



# Working with Graphics

You often need to add graphics, such as bitmap images or 2D shapes, on the pages of your website. In addition, you can add text and apply different colors and gradients to the graphics on the Web pages. The canvas element in HTML provides a drawing surface that allows you to add text, shapes, and images to the website dynamically. In addition, you can add the transition and animation effects to the graphics to make the Web pages more visually appealing.

This chapter discusses a canvas and the various graphic objects that can be created on the canvas. In addition, it discusses how to apply animations and transformations on the graphic objects of the canvas.

## Objectives

In this lesson, you will learn to:

-  Introduce canvas
-  Transform and animate canvas elements



# Introducing Canvas

LearnGraphs Ltd. offers online tutorials on basic graphing skills. These tutorials include a wide variety of graphs, such as bar graph, histogram, and pie chart, which need to be drawn on their Web pages. However, to draw graphs on a Web page, you need to incorporate shapes, such as lines, arcs, and circles, and apply effects, such as colors and gradients, on the shapes. With the introduction of canvas element in HTML, you can create graphics and apply animations on a Web page easily. The canvas element has a huge set of functions in the form of 2D Application Programming Interface (API) that is used to draw graphics on the canvas.

## Creating a Canvas

Canvas provides an easy and a powerful way to create graphics on a Web page. A canvas has no content of its own. A canvas is simply an area on a Web page that acts as a container for embedding graphic objects. It allows dynamic rendering of bitmap images and 2D shapes by using JavaScript. You need to perform the following tasks to create a canvas and use it for drawing a variety of graphics:

1. Define the canvas
2. Access the canvas

## Defining the Canvas

A canvas is defined by using the `<CANVAS>` tag. This tag is defined in the body section of the HTML document. The `<CANVAS>` tag provides the various attributes that enable you to specify the size, border, and ID for the canvas. The attributes provided by the `<CANVAS>` tag are listed in the following table.

<i>Attribute</i>	<i>Description</i>
<i>ID</i>	<i>Is used to specify a unique ID for the canvas that is used to identify the canvas in the JavaScript code.</i>
<i>width</i>	<i>Is used to specify the width for the canvas in pixels. The default value for the <i>width</i> attribute is 300 pixels.</i>
<i>height</i>	<i>Is used to specify the height for the canvas in pixels. The default value for the <i>height</i> attribute is 150 pixels.</i>
<i>Style</i>	<i>Is used to define the style to be applied to the canvas.</i>

*The Attributes of the <CANVAS> Tag*

You can define a canvas by using the following code within the `<BODY>` tag:

```
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid black">
</CANVAS>
```

The preceding code will create a canvas with the ID, `myCanvas`, of size 300 x 300 with a solid, black border of 1px thickness.

## Accessing the Canvas Element

Defining the canvas element only creates a blank drawing surface. However, to actually draw graphic objects on the canvas, you need to access the canvas in the JavaScript code. You can write the following code in the `<BODY>` tag to access the canvas element:

```
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
</SCRIPT>
```

In the preceding code, the `getElementById()` method is used to return the reference of the element with the ID, `myCanvas`, and store the reference in the variable named `c`. Further, the `getContext()` method is used to return a drawing context object that provides the methods and properties needed for drawing graphics on the canvas. This object is stored inside the variable named `ctx`. The `getContext()` method accepts a string argument that specifies the type of canvas to be created. In the preceding code, `2d` is passed as an argument to the `getContext()` method as majority of the browsers support the creation of 2D graphics inside the canvas.

## Working with Color, Shapes, and Styles

Consider a scenario of a website for preschoolers, which provides basic learning solutions to kids. It provides simple online activities that assist kids to identify basic colors and shapes. For example, one of the activities displays a color picker and various shapes to the students. The students are asked to identify and fill a particular shape with the specified color as a part of this activity. To create such applications, you need to embed color, shapes, and styles on a Web page. In a canvas, this can be achieved by using the JavaScript predefined color and style properties and methods.

The introduction of canvas has simplified the task of drawing objects, such as rectangles, on a Web page. You can easily draw these objects by using the JavaScript methods. Further, you can also specify the colors, gradients, or pattern styles to decorate the graphic objects on the canvas.

### Working with Shapes

After creating a canvas, you can draw shapes, such as rectangle and square, on it. Rectangles and squares are the easiest shapes to draw on the canvas element by using the JavaScript functions. Using these functions, you can create a shape, clear a certain portion of the shape, and apply outline to the shape. For this, you can use the following methods:

- `rect()`
- `fillRect()`
- `strokeRect()`
- `clearRect()`

`rect()`

The `rect()` method is used to create a rectangle on the canvas. However, it picks the default color of the canvas to draw the outline of the rectangle. Therefore, the rectangle is not visible on the canvas. To make a rectangle visible on the canvas, you need to provide its outline or stroke color by using the `stroke()` method. This method uses the default black stroke to draw a rectangle.

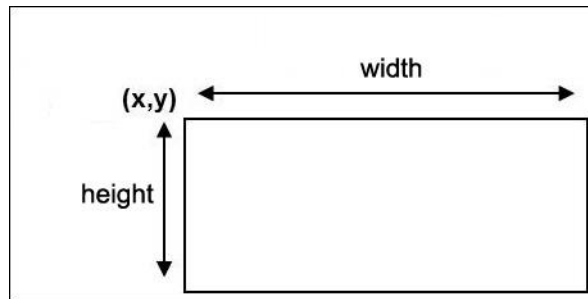
The syntax of using the `rect()` method to create a rectangle on a canvas is:

```
rect(x,y,width,height);
```

In the preceding syntax:

- `x`: Specifies the x-coordinate of the upper-left corner of the rectangle.
- `y`: Specifies the y-coordinate of the upper-left corner of the rectangle.
- `width`: Specifies the width of the rectangle, in pixels.
- `height`: Specifies the height of the rectangle, in pixels.

The following figure explains the dimensions of a rectangle.

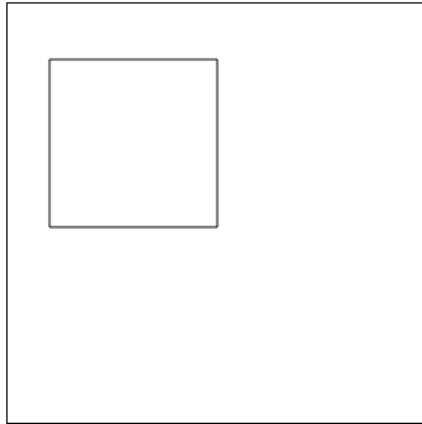


*The Dimensions of a Rectangle*

Consider the following code for creating a rectangle on the canvas having the ID, `myCanvas`:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.rect(30,40,120,120);
ctx.stroke();
</SCRIPT>
</BODY>
</HTML>
```

The preceding code creates a rectangle of size 120 x 120 starting from the coordinates, (30, 40), on the canvas, as shown in the following figure.



*The Rectangle Created on the Canvas*

## fillRect()

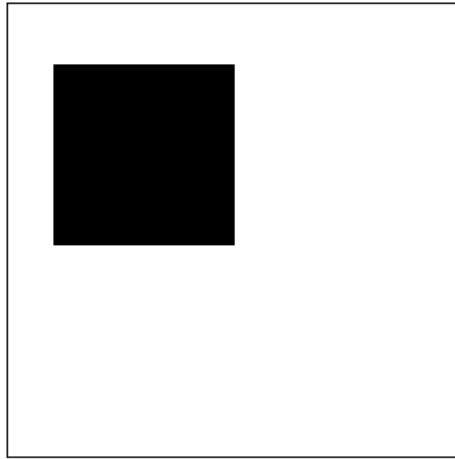
The `fillRect()` method is used to create a rectangle filled with the specified color. The default fill color is black. The following syntax is used to create a filled rectangle on a canvas:

```
fillRect(x,y,width,height);
```

Consider the following code for creating a filled rectangle on a canvas:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillRect(30,40,120,120);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code will create a rectangle filled with the black color, as shown in the following figure.



*A Rectangle Filled with Black Color*

`strokeRect()`

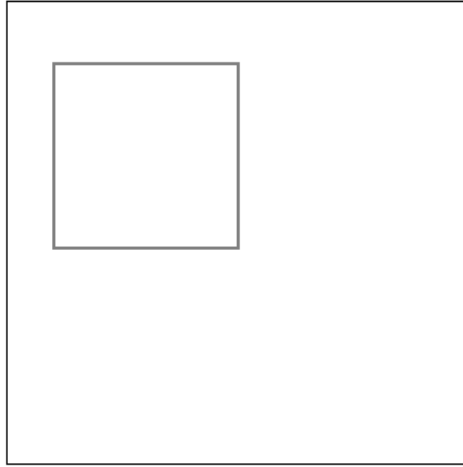
When you create a rectangle by using the `rect()` method, you also need to use the `stroke()` method to define its outline on the canvas. Instead of using two methods, `rect()` and `stroke()`, to draw a rectangle on a canvas, you can use the single method, `strokeRect()`, to draw a rectangle with the specified stroke color. By default, the `strokeRect()` method uses the black color to create an outline of the rectangle. The syntax for using the `strokeRect()` method to draw a rectangle on a canvas is:

```
strokeRect(x,y,width,height);
```

Consider the following code for drawing a rectangle on a canvas:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.strokeRect(30,40,120,120);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code will create a rectangle of size 120 x 120 at the position, (30, 40), on the canvas, as shown in the following figure.



*The Rectangle Created by Using the strokeRect() Method*

### clearRect()

The `clearRect()` method is used to clear a portion of the rectangle. It clears the specified pixels within the given rectangle. The following syntax can be used to clear a rectangle on a canvas:

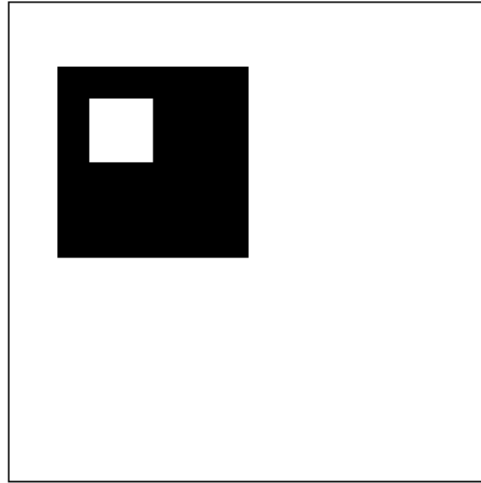
```
clearRect(x,y,width,height);
```

Consider the following code to clear a part of the rectangle created on the canvas having the ID, `myCanvas`:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillRect(30,40,120,120);
ctx.clearRect(50,60,40,40);
</SCRIPT>
</BODY>
</HTML>
```



The preceding code clears a portion of the rectangle of size 40 x 40 starting from the coordinates, (50, 60), as shown in the following figure.



*The Output Derived by Using the clearRect() Method*

Now, consider the following code to draw rectangular shapes on the canvas by using the various methods:

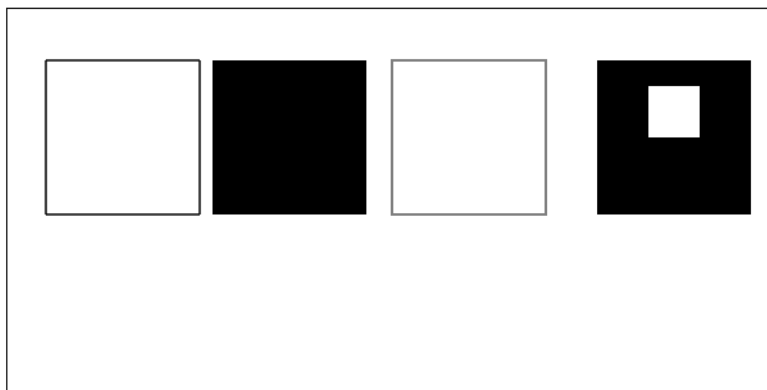
```
<!DOCTYPE HTML>
<HTML>

<BODY>
<CANVAS ID="myCanvas" width="600" height="300"
style="border:1px solid black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.rect(30,40,120,120);
ctx.stroke();

ctx.fillRect(160,40,120,120);
ctx.stroke();

ctx.strokeRect(300,40,120,120);
ctx.fillRect(460,40,120,120);
ctx.clearRect(500,60,40,40);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code creates rectangles on the canvas, as shown in the following figure.



*The Rectangles Created on the Canvas*

## Working with Colors

The graphic objects on a canvas are created by using the default stroke and fill color. However, you can use colors other than the default color for creating the graphic objects. The following properties can be used to apply colors on the canvas objects:

- `fillStyle`
- `strokeStyle`
- `shadowColor`

### `fillStyle`

The `fillStyle` property is used to define a color that will be used to fill any closed shape drawn on the canvas. The default value of the `fillStyle` property is solid black. The following syntax is used to apply the fill style on a graphic object:

```
fillStyle="color";
```

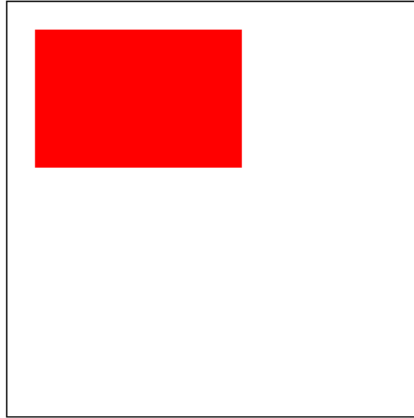
In the preceding syntax, you can specify the color as red, green, or blue. In addition, you can also specify the hexadecimal value of the color ranging from 000000 to FFFFFFFF.

Consider the following code for applying the fill style on a rectangle drawn on the canvas having the ID, `myCanvas`:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="red";
ctx.fillRect(20,20,150,100);
```

```
</SCRIPT>
</BODY>
</HTML>
```

The preceding code snippet creates a rectangle of size 150 x 100 filled with the red color at the position, (20, 20), on the canvas, as shown in the following figure.



*A Rectangle Filled with the Red Color*

## strokeStyle

The `strokeStyle` property is used to set the outline color of a shape drawn on the canvas. The default value of the `strokeStyle` property is solid black. The following syntax can be used to apply the stroke style on a graphic object:

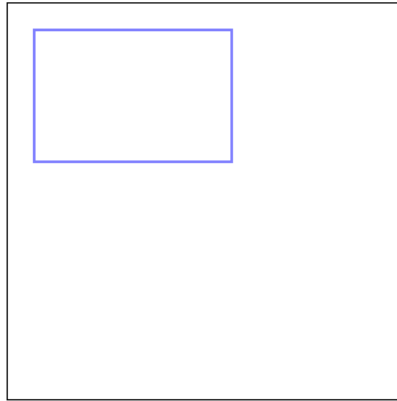
```
strokeStyle="color";
```

In the preceding syntax, `color` specifies the name or hexadecimal value of the color.

Consider the following code for applying the stroke style on a rectangle:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.strokeStyle="blue";
ctx.strokeRect(20,20,150,100);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code will create a rectangle of size 150 x 100 with its outline colored in blue at the position, (20, 20), on the canvas, as shown in the following figure.



*The Rectangle with Strokes Colored in Blue*

## shadowColor

Once you have drawn a shape on the canvas, you may want to make it more stylish by casting a shadow on it. To cast a shadow of a graphic object on the canvas, you need to specify the color of the shadow. In addition, you need to specify how blurred you want your shadow to be. The `shadowColor` property is used to set the color for the shadows appearing on the graphic objects and the `shadowBlur` property is used to set the blur level for the shadows.

You can use the following syntax to use the `shadowColor` property:

```
shadowColor="color";
```

In the preceding syntax, `color` specifies the color that will be applied on shadows. The default value of the `shadowColor` property is solid black.

You can use the following syntax to define the `shadowBlur` property:

```
shadowBlur=number;
```

In the preceding syntax, `number` specifies the blur level of the shadow. It can accept the integer values, such as 1, 2, and 20. Its default value is 0.

Consider the following code for applying shadows on a rectangle:

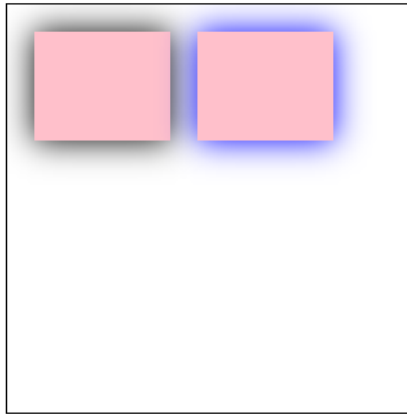
```
<!DOCTYPE HTML>
<HTML>
  <BODY>
    <CANVAS ID="myCanvas"   width="300" height="300" style="border:1px solid
    black">
  </CANVAS>
</SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.shadowBlur=40;
```

```

ctx.fillStyle="pink";
ctx.shadowColor="black";
ctx.fillRect(20,20,100,80);
ctx.shadowColor="blue";
ctx.fillRect(140,20,100,80);
</SCRIPT>
</BODY>
</HTML>

```

In the preceding code, the blur level of the shadow of graphic objects is set to 40. In addition, the shadow color for the first rectangle is set to black and the shadow color for the second rectangle is set to blue. The output derived by using the `shadowBlur` and `shadowColor` properties is displayed in the following figure.



*The Output Derived by Using the shadowBlur and shadowColor Properties*

## Working with Styles

Apart from creating simple shapes on the canvas, you can also apply styles, such as gradients, on them. A gradient is an object that provides smooth transition between two or more colors. To work with gradient styles, you can use the following methods:

- `addColorStop()`
- `createLinearGradient()`
- `createRadialGradient()`
- `createPattern()`

### `addColorStop()`

To create gradients, you need to first specify the colors and their positions in a gradient object. This is because the gradients are not visible until the colors are added to the objects. Therefore, to actually make the gradient effects visible on a graphic object, you need to add colors. You can add one or more colors on a gradient object by using the `addColorStop()` method. The `addColorStop()` method is used to specify the colors and their corresponding positions in a gradient object. The following syntax can be used to define the `addColorStop()` method:

```
addColorStop(position,color);
```

In the preceding syntax:

- **position**: Specifies a value between 0.0 to 1.0 to represent the position from where to start and end the gradient color.
- **color**: Specifies the color that needs to be applied on the respective position.

The `addColorStop()` method is used along with the `createLinearGradient()` or `createRadialGradient()` method to display the gradients.

`createLinearGradient()`

The `createLinearGradient()` method is used to return a gradient object that represents a linear gradient for painting the specified color along a line. The following syntax can be used to apply a linear gradient:

```
createLinearGradient(x0,y0,x1,y1);
```

In the preceding syntax:

- **x0**: Specifies the x-coordinate of the start point of the gradient.
- **y0**: Specifies the y-coordinate of the start point of the gradient.
- **x1**: Specifies the x-coordinate of the end point of the gradient.
- **y1**: Specifies the y-coordinate of the end point of the gradient.

After creating the linear gradient object, you need to create the gradients by using the `addColorStop()` method. Once you have created the linear gradient, you need to apply it on a graphic object by using the following ways:

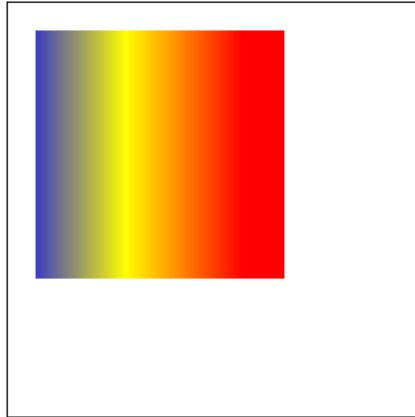
- Fill the graphic object with the linear gradient by using the `fillStyle` property.
- Apply the linear gradient on the outline of the graphic object by using the `strokeStyle` property.

Consider the following code for applying a linear gradient on a rectangle:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grad=ctx.createLinearGradient(0,0,170,0);
grad.addColorStop(0,"blue");
grad.addColorStop("0.5","yellow");
grad.addColorStop(1,"red");
ctx.fillStyle=grad;
ctx.fillRect(20,20,180,180);
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, a gradient object is created by using the `createLinearGradient()` method. Further, the `addColorStop()` method is used to specify different colors to the gradient object, and then the gradient object is passed to the `fillStyle` property to shade the rectangle in three different colors from left

to right. The output derived by using the `createLinearGradient()` method is displayed, as shown in the following figure.



*The Output Derived by Using the `createLinearGradient()` Method*

### `createRadialGradient()`

The `createRadialGradient()` method is used to return a gradient object that represents a radial or circular gradient to be applied on a graphic object. A circular gradient paints colors along a cone specified by two circles, inner and outer. The following syntax can be used to apply a radial gradient:

```
createRadialGradient(x0,y0,r0,x1,y1,r1);
```

In the preceding syntax:

- `x0`: Specifies the x-coordinate of the start point of the gradient.
- `y0`: Specifies the y-coordinate of the start point of the gradient. (`x0,y0`) specifies the center coordinate of the first circle of the cone.
- `r0`: Specifies the radius of the starting circle.
- `x1`: Specifies the x-coordinate of the end point of the gradient.
- `y1`: Specifies the y-coordinate of the end point of the gradient. (`x1,y1`) specifies the center coordinate of the second circle of the cone.
- `r1`: Specifies the radius of the ending circle.

After creating the radial gradient object, you need to create the gradients by using the `addColorStop()` method. Once you have created the radial gradient, you need to apply it on a graphic object by using the following ways:

- Fill the graphic object with the radial gradient by using the `fillStyle` property.
- Apply the radial gradient on the outline of the graphic object by using the `strokeStyle` property.

Consider the following code for applying a radial gradient on a rectangle:

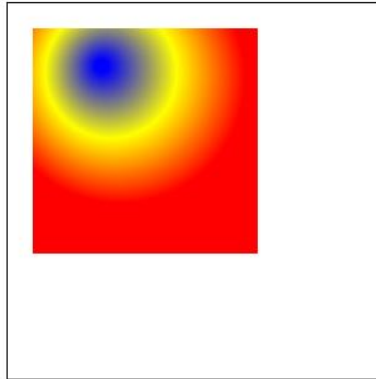
```
<!DOCTYPE HTML>
<HTML>
  <BODY>
    <CANVAS ID="myCanvas"    width="300" height="300" style="border:1px solid black">
```

```

</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grad=ctx.createRadialGradient(75,50,5,90,60,100);
grad.addColorStop(0,"blue");
grad.addColorStop("0.5","yellow");
grad.addColorStop(1,"red");
ctx.fillStyle=grad;
ctx.fillRect(20,20,180,180);
</SCRIPT>
</BODY>
</HTML>

```

In the preceding code, a gradient object is created by using the `createRadialGradient()` method. Further, the `addColorStop()` method is used to specify different colors to the gradient object, and then the gradient object is passed to the `fillStyle` property to shade the rectangle in three different colors along the radius given for the circle. The output derived by using the `createRadialGradient()` method is displayed in the following figure.



*The Output Derived by Using the createRadialGradient() Method*

`createPattern()`

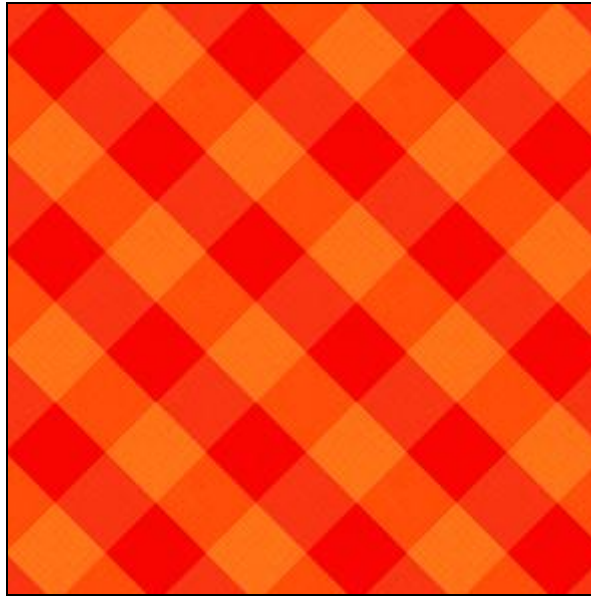
The `createPattern()` method is used to create a pattern by displaying an image repeatedly on a canvas in the specified direction. For example, consider the following image.



*The Image*



If the preceding image is repeated vertically and horizontally, you can create a pattern, as shown in the following figure.



*A Pattern*

The following syntax can be used to create a pattern:

```
createPattern(img, "repeat|repeat-x|repeat-y|no-repeat");
```

In the preceding syntax:

- **img**: Specifies the image or video to be used to create a pattern.
- **repeat**: Specifies that the pattern should be repeated horizontally and vertically.
- **repeat-x**: Specifies that the pattern should be repeated horizontally.
- **repeat-y**: Specifies that the pattern should be repeated vertically.
- **no-repeat**: Specifies that the pattern should be displayed only once.

Consider the following code snippet for repeating an image horizontally and vertically:

```
<P>Image to use:</P>
<IMG src="pattern.png" ID="pattern">
<P>Canvas:</P>
<BUTTON onclick="draw('repeat')">Repeat</BUTTON>
<CANVAS ID="myCanvas" width="300" height="150" style="border:1px solid
#d3d3d3;">
Your browser does not support the HTML5 canvas tag.</CANVAS>
<SCRIPT>
function draw(direction)
{
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var img=document.getElementById("pattern")
```

```

var pat=ctx.createPattern(img,direction);
ctx.rect(0,0,300,150);
ctx.fillStyle=pat;
ctx.fill();
}
</SCRIPT>

```

The preceding code snippet repeats the `pattern.png` image horizontally and vertically in the rectangular area on the canvas, when the user clicks the **Repeat** button, as shown in the following figure.



*The Repeating Image on the Canvas*



***Just a minute:***

*Which one of the following methods can be used to remove a certain portion of the rectangle on the canvas?*

1. `rect()`
2. `createRect()`
3. `strokeRect()`
4. `clearRect()`

***Answer:***

4. `clearRect()`



## Working with Path, Text, and Images

LearnGraphs Ltd. offers tutorials on creating pie charts. For this, a pie chart, along with its caption, needs to be drawn. However, a pie chart is a circular chart that is divided into various sectors. To create a pie chart, you need to draw a circle. In addition, to divide the circle into various sectors, you need to draw lines. In a canvas, you can create shapes, such as circles, lines, arcs, and triangles, by using path methods and properties.

Further, to insert the caption for the pie chart, you need to insert text. You can insert text in a canvas by using the text properties and methods. In addition, you can also add images to the canvas.

## Working with Path

A path is a series of points joined together to create lines or shapes. In a canvas, you can use lines or paths to draw shapes other than rectangles or squares. Using paths or lines, you can create shapes, such as circles, polygon, or triangles. However, to create a path, you need to first start or begin the path. Next, you need to call methods, such as `moveTo()` and `lineTo()`, to actually draw the path. Finally, you need to end or close the path so that the shape gets created. To create shapes by using path, you can use various methods. These methods are described in the following table.

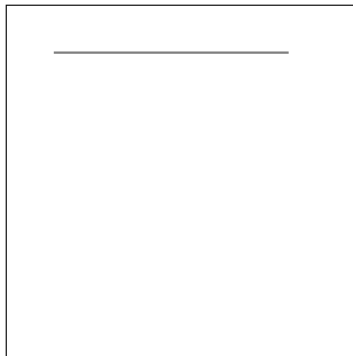
<b>Method</b>	<b>Description</b>
<code>fill()</code>	<i>Is used to fill the path on the canvas. The default color is black. You can change the fill color by using the <code>fillStyle</code> property.</i>
<code>stroke()</code>	<i>Is used to draw the outline of the path. The default color is black. You can change the outline color by using the <code>strokeStyle</code> property.</i>
<code>beginPath()</code>	<i>Is used to begin a path or reset the current path.</i>
<code>moveTo(x,y)</code>	<i>Is used to move the path to the <math>(x,y)</math> coordinates on the canvas.</i>
<code>closePath()</code>	<i>Is used to create a path from the current position back to the starting position.</i>
<code>lineTo(x,y)</code>	<i>Is used to create a line from the starting position to the ending position specified by the <math>(x,y)</math> coordinates. The starting position is defined by the <math>(x,y)</math> coordinates specified in the <code>moveTo()</code> method.</i>
<code>clip()</code>	<i>Is used to clip a specified area from the canvas.</i>
<code>arc(x,y,r,sAngle,eAngle,counterClockwise)</code>	<i>Is used to create an arc or a curve on the coordinate points, <math>(x,y)</math>, with the radius, <math>r</math>, from the starting angle, <math>sAngle</math>, to the ending angle, <math>eAngle</math>, on the canvas. The last parameter specifies whether the drawing should be counterclockwise or clockwise. The false value specifies clockwise and the true value specifies counterclockwise.</i>
<code>arcTo(x1,y1,x2,y2,r)</code>	<i>Is used to create an arc between two coordinate points, <math>(x1,y1)</math> and <math>(x2,y2)</math>, with the radius, <math>r</math>, on the canvas.</i>

*The Path Methods*

Consider the following code for creating a line on a canvas:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.moveTo(40, 40);
ctx.lineTo(240, 40);
ctx.stroke(); </SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `beginPath()` method will begin the path on the canvas. Further, the `lineTo()` method will draw a straight line from the canvas coordinates, (40,40), specified in the `moveTo()` method up to the coordinates, (240, 40), as shown in the following figure.

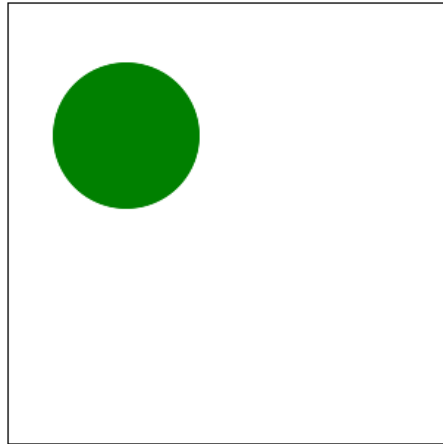


*The Line Drawn on a Canvas*

Consider the following code for creating a circle on a canvas:

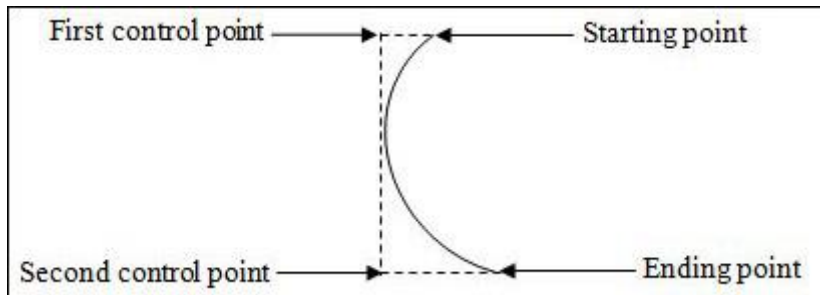
```
<!DOCTYPE HTML>
<HTML> <BODY>
<CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(80, 90, 50, 0, Math.PI*2, false);
ctx.closePath();
ctx.fillStyle="green";
ctx.fill();
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, a circle of the radius, 50, filled with the green color is drawn on the canvas at the coordinates, (80, 90), as shown in the following figure.



*The Output Derived by Using the arc() Method*

In addition to drawing lines, arcs, or circles, you can also draw more complex curvatures, such as the bezier curves. A bezier curve is defined with a context or starting point, two control points, and an ending point, as shown in the following figure.



*The Bezier Curve*

You can create the bezier curves by using the path method, `bezierCurveTo()`. The following syntax can be used to create a bezier curve:

```
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);
```

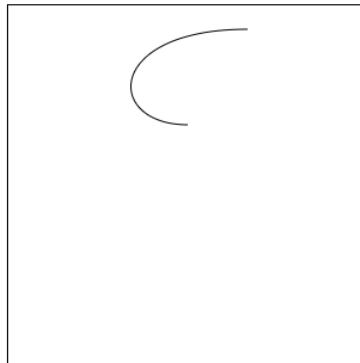
In the preceding syntax:

- `cp1x`: Specifies the x-coordinate of the first control point.
- `cp1y`: Specifies the y-coordinate of the first control point.
- `cp2x`: Specifies the x-coordinate of the second Bezier control point.
- `cp2y`: Specifies the y-coordinate of the second Bezier control point.
- `x`: Specifies the x-coordinate of the ending point.
- `y`: Specifies the y-coordinate of the ending point.

Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="300" height="300"
style="border:1px solid black">
</CANVAS>
<SCRIPT>
var c=document.getElementById('myCanvas');
var ctx=c.getContext('2d');
ctx.beginPath();
ctx.moveTo(200,20);
ctx.bezierCurveTo(80,20,80,100,150,100);
ctx.stroke();
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, a bezier curve is drawn starting from the point, (200, 20), to the point, (150, 100), as shown in the following figure.



*The Bezier Curve*

## Working with Text

In addition to drawing shapes or lines on the canvas, you can also draw text on it. You can also apply different styles on the text. For this, you can use the various text properties. The following table lists the text properties.

<b><i>Property</i></b>	<b><i>Value</i></b>	<b><i>Description</i></b>
<i>font</i>	<i>font-style font-variant font-weight font-size font-family</i>	<i>Is used to set the font for the text on the canvas.</i>
<i>textAlign</i>	<i>start end center left right</i>	<i>Is used to set the alignments for the text.</i>

<i>Property</i>	<i>Value</i>	<i>Description</i>
<code>textBaseLine</code>	<code>top hanging middle bottom alphabetic</code>	<i>Is used to set the baseline for the text where it will be drawn relative to its starting point.</i>

### *The Text Properties*

The preceding properties can be used to decorate the text. However, you need to use the following methods to actually draw the text on a canvas:

- `fillText()`
- `strokeText()`

#### `fillText()`

The `fillText()` method is used to draw a text filled with the solid color on a canvas. The default value for `fillText()` is black. The following syntax can be used to draw a filled text:

```
fillText(text,x,y,width);
```

In the preceding syntax:

- **text**: Specifies the text to be written on the canvas.
- **x**: Specifies the x-coordinate of the starting point of the text.
- **y**: Specifies the y-coordinate of the starting point of the text.
- **width**: Specifies the width of the text.

Consider the following code for drawing a filled text on a canvas:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas" width="200" height="200" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="15px Georgia";
ctx.fillText("Fill Text",10,60,120);
ctx.font="15x Verdana";
var gradient=ctx.createLinearGradient(0,0,170,0);
gradient.addColorStop("0","magenta");
gradient.addColorStop("0.5","blue");
gradient.addColorStop("1.0","red");
ctx.fillStyle=gradient;
ctx.fillText("Fill Text with Gradient",10,90,120);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code sets the font style of the text to 15px Georgia by using the `font` property and draws the text, `Fill Text`, at the canvas coordinates, (10, 60), on the canvas. Similarly, it draws the text, `Fill`

Text with Gradient, at the canvas coordinates, (10, 90), and applies the gradient on it by using the `createLinearGradient()` method and the `fillStyle` property, as shown in the following figure.



*The Output Derived by Using the `fillText()` Property*

`strokeText()`

The `strokeText()` method is used to draw a text at the specified position on the canvas by using the current font style and color. The default outline color used by this method to draw a text on the canvas is black. The syntax to use the `strokeText()` method is:

```
strokeText(text,x,y,width);
```

In the preceding syntax:

- `text`: Specifies the text to be written on the canvas.
- `x`: Specifies the x-coordinate of the starting point of the text.
- `y`: Specifies the y-coordinate of the starting point of text.
- `width`: Specifies the width of the text.

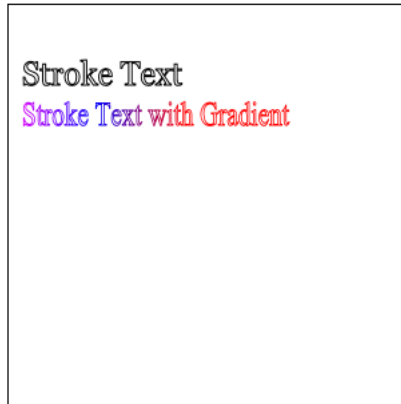
Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
  <BODY>
    <CANVAS ID="myCanvas" width="300" height="300" style="border:1px solid
    black">
    </CANVAS>
    <SCRIPT>
      var c=document.getElementById("myCanvas");
      var ctx=c.getContext("2d");
      ctx.font="25px Georgia";
      ctx.strokeText("Stroke Text",10,60,120);
      ctx.font="25x Verdana";
      var gradient=ctx.createLinearGradient(0,0,170,0);
      gradient.addColorStop("0","magenta");
      gradient.addColorStop("0.5","blue");
      gradient.addColorStop("1.0","red");
      ctx.strokeStyle=gradient;
```



```
ctx.strokeText("Stroke Text with Gradient",10,90,200);
</SCRIPT>
</BODY>
</HTML>
```

The preceding code sets the font style of the text to 25px Georgia by using the `font` property and draws the text, `Stroke text`, at the canvas coordinates, (10,60), on the canvas. Similarly, it draws the text, `Stroke Text with Gradient`, at the canvas coordinates, (10,90), and applies the gradient on it by using the `createLinearGradient()` method and the `strokeStyle` property, as shown in the following figure.



*The Output Derived by Using the `strokeText()` Method*

## Working with Images

In a canvas, you can also draw images, image clips, or render videos. For this, you can use the `drawImage()` method. The `drawImage()` method is used to draw an image or a video on the canvas. In addition, it enables you to draw a part of an image on the canvas. To define the `drawImage()` method, you can use the following syntaxes:

```
drawImage(img,x,y);
drawImage(img,x,y,width,height);
drawImage(img,sx,sy,swidth,sheight,x,y,width,height);
```

In the preceding syntax:

- `img`: Specifies an image that needs to be drawn on the canvas.
- `x`: Specifies the x-coordinate for the starting point of the image.
- `y`: Specifies the y-coordinate for the starting point of the image.
- `width`: Specifies the width of the image. It is an optional argument. If not specified, the actual width of the image is taken by default.
- `height`: Specifies the height of the image. If not specified, the actual height of the image is taken by default.
- `sx`: Specifies the x-coordinate where to start the clipping of the image.
- `sy`: Specifies the y-coordinate where to start the clipping of the image.
- `swidth`: Specifies the width of the clipped image.
- `sheight`: Specifies the height of the clipped image.

Consider the following code to draw an image on a canvas:

```
<!DOCTYPE HTML>
<HTML>
<BODY onload=setImage()>
<CANVAS ID="myCanvas" width="250" height="300" style="border:1px solid
#d3d3d3;">
Your browser does not support the HTML5 canvas tag.</CANVAS>
<SCRIPT>
function setImage(){
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var img=new Image();
img.src="orchid.jpg";
img.onload = function() {
    ctx.drawImage(img,10,10,150,150);
}
}
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the image is drawn on the canvas at the point, (10, 10), and of size 150 x 150, as shown in the following figure.



*The Drawing of an Image on the Canvas*

## Working with Graphs

A graph is a way of representing relationships among a collection of items. A graph consists of a set of objects, called nodes, which are connected by links called edges. In a canvas, you can create graphs, such as bar graph or pie chart, to represent relationships. These graphs can be created by using methods provided to draw the shapes. However, to draw graphs by using these methods is a tedious task. To simplify this task, you can use the various freely-downloadable JavaScript libraries. One such library is RGraph. To create graphs by using the RGraph library, you need to download this light-weight JavaScript library and save it in your system. RGraph allows you to create different types of graphs on a Web page.

Once the JavaScript library is downloaded, it can be referred to a Web page by using the `<SCRIPT>` tag in the head section of the Web document.

The following syntax is used to specify the jQuery library:

```
<SCRIPT type="text/javascript" src="RGraph_file_name" ></SCRIPT>
```

In the preceding syntax:

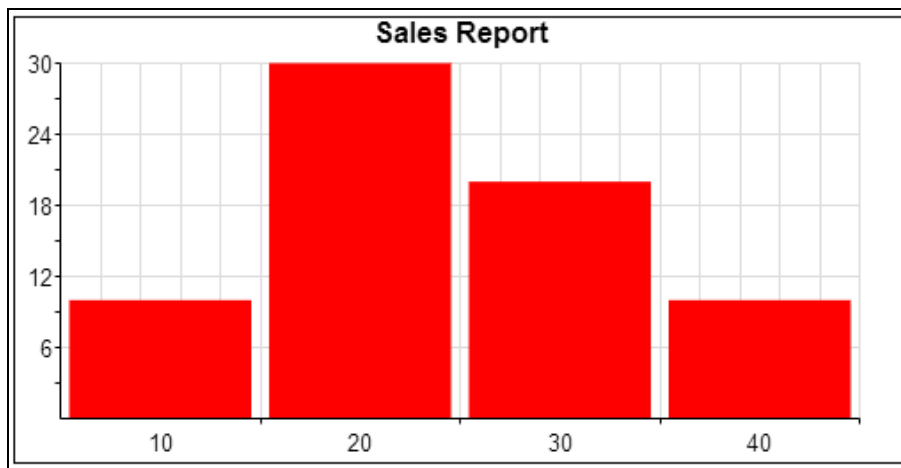
- The `<SCRIPT>` tag instructs the browser that the HTML document uses a script.
- The `type` attribute specifies the type of scripting used.
- The `src` attribute specifies the name of the JavaScript library used. If the JavaScript library is stored at the same location as the HTML document, only the name of the library can be given. However, if the JavaScript library and the HTML document are stored at different locations, you have to specify the complete path of the JavaScript library.

For example, you want to create a bar graph on a Web page. For this, you need to download the **RGraph.common.core.js** and **RGraph.bar.js** files. Consider the following code snippet to create a bar graph on a Web page:

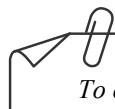
```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT type="text/javascript" src="RGraph.common.core.js" ></SCRIPT>
<SCRIPT type="text/javascript" src="RGraph.bar.js" ></SCRIPT>
  <TITLE>Bar chart</TITLE>
</HEAD>
<BODY>

<CANVAS width="500" height="250" ID="test" style="border:1px solid
black"></CANVAS>
<SCRIPT type="text/javascript" charset="utf-8">
var bar = new RGraph.Bar('test', [10,30,20,10]);
bar.Set('chart.colors', ['red']);
bar.Set('chart.title', "Sales Report" );
bar.Set('chart.labels', ["10" , "20" , "30" , "40" ]);
bar.Draw();
</SCRIPT>
</BODY>
</HTML>
```

The preceding code creates a graph on the canvas, as shown in the following figure.



*A Bar Graph*



**Note**

To create different graphs on a Web page, you need to download different JavaScript files. For this, you can refer to the <http://www.rgraph.net/demos> link.



**Just a minute:**

Which one of the following methods is used to draw the outline of the path on the canvas?

1. `fill()`
2. `stroke()`
3. `lineTo()`
4. `moveTo()`

**Answer:**

2. `stroke()`



## Activity 6.1: Introducing Canvas

# Transforming and Animating Canvas Elements

LearnGraphs Ltd. wants to make the graphics visually appealing to the users. For this, they need to animate or transform the shapes, such as scaling or rotating the line drawn on the canvas. To apply such animations, you need to apply animation effects on the shapes. With a canvas, you can transform, rotate, or scale graphic objects.

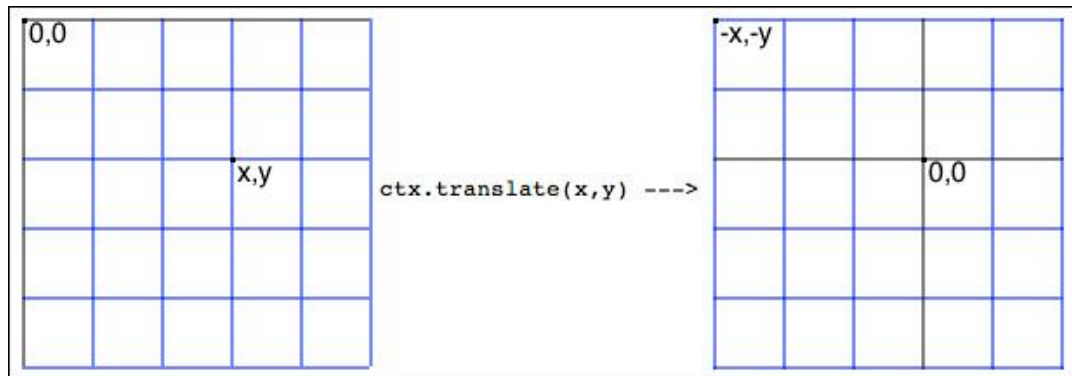
## Transforming Canvas Elements

With CSS, you can move HTML elements from one position to another easily. However, while working with a canvas, you may want to shift the graphic objects from one position to another and increase or decrease their size. This can be done by transforming the canvas elements. To transform the canvas elements, you can use the following methods:

- `translate()`
- `scale()`
- `rotate()`

### Translate

The `translate()` method is used to reset the origin of the canvas to the specified position, as shown in the following figure.



*The Resetting of the Origin of the Canvas*

In the preceding figure, the origin of the canvas is shifted to the (x, y) coordinate. The `translate()` method enables you to move all the graphic objects on the canvas by using just one method. For example, you have drawn a complex drawing on the canvas and you need to move the drawing around on the canvas. To perform such a task, you need to adjust the x and y positions of all the graphic objects involved in the drawing. It is a time-consuming and an error-prone task. You can simplify this task by just translating the context or the origin of the canvas to the new position and then, using the original x and y positions of all the graphic objects, redraw them on the canvas. This way, the graphic objects will be redrawn by taking into account the position of the new origin. Therefore, the graphic objects will be moved to the desired location on the canvas by using just one method call.

The syntax of the `translate()` method is:

```
translate(x,y);
```

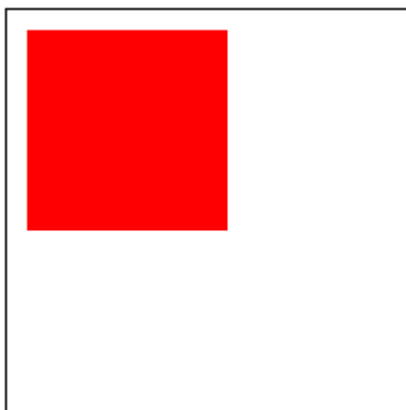
In the preceding syntax:

- `x`: Specifies the value to be added to the existing value of the x coordinate.
- `y`: Specifies the value to be added to the existing value of the y coordinate.

For example, you have created a rectangle by using the following code snippet:

```
var c=document.getElementById("myCanvas");  
var ctx=c.getContext("2d");  
ctx.fillStyle="red";  
ctx.fillRect(10,10,100,100);
```

The preceding code snippet will draw a rectangle at the coordinates, (10, 10), as shown in the following figure.



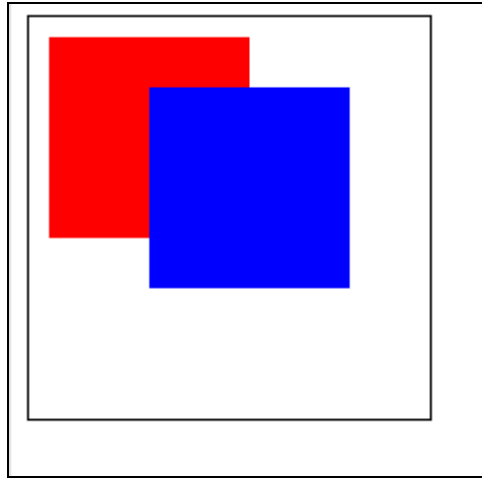
*The Rectangle*

Now, you want that the rectangle should be redrawn starting from the coordinates, (60, 35). For this, you can use the `translate()` method, as shown in the following code snippet:

```
var c=document.getElementById("myCanvas");  
var ctx=c.getContext("2d");  
ctx.fillStyle="red";  
ctx.fillRect(10,10,100,100);  
ctx.translate(50,25);  
ctx.fillStyle="blue";  
ctx.fillRect(10,10,100,100);
```

In the preceding code snippet, the first rectangle is drawn in the red color at the point, (10, 10), from the origin of the canvas. The `translate()` method then resets the origin of the canvas at the point, (50, 25). The second rectangle is drawn in the blue color at the point, (10, 10), from the new origin, (50, 25).

Therefore, the second rectangle will be drawn at the point, (60, 35), from the point, (0, 0), of the canvas, as shown in the following figure.



*The Output Derived by Using the translate() Method*

## Scale

The `scale()` method is used to increase or decrease the units in the canvas grid. This will allow you to draw scaled down or enlarged graphic objects. The following syntax can be used to scale the graphic objects:

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

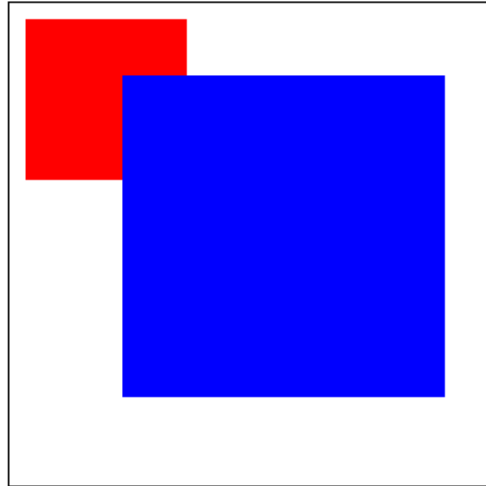
In the preceding syntax:

- `scalewidth`: Specifies the width (in percentage), a graphic object should be scaled to.
- `scaleheight`: Specifies the length (in percentage), a graphic object should be scaled to.

Consider the following code for scaling a rectangle on a canvas:

```
<!DOCTYPE HTML>
<HTML>
  <BODY>
    <CANVAS ID="myCanvas"    width="300" height="300" style="border:1px solid
    black">
    </CANVAS>
    <SCRIPT>
      var c=document.getElementById("myCanvas");
      var ctx=c.getContext("2d");
      ctx.fillStyle="red";
      ctx.fillRect(10,10,100,100);
      ctx.translate(50,25);
      ctx.scale(2,2);
      ctx.fillStyle="blue";
      ctx.fillRect(10,10,100,100);
    </SCRIPT>
  </BODY>
</HTML>
```

In the preceding code, the red colored rectangle of size 100 x 100 is created starting from the canvas coordinates, (10, 10). The origin of the canvas is reset at the point, (50, 25), by using the `translate()` method. Then, the width and height of the rectangle are scaled by the factor of 2 by using the `scale()` method. Therefore, the width and height of the blue colored rectangle gets doubled and is redrawn from the canvas coordinates, (60, 35), as shown in the following figure.



*The Output Derived by Using the `scale()` Method*

## Rotate

The `rotate()` method is used to rotate a graphic object to a specified degree in the clockwise direction. The following syntax can be used to apply rotation:

```
rotate(angle);
```

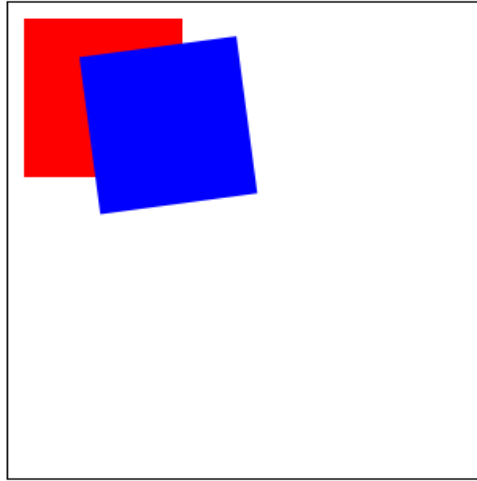
In the preceding syntax, `angle` specifies the degree a graphic object should be rotated to.

Consider the following code for applying rotations on a canvas:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<CANVAS ID="myCanvas"    width="300" height="300" style="border:1px solid
black">
</CANVAS>
<SCRIPT>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="red";
ctx.fillRect(10,10,100,100);
ctx.rotate(25);
ctx.fillStyle="blue";
ctx.fillRect(40,40,100,100);
</SCRIPT>
</BODY>
</HTML>
```



In the preceding code, a red colored rectangle is drawn at the coordinates, (10, 10). Further, the angle to rotate the canvas element is specified as 25. Therefore, the rectangle will be redrawn in the blue color starting from the coordinates, (40, 40), and rotated at the 25 degree angle, as shown in the following figure.



*The Output Derived by Using the rotate() Method*



### ***Just a minute:***

*Which one of the following methods is used to increase the size of a graphic element?*

1. `translate()`
2. `scale()`
3. `rotate()`
4. `transform()`

### ***Answer:***

2. `scale()`



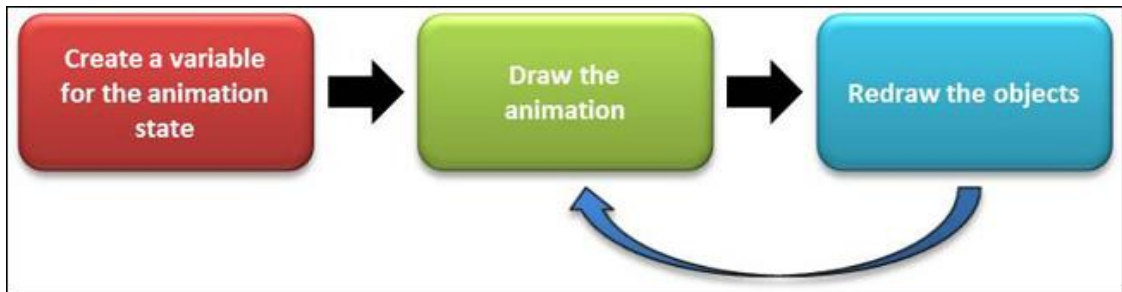
## **Animating Canvas Elements**

An animation is a visual technique that provides the illusion that an object is moving by displaying graphic objects in a rapid sequence. In an animation, the object is drawn first, next the canvas is cleared, and then the frame is drawn again with slight changes in its properties. The process is repeated very fast to create an illusion. For example, you want to represent a moving ball on the canvas. You have applied a single rotation on a ball that shows transformation. However, an animation is created if you apply multiple rotations on that ball.

To create such animations, you need to perform the following steps:

1. Create variables for the animation state.
2. Draw the animation.
3. Redraw the objects.

The following figure depicts the process of creating animations on the canvas.



*The Process of Creating Animations on the Canvas*

## Creating Variables for the Animation State

Variables are created to store the initial properties of the graphic objects, such as width or height. Further, they can be used to store the updated values of the graphic objects. Therefore, whenever a new position is calculated for a graphic object, the new values will get stored in the variables.

Consider an example of creating a ball moving on a canvas surface. To create such an animation, you need to define the variables first. Consider the following code snippet for defining variables in JavaScript:

```
var canvas;  
var ctx;  
var x = 400;  
var y = 300;  
var dx = 2;  
var dy = 4;  
var WIDTH = 400;  
var HEIGHT = 300;
```

In the preceding code snippet, the variables have been created that can be used to draw an animation on the canvas.

## Drawing the Animation

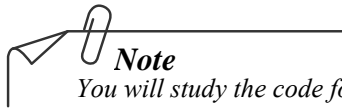
After creating the variables, you need to draw the graphic object on the screen. Consider the following code snippet for drawing a ball on the canvas:

```
function circle() {  
    ctx.beginPath();  
    ctx.fillStyle="red";  
    ctx.arc(x, y, 10, 0, Math.PI*2, true);  
    ctx.fill();  
    ctx.closePath();  
}
```

```
function clear() {
    ctx.clearRect(0, 0, WIDTH, HEIGHT);
}
function init() {
    canvas = document.getElementById("canvas");
    ctx = canvas.getContext("2d");
    return setInterval(draw, 10);
}
```

The preceding code snippet defines the following functions:

- **init()** function: In this function, the element ID for the canvas element is stored in the variable, `canvas`. Further, the context for the canvas element is retrieved in the variable, `ctx`. The context, `ctx`, can now be used to draw the graphic objects on the canvas. It is the first function that will be called in the code. In the next line of the code, the `setInterval()` function is used to call the `draw()` function every 10 milliseconds. The `draw()` function is a user-defined function that is created to redraw a ball on the canvas.



### Note

*You will study the code for the `draw()` function in the next topic.*

- **circle()** function: In this function, the `beginpath()` method is called to start a new path. The `arc()` method is used to define the size and shape of the circle. Further, the `fill()` method is used to fill the entire circle with the specified color.
- **clear()** function: In this function, the `clearRect()` method is called to erase the graphic objects from the canvas.

## Redrawing the Objects

Now, to create the animation effects, you need to repeatedly keep drawing the graphic objects with the updated properties.

Consider the following code snippet for redrawing a ball on the canvas:

```
function draw() {
    clear();
    circle();
    if (x > WIDTH || x < 0)
        dx = -dx;
    if (y > HEIGHT || y < 0)
        dy = -dy;

    x += dx;
    y += dy;
}
```

In the preceding code snippet, the `clear()` function is first called to clear the canvas. Further, the `circle()` function is called to create a circle on the canvas.

The circle has the radius, 10, and its origin is at (x,y). To move the circle, you need to change the values of the variables, x and y. Whenever the `draw()` function is executed, the variables, dx and dy, will determine the values for the variables, x and y. If the value of x is not greater than the width of the canvas or if it is less than zero, the value of x will get changed by dx. However, if the value exceeds the width size, the variable, dx, will be set to -dx, and the process continues. The same condition is applied for the variable, y.

The following lines show the entire code for rotating a ball:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>Rotating Ball</TITLE>
</HEAD>
<BODY>
<DIV>
<CANVAS ID="canvas" width="400" height="300" style="border:2px solid black">
</CANVAS>
</DIV>

<SCRIPT type="text/javascript">
var canvas;
var ctx;
var x = 400;
var y = 300;
var dx = 2;
var dy = 4;
var WIDTH = 400;
var HEIGHT = 300;

function circle() {
    ctx.beginPath();
    ctx.fillStyle="red";
    ctx.arc(x, y, 10, 0, Math.PI*2, true);
    ctx.fill();
    ctx.closePath();
}

function clear() {
    ctx.clearRect(0, 0, WIDTH, HEIGHT);
}

function init() {
    canvas = document.getElementById("canvas");
    ctx = canvas.getContext("2d");
    return setInterval(draw, 10);
}

function draw() {
    clear();
    circle();
    if (x > WIDTH || x < 0)
        dx = -dx;
    if (y > HEIGHT || y < 0)
        dy = -dy;
```

```
    x += dx;  
    y += dy;  
}
```

```
init();  
</SCRIPT>  
</BODY>  
</HTML>
```

▪

▪

## Practice Questions

1. Which one of the following path properties is used to create a path from the current position back to the starting position?
  - a. `beginPath()`
  - b. `closePath()`
  - c. `lineTo()`
  - d. `moveTo()`
2. Identify the correct syntax to apply a radial gradient on a canvas.
  - a. `createRadialGradient(x0,y0,r0);`
  - b. `createRadialGradient(x0,y0,x1,y1);`
  - c. `createRadialGradient(r0,r1);`
  - d. `createRadialGradient(x0,y0,r0,x1,y1,r1);`
3. Which one of the following methods is used to reset the origin of the canvas to the specified position?
  - a. `rotate()`
  - b. `translate()`
  - c. `scale()`
  - d. `fill()`
4. Which one of the following methods is used to return a drawing context object on the canvas?
  - a. `getContext()`
  - b. `getElementById()`
  - c. `write()`
  - d. `writeln()`
5. Which one of the following methods is used to specify the colors in a gradient object?
  - a. `colorStop()`
  - b. `addColorStop()`
  - c. `addColor()`
  - d. `color()`

## Summary

In this chapter, you learned that:

- Canvas provides an easy and a powerful way to create graphics on a Web page.
- A canvas is defined by using the `<CANVAS>` tag. This tag is defined in the body section of the HTML document.
- Defining the canvas element only creates a blank drawing surface. However, to actually draw graphic objects on the canvas, you need to access the canvas in the JavaScript code.
- You can use the following methods to draw shapes on canvas:
  - `rect()`
  - `fillRect()`
  - `strokeRect()`
  - `clearRect()`
- The following properties can be used to apply colors on the canvas objects:
  - `fillStyle`
  - `strokeStyle`
  - `shadowColor`
- Apart from creating simple shapes on the canvas, you can also apply styles, such as gradients, on them.
- A path is a series of points joined together to create lines or shapes. In a canvas, you can use lines or paths to draw shapes other than rectangles or squares.
- In addition to drawing shapes or lines on the canvas, you can also draw text on it. You can also apply different styles on the text.
- The `drawImage()` method is used to draw an image or a video on the canvas.
- In a canvas, you can create graphs, such as bar graph or pie chart, to represent relationships.
- To transform the canvas elements, you can use the following methods:
  - `translate()`
  - `scale()`
  - `rotate()`
- To create such animations, you need to perform the following steps:
  - Create a variable for the animation state.
  - Draw the animation.
  - Redraw the objects.

