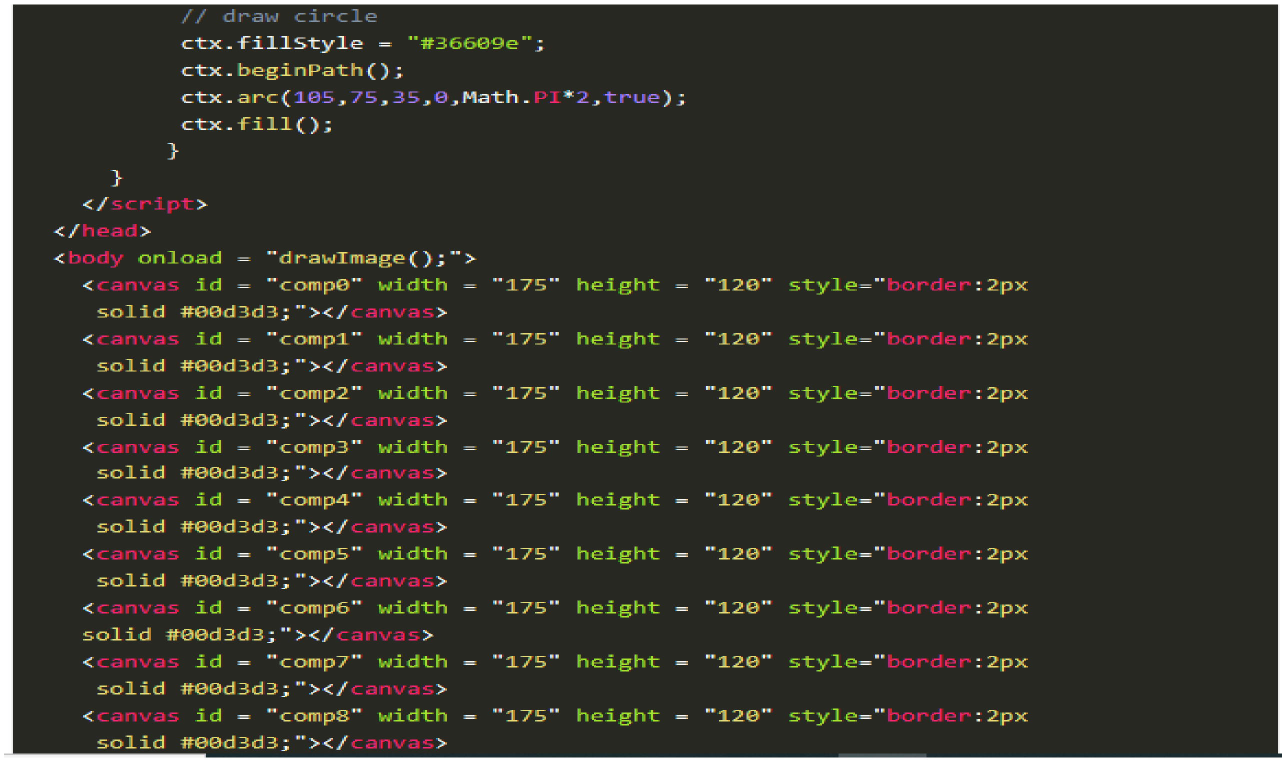
| Source-attribute | Description |
|---|---|
| source-over | This is the default value if not specified.<br>Draws/displays source shape on top of the destination shape. |
| Source-in | The source shape is drawn into the destination shape. This only shows the shape that both source and destination shapes overlapped and remaining shapes outside overlap gets cleared. |
| Source-out | Displays the source shape out of the destination shape. This only shows the destination shape that outside the overlap and remaining gets cleared. |
| Source-atop | The source shape is drawn on the destination shape. This only shows the source shape that includes destination shape overlap and remaining destination shape outside source gets cleared. |
| Lighter | Displays the source shape and the destination shape. The overlap area color gets calculated automatically from source and destination shape and displays with the color. |
| Xor | The source shape is combined by using an exclusive OR with the destination shape. i.e. the overlapped portion wont display and remaining portion of the shapes gets displayed. |
| Destination-over | Displays the destination shape on top of the source shape. |
| Destination-in | Displays the destination shape into the source shape. This displays only the destination shape overlapped area with source shape color and remaining area gets cleared. |
| Destination-out | Displays the destination shape out of the source shape.This displays the source shape only that is out of the destination overlap and remaining shapes are gets cleared. |
| Destination-atop | Displays the destination shape on top of the source shape. This displays the overlapped shape + destination shape and remaining shape gets cleared. |
| Darker | Where two shapes overlap the color is defined by deducting color values. |

## Example-

Below example describes the usage of globalCompositeOperation attribute to produce all possible compositions -

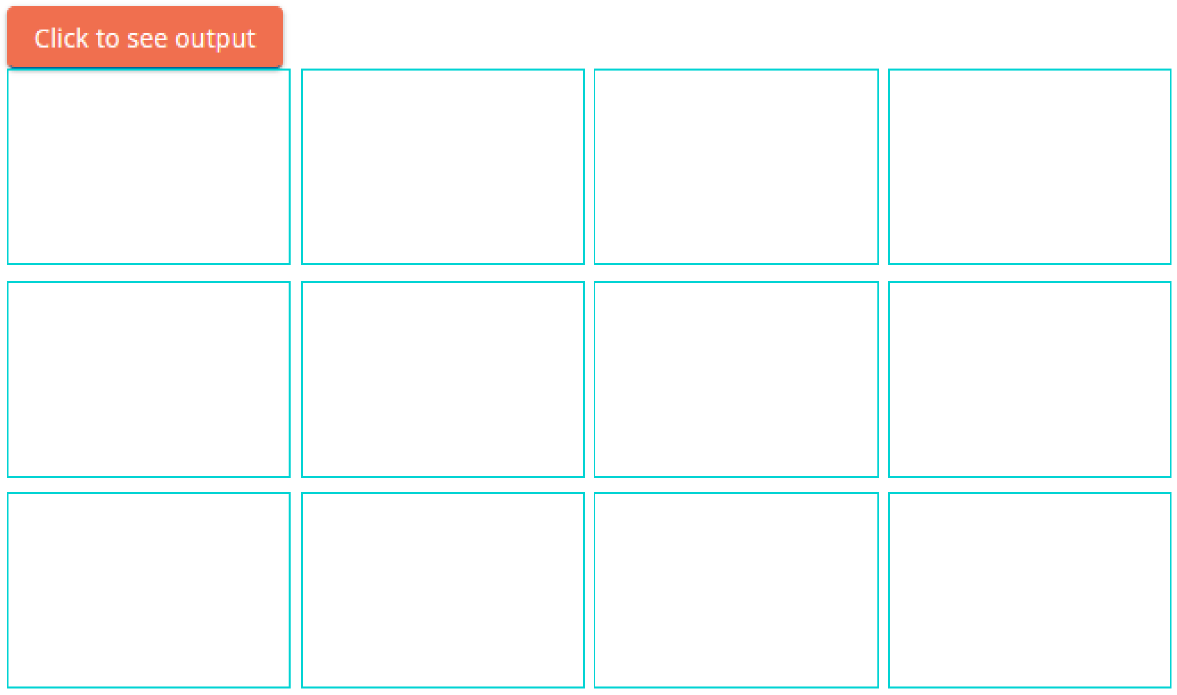


```
 <!DOCTYPE HTML>
<html>
 <head>
   <script type = "text/javascript">
     var ct = ['source-over','source-in','source-out','source-atop',
     'destination-over','destination-in','destination-out',
     'destination-atop','lighter','darker','copy','xor'];
     function drawImage() {
       for (i=0;i<ct.length;i++) {
          //Text
          ctx.font = "16px calibri";
          ctx.fillText(ct[i],10,10);
          var ctx = document.getElementById('comp'+i).getContext('2d');
          // draw rectangle
          ctx.fillStyle = "#368d9e";
          ctx.beginPath();
          ctx.arc(75,45,35,0,Math.PI*2,true);
          ctx.fill();
          // set composite property
          ctx.globalCompositeOperation = ct[i];
```



```
          // draw circle
          ctx.fillStyle = "#36609e";
          ctx.beginPath();
          ctx.arc(105,75,35,0,Math.PI*2,true);
          ctx.fill();
        }
     }
   </script>
 </head>
 <body onload = "drawImage();">
   <canvas id = "comp0" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp1" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp2" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp3" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp4" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp5" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp6" width = "175" height = "120" style="border:2px
   solid #00d3d3;"></canvas>
   <canvas id = "comp7" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
   <canvas id = "comp8" width = "175" height = "120" style="border:2px
    solid #00d3d3;"></canvas>
    <canvas id = "comp9" width = "175" height = "120" style="border:2px
      solid #00d3d3;"></canvas>
    <canvas id = "comp10" width = "175" height = "120" style="border:2px
      solid #00d3d3;"></canvas>
    <canvas id = "comp11" width = "175" height = "120" style="border:2px
      solid #00d3d3;"></canvas>
  </body>
 </html>
```

Output -



## HTML5 Canvas Animations:

**setInterval()** and **setTimeout()** methods used to produce an animation. HTML5 canvas offers essential methods to draw a picture and remove it totally. We can take Javascript support to mimic great animation over an HTML5 canvas.

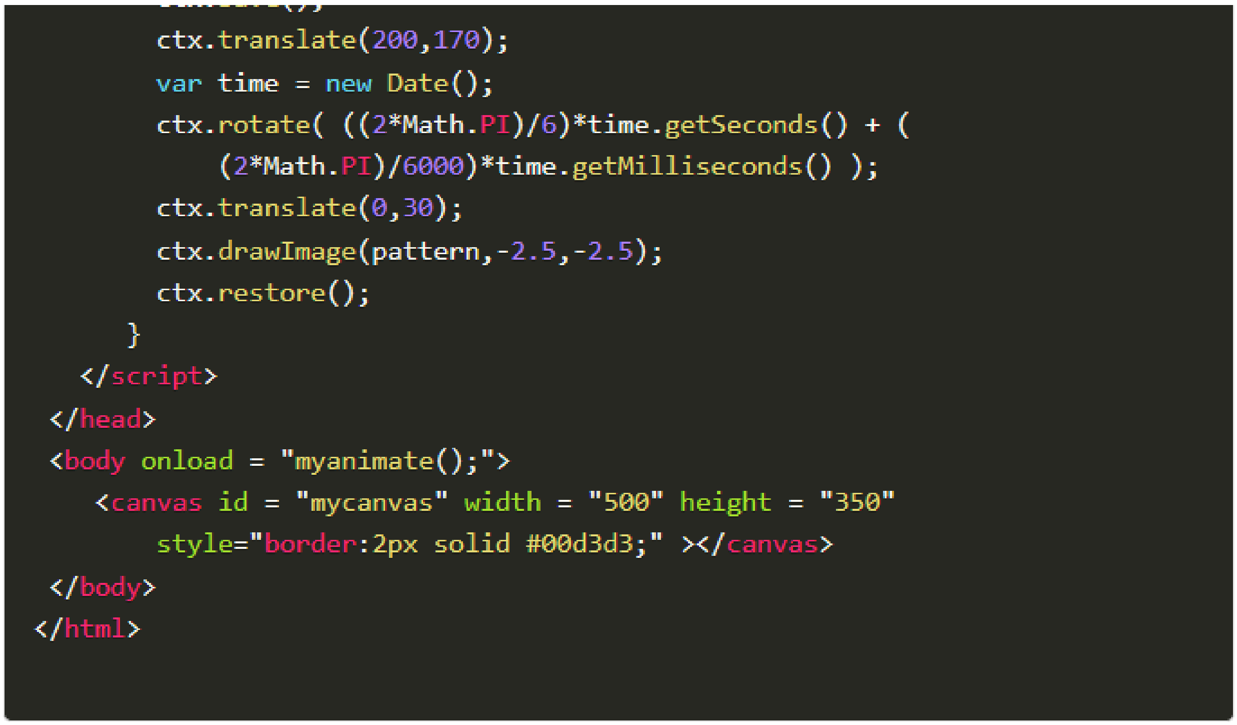
Following are the two crucial Javascript procedures which would be utilized to animate a picture on a canvas –

| Method | Description |
|---|---|
| **setInterval(callback, time);** | This process continually executes the provided code later a given time (milliseconds). |
| **setTimeout(callback, time);** | This process executes the provided code only once later a given time (milliseconds). |

### Example -

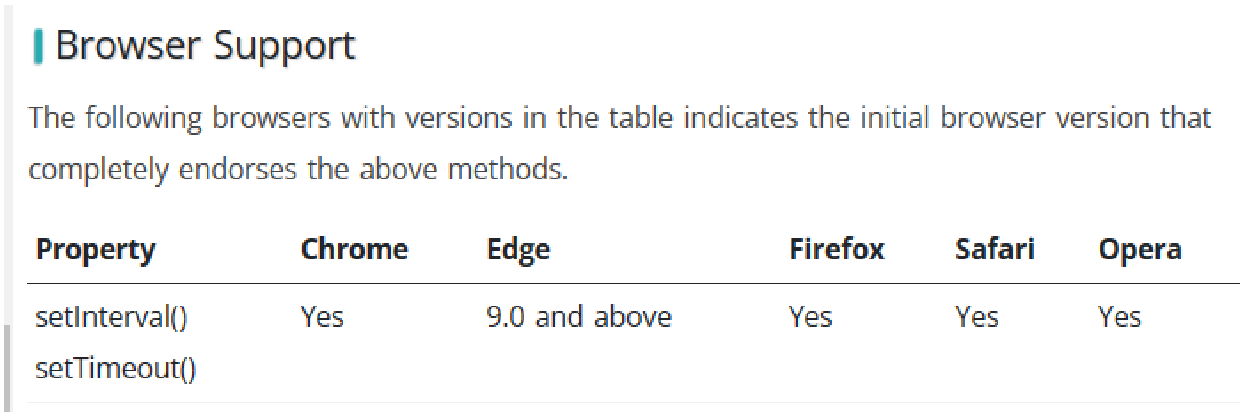
Following example shows how to turn a small picture continuously animating -

```
 <!DOCTYPE HTML>
<html>
 <head>
   <script type = "text/javascript">
      var pattern = new Image();
      function myanimate() {
            pattern.src = img/2leftarrow.png';
            setInterval(animateShape, 75);
      }
      function animateShape() {
        // get the canvas element using the DOM
        var canvas = document.getElementById('mycanvas');
        // use getContext to use the canvas for drawing
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = 'rgba(0,0,0,0.4)';
        ctx.strokeStyle = 'rgba(0,90,90,0.4)';
        ctx.save();
```



```
        ctx.translate(200,170);
        var time = new Date();
        ctx.rotate( ((2*Math.PI)/6)*time.getSeconds() + (
            (2*Math.PI)/6000)*time.getMilliseconds() );
        ctx.translate(0,30);
        ctx.drawImage(pattern,-2.5,-2.5);
        ctx.restore();
      }
   </script>
 </head>
 <body onload = "myanimate();">
    <canvas id = "mycanvas" width = "500" height = "350"
        style="border:2px solid #00d3d3;" ></canvas>
 </body>
</html>
```

Output-



## Browser Support

The following browsers with versions in the table indicates the initial browser version that completely endorses the above methods.

| Property | Chrome | Edge | Firefox | Safari | Opera |
|---|---|---|---|---|---|
| setInterval()<br>setTimeout() | Yes | 9.0 and above | Yes | Yes | Yes |

*****************************************************************