

## Unit: 4

### HTML5- CANVAS

#### Canvas Introduction:

- The HTML5 canvas element can be used to draw graphics on the webpage via JavaScript.
- The canvas was originally introduced by Apple for the Mac OS dashboard widgets and to power graphics in the Safari web browser.
- Later it was adopted by the Firefox, Google Chrome and Opera.
- Now the canvas is a part of the new HTML5 specification for next generation web technologies.
- By default the <canvas> element has 300px of width and 150px of height without any border and content.
- However, custom width and height can be defined using the CSS **height** and **width** property whereas the border can be applied using the CSS **border** property.
- In layman term, Canvas means **Painting**.
- In **HTML5**, you can draw a circle, line, text, sphere, arc, polygon, **gradient**, etc on canvas.
- The HTML Canvas is meaningful only if you use **JavaScript**, so if you do not have a basic knowledge of JavaScript, it will not hold any importance.

#### How do you use Canvas?

The <canvas> tag is used for this purpose.

Few Web designers feel they can only use Canvas for Graphics but this is completely wrong. You can use Canvas for multiple purposes like –

- canvas is used to create graphics
- use this when you want a pixel-perfect solution
- To draw the text faster
- We can use it to animate
- And we can also create HTML Games.

The default size of the canvas is 300 \* 150 px without any border or content.



**Example of a canvas with default width and height:**

```
<canvas style="border:2px solid blue;"></canvas>
```

**Note/Info:**

1. In the above example, there is no id attribute but in a practical scenario, each canvas must have an id because JavaScript will be using this id to draw the graphics.
2. If you are using Chrome browser or any other modern browser, you can easily find out which JavaScript files are used by the canvas to draw the graphics. Just use the developer mode(Inspect). Just right click on the browser and click on 'inspect' or type F12 on your browser.

So, how do you start drawing canvas?

There are some important points which you should keep in mind while dealing with Canvas –

- You should know at least the basics of *JavaScript* to master the Canvas. If you do not have knowledge of JavaScript, we recommend you to learn the basics of JavaScript.
- Understand about the Horizontal axis(X-axis) and the Vertical axis(y-axis)
- Learn about the control points, Starting Points, End Points, etc.

- You should know all the important **functions** which help us to draw a line, a circle, arc, text, image, etc on canvas. We will show you all the important functions related to Canvas as well.

### Canvas Drawing:

There are multiple ways of drawing on the canvas and this depends on what you want to actually draw.

For Example, you can draw –

- Rectangle
- Line
- Circle
- Quadratic curve or Arc
- Polygon
- Gradient and a lot more

### Understanding Canvas Coordinates:

- ✓ The canvas is a two-dimensional rectangular area.
- ✓ The coordinates of the top-left corner of the canvas are (0, 0) which is known as origin, and the coordinates of the bottom-right corner are (*canvas width*, *canvas height*).
- ✓ Here's a simple demonstration of canvas default coordinate system.
- ✓ The <canvas> element is supported in all major web browsers such as Chrome, Firefox, Safari, Opera, IE 9 and above.



- HTML5 element `<canvas>` gives you an easy and powerful way to draw graphics using JavaScript.
- It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.
- Here is a simple `<canvas>` element which has only two specific attributes **width** and **height** plus all the core HTML5 attributes like id, name and class, etc.

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

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

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

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

```
<!DOCTYPE HTML>
```

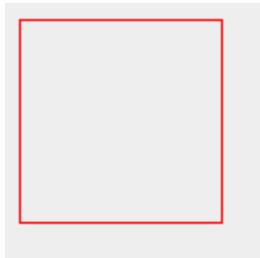
```
<html>
```

```
<head>
```

```
<style>
  #mycanvas{border:1px solid red;}
</style>
</head>

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

This will produce the following result –



## The Rendering Context:

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

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

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

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

if (canvas.getContext) {
  var ctx = canvas.getContext('2d');
  // drawing code here
} else {
```

```
// canvas-unsupported code here  
}
```






## Browser Support:

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

You can use Explorer Canvas to have canvas support through Internet Explorer. You just need to include this JavaScript as follows –

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

## Supporting Browsers

Element	 Chrome	 IE	 Firefox	 Opera	 Safari
<canvas>	Yes	Yes	Yes	Yes	Yes

## Canvas Examples:

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

**Note:** Always specify an **id** attribute (to be referred to in a script), and a **width** and **height** attribute to define the size of the canvas.

To add a border, use the **style** attribute.

Here is an example of a basic, empty canvas:



## Example

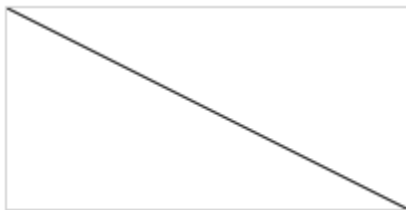
```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid  
#000000;">  
</canvas>
```

## Add a JavaScript:

After creating the rectangular canvas area, you must add a JavaScript to do the drawing.

Here are some examples:

### Draw a Line



## Example

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.moveTo(0, 0);  
ctx.lineTo(200, 100);
```

```
ctx.stroke();  
</script>
```

## 1. Draw rectangle in canvas:

To draw a rectangle, you need to consider these steps-

1. Select the Canvas using its element like **id**. To achieve this, you can use the **getElementById()** method.
2. As canvas is a 2d design, you need to draw a 2d object. You can use the **getContext()** method for this purpose.
3. You need to actually draw the design inside the canvas. Use the below bullet points to understand this –
  - You need to select a style like color, gradient, etc to the object, you can use the **fillStyle** property. If you do not explicitly define the **fillStyle** color, it will be black by default.
  - You should tell the browser about the Horizontal axis offset(**x-offset**) from *left*, Vertical axis offset(**y-offset**) from **top**, **width**, & **height** using the **fillRect** property.

### Note/Info:

The functions are case sensitive means 'fillStyle' function is correct name while 'fillstyle' is wrong function name.

In case of rectangle, it is important to draw the style first (using fillStyle property) before drawing the rectangle(using fillRect property).



getElementById() and getContext() functions will be set for almost all the canvas drawings.

### **Example of a basic canvas Drawing:**

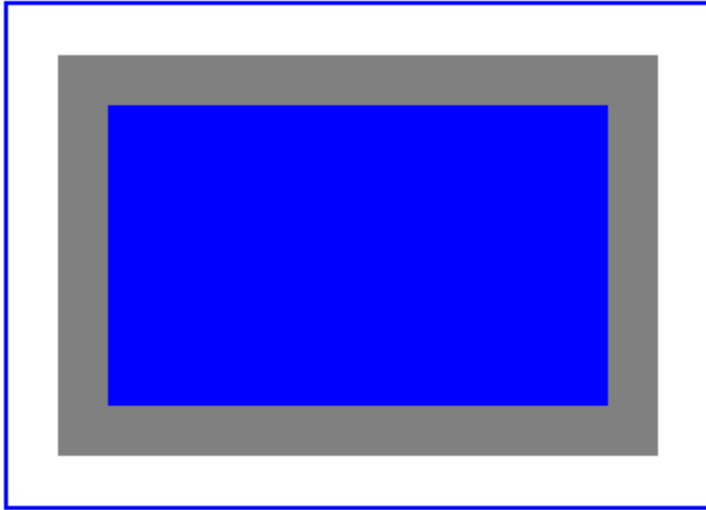
```
<script>
var getId = document.getElementById('canvas-drawing');
var getIdContext = getId.getContext('2d');
getIdContext.fillStyle = 'grey';
getIdContext.fillRect(25,25,300,200);
</script>
```

### **Canvas draw rectangle fill**

Now, suppose you want to create another drawing with blue color inside the canvas, then you can further add –

```
getIdContext.fillStyle = 'blue';
getIdContext.fillRect(50,50,250,150);
```

so, the Canvas along with the drawing will look like this –



## Drawing a Rectangle:

You can create rectangle and square shapes using the `rect()` method. This method requires four parameters x, y position of the rectangle and its width and height.

The basic syntax of the `rect()` method can be given with:

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

The following JavaScript code will draw a rectangle shape centered on the canvas.

### *Example*

[Try this code »](#)

```
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        context.stroke();
    };
</script>
```

There are three methods that draw rectangles on the canvas –

Sr.No.	Method and Description
1	<b>fillRect(x,y,width,height)</b> This method draws a filled rectangle.
2	<b>strokeRect(x,y,width,height)</b> This method draws a rectangular outline.
3	<b>clearRect(x,y,width,height)</b> This method clears the specified area and makes it fully transparent

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

## Canvas draw Line:

To draw Line –

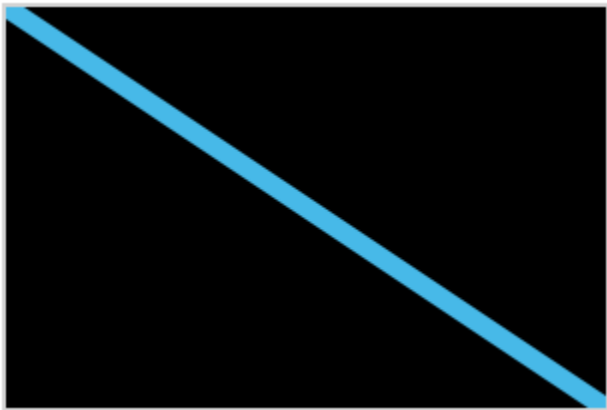
- First, you need to include **getElementById()** & **getContext()** properties.
- Next, you need to use **beginPath()** function to begin the path but this is optional now so you can remove it.
- The next 2 important functions are **moveTo(start-x-offset, start-y-offset)** & **lineTo(end-x-offset, end-y-offset)**.
- Other optional properties are **lineWidth**, **strokeStyle** or **lineCap** properties and you can use them depending on your requirements. **strokeStyle** can have any color,

gradient or pattern while **lineCap** can have any one of these values – butt, square, rounded.

- And, the last one is **stroke()** function & this is mandatory to draw the line.

#### Example - Draw lines to draw a basic email

```
var getId1 = document.getElementById('canvas-text-no-color');  
var getIdContext1 = getId1.getContext('2d');  
getIdContext1.beginPath();  
getIdContext1.moveTo(10,20);  
getIdContext1.lineTo(120,180);  
getIdContext1.lineWidth = 10;  
getIdContext1.lineCap = "square";  
getIdContext1.strokeStyle = "blue";  
getIdContext1.stroke();
```



## Drawing a Line

The most basic path you can draw on canvas is a straight line. The most essential methods used for this purpose are `moveTo()`, `lineTo()` and the `stroke()`.

The `moveTo()` method defines the position of drawing cursor onto the canvas, whereas the `lineTo()` method used to define the coordinates of the line's end point, and finally the `stroke()` method is used to make the line visible. Let's try out an example:

### ***Example***

**Try this code »**

```
<script>
  window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.moveTo(50, 150);
    context.lineTo(250, 50);
    context.stroke();
  };
</script>
```

### **Line methods:**

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

Sr.No.	Method and Description
1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.

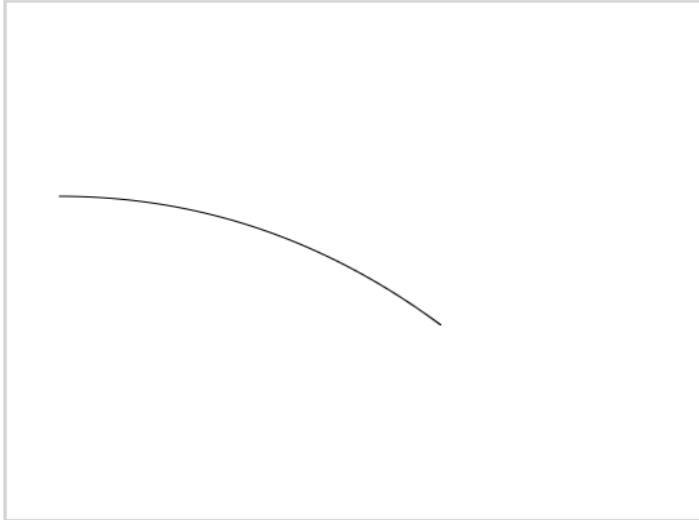
4	<b>fill()</b> This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>lineTo(x, y)</b> This method adds the given point to the current subpath, connected to the previous one by a straight line.

## 5. Draw Quadratic Curves or Arc on Canvas

To draw Quadratic Curve-

- **getElementById()** & **getContext()** are mandatory functions.
- **beginPath()** function is optional.
- use **moveTo(start-x-offset, start-y-offset)** where start-x-offset & start-y-offset are the starting points (x,y).
- The next important function is **quadraticCurveTo(control-point-x, control-point-y, end-x-point,end-y-point)**

- **strokeStyle** = “color-name” is optional to set a color for the border(stroke)
- And, the last one is **stroke()** function & this is mandatory to draw the circle.



**Example - Draw circular flags of different countries**

```
var getId1 = document.getElementById('canvas-draw-circle');  
var getIdContext1 = getId1.getContext('2d');  
getIdContext1.beginPath();  
getIdContext1.moveTo(30,150);  
getIdContext1.quadraticCurveTo(150,150,250, 250);  
getIdContext1.stroke();
```

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

S.No.	Method and Description
-------	------------------------

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

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

## Drawing a Arc

You can create arcs using the `arc()` method. The syntax of this method is as follow:

```
context.arc(centerX, centerY, radius, startingAngle, endingAngle, counterclockwise);
```

The JavaScript code in the following example will draw an arc on the canvas.

### *Example*

[Try this code »](#)

```
<script>
```



```
window.onload = function() {  
    var canvas = document.getElementById("myCanvas");  
    var context = canvas.getContext("2d");  
    context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);  
    context.stroke();  
};  
</script>
```

## Drawing Path and Shapes on Canvas

In this section we're going to take a closer look at how to draw basic paths and shapes using the newly introduced HTML5 canvas element and JavaScript.

Here is the base template for drawing paths and shapes onto the 2D HTML5 canvas.

Example:

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
<meta charset="utf-8">  
  
<title>Drawing on Canvas</title>  
  
<script>  
  
    window.onload = function() {  
  
        var canvas = document.getElementById("myCanvas");  
  
        var context = canvas.getContext("2d");  
  
        // draw stuff here  
  
    };  
}
```

```
</script>

</head>

<body>

    <canvas id="myCanvas" width="300" height="200"></canvas>

</body>

</html>
```

All the lines except those from 7 to 11 are pretty straight forward. The anonymous function attached to the `window.onload` event will execute when the page load. Once the page is loaded, we can access the canvas element with `document.getElementById()` method. Later we have defined a 2D canvas context by passing 2d into the `getContext()` method of the canvas object.

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

S.No.	Method and Description
1	<b>beginPath()</b> This method resets the current path.
2	<b>moveTo(x, y)</b> This method creates a new subpath with the given point.
3	<b>closePath()</b> This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	<b>fill()</b>

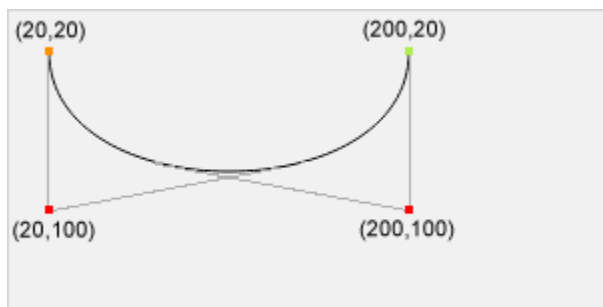
	This method fills the subpaths with the current fill style.
5	<b>stroke()</b> This method strokes the subpaths with the current stroke style.
6	<b>arc(x, y, radius, startAngle, endAngle, anticlockwise)</b> Adds points to the subpath such that the arc described by the circumference of the circle described by the arguments, starting at the given start angle and ending at the given end angle, going in the given direction, is added to the path, connected to the previous point by a straight line.

## HTML5 Canvas - Drawing Bezier Curves

### Definition and Usage

The `bezierCurveTo()` method adds a point to the current path by using the specified control points that represent a cubic Bézier curve.

A cubic bezier curve requires three points. The first two points are control points that are used in the cubic Bézier calculation and the last point is the ending point for the curve. The starting point for the curve is the last point in the current path. If a path does not exist, use the [beginPath\(\)](#) and [moveTo\(\)](#) methods to define a starting point.



Start point

`moveTo(20,20)`

Control point 1

```
bezierCurveTo(20,100,200,100,200,20)
```

Control point 2

```
bezierCurveTo(20,100,200,100,200,20)
```

End point

```
bezierCurveTo(20,100,200,100,200,20)
```

**Tip:** Check out the [quadraticCurveTo\(\)](#) method. It has one control point instead of two.

**JavaScript syntax:**

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

Parameter Values

Parameter	Description
<i>cp1x</i>	The x-coordinate of the first Bézier control point
<i>cp1y</i>	The y-coordinate of the first Bézier control point
<i>cp2x</i>	The x-coordinate of the second Bézier control point
<i>cp2y</i>	The y-coordinate of the second Bézier control point
<i>x</i>	The x-coordinate of the ending point
<i>y</i>	The y-coordinate of the ending point

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

S.No.	Method and Description
1	<b>beginPath()</b>

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

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