HTML 5 - Canvas.

The Rendering Context:-

* The CanvasRenderingContext2D interface, part of the Canvas API.
* Provides the 2D rendering context for the drawing surface of a <canvas> element.
* It is used for drawing shapes, text, images and other objects.
* The interfaces properties and methods are described in the reference section of the page.
* The Canvas tutorial has more explanation, examples and resources, as well.
* For OffscreenCanvas, there is an equivalent interface that provides the rendering context.

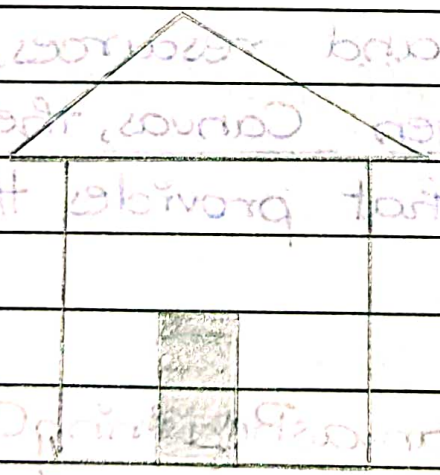
Ex:-

To get a CanvasRederingContext2D instance, you must first have an HTML <canvas> elements to work this:-

```
<canvas id="my-house" width="300"
    height="300"></canvas>
Const canvas = document.getElementById
    ("my-house");
```

```
const ctx=canvas.getContext("2d");
ctx.lineWidth=10;
ctx.strokeRect(75,140,150,110);
ctx.fillRect(130,190,40,60);
ctx.beginPath();
ctx.moveTo(50,140);
ctx.lineTo(150,60);
ctx.lineTo(250,140);
ctx.closePath();
ctx.stroke();
```

Output:-



Browser Support:-

*With the exception of Internet Explorer 8, HTML 5 Canvas is supported in some way b most modern web browsers, with specific

feature support growing on an almost daily basis.

*The best support seems to be from Google Chrome, followed closely by Safari, Firefox, and Opera.

*We will utilize a JavaScript library named modernizr.js that will help us figure out which browsers supports which Canvas features.

*At the same time, if you are worried about Internet Explorer, version 9 promises to have support for Canvas.

*In the meantime, you can check out Google Chrome Frame (http://code.google.com/chrome/chromeframe/), which delivers support for IE.

Canvas - Drawing Rectangles:-

*The HTML 5 <canvas> tag is used to draw graphics, animation, etc. using scripting.

*It is a new tag introduced in HTML 5.

*The canvas has a DOM method called getContext, which obtains rendering context and its drawing functions.

*This functions takes one parameter, the type of context 2d.
*To draw a rectangle with HTML 5 canvas, use the fillRect(x, y, width, height) method:

Syntax:-

ctx.fillRect(x, y, width, height)

x- The x-coordinate (in pixels), the upper-left corner of the rectangle in relation to the coordinates of the canvas.
y- The y-coordinate (in pixels), the upper-left corner of the rectangle in relation to the coordinates of the canvas.
width- The width (in pixels), of the rectangle.
height- The height (in pixels), of the rectangle.

Ex:-

```
<!DOCTYPE html>
<html>
<head>
 <title>Canvas Rectangle</title>
    <style>
```

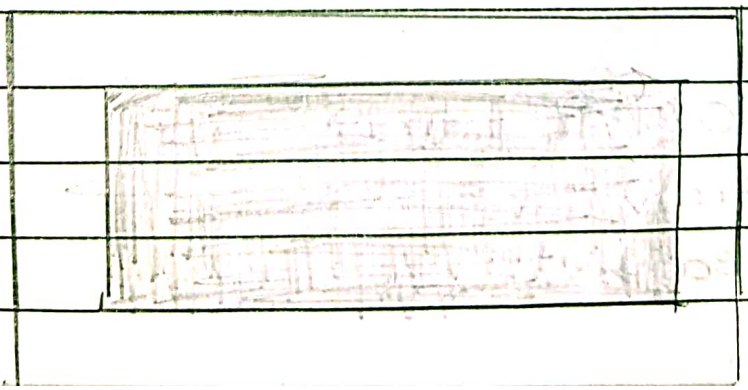
```
canvas {
    border: 1px solid black;
}
</style>
</head>
<body>
<canvas id="myCanvas" width="200"
        height="200"></canvas>

<script>
var canvas = document.getElementById
             ("myCanvas");
var ctx = canvas.getContext("2d");

ctx.fillStyle = "red"; or '#fa4b2a'
ctx.fillRect(50,50,100,100); or (10,40,140,160);
                             or (20,20,150,100);
</script>
</body>
</html>
```

JavaScript

Output:-



Canvas - Drawing Paths:-

* Paths are a sequence of lines or curves that can be used to create complex shapes on the canvas.
* The canvas API provides methods to define and manipulate paths:

1. beginPath():- Starts a new path or resets the current path.
2. moveTo(x, y):- Moves the current drawing position to the specified point.
3. lineTo(x, y):- Draws a straight line from the current position to the specified point.
4. arc(x, y, radius, startAngle, endAngle, anticlockwise):- Draws an Arc with the given parameters.
5. closePath():- Connects the last point of the path to the starting point, creating a closed shape.

Ex:-

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

```
<title>Canvas Drawing Paths</title>
<style>
  canvas{
    border:1px solid black;
  }
</style>
</head>
<body>
    <canvas id="myCanvas" width="500"
            height="500"></canvas>

<script>
const canvas=document.getElementById
                ("myCanvas");
const ctx = canvas.getContext("2d");

ctx.beginPath();
ctx.moveTo(50,50);
ctx.lineTo(150,50);
ctx.LineTo(150,150);
 ctx.closePath();
 ctx.stroke();
</script>
</body>
</html>
```

Canvas Drawing - Lines:-

* The lineTo() method adds a new point and create a line To that point FROM the last specified point in the canvas (this method does not draw the line).
* First, create a new line by calling the beginPath() method.
* Second, move the drawing cursor to the point(x, y) without drawing a line by calling the moveTo(x, y).
* Finally, draw a line from the previous point to the point(x, y) by calling the lineTo(x, y) method.

Example:-

```
<!DOCTYPE html>
<html>
<head>
 <title> Canvas Drawing Line </title>
 <style>
  canvas {
      border: 1px solid black;
      }
</style>
```

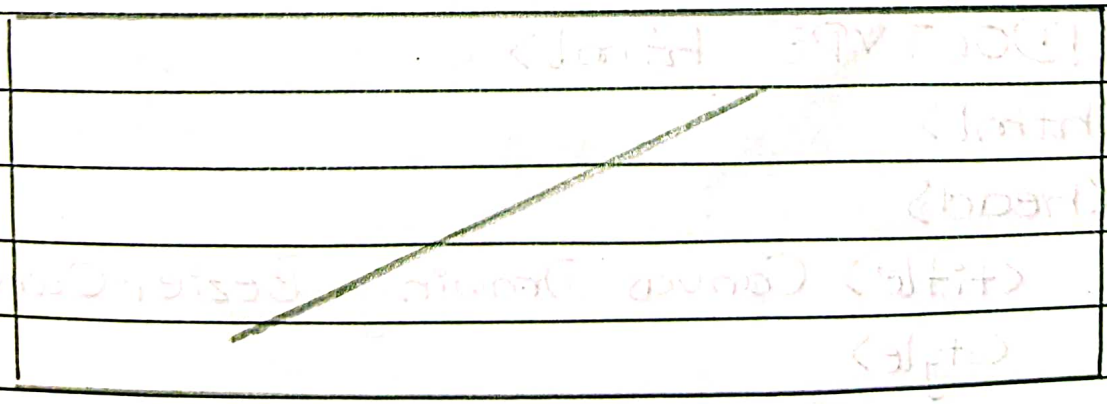
```
</head>
<body>
  <canvas id = "myCanvas" width = "200"
    height = "200" ></canvas>

  <script>
  const canvas : document.getElementById
                  ("myCanvas");
  const ctx = canvas.getContext("2d");

  ctx.beginPath();
  ctx.moveTo(50, 50);
  ctx.lineTo(150, 150);
  ctx.stroke();

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

Output:-



Canvas - Drawing Bezier Curves:-

* The bezierCurveTo() method adds a point to the current path by using the specified control points that represent a cubic Bezier Curve.
* A cubic bezier curve requires three points.
* The first two points are control points tha are used in the cubic Bezier calculation and the last point is the ending point for the curve.
* The starting point for the curve is the las point ~~to~~ in the current path.
* If a path does not exist, use the beginPath() and moveTo() methods to define a starting point.

Example:-

```
<!DOCTYPE html>
<html>
<head>
    <title> Canvas Drawing BezierCurve </title>
    <style>
```

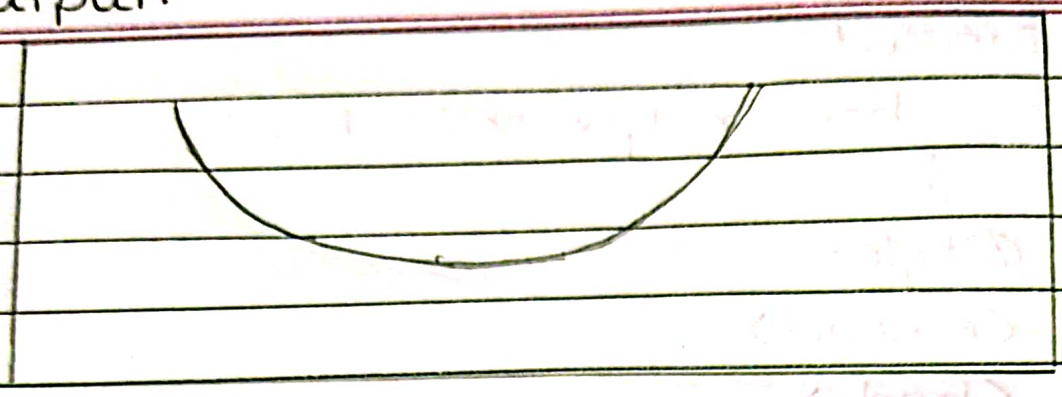
```
canvas {
    border: 1px solid black;
  }
</style>
</head>
<body>
  <canvas id="myCanvas" weigth="200"
          height="200"> </canvas>

<script>
const canvas=document.getElementById
            ("myCanvas");
const ctx = canvas.getContext("2d");

ctx.beginPath();
ctx.moveTo(20,20);
ctx.bezierCurveTo(20,100,200,100,200,20);
ctx.stroke();

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

Output:-



Canvas-Drawing Quadratic Curve:-

* The quadraticCurveTo() method adds a point to the current path by using the specified control points that represent a quadratic Bezier curve.
* A quadratic Bezier curve requires two points.
* The first point is a control point that is used in the quadratic Bezier calculation and the second 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.

Output:-

Canvas - Using Images:-

*Canvas is an HTML element that allows for dynamic, scriptable rendering of 2D and 3D graphics.
*Images can be used to draw on a canvas using the drawImage() method in JavaScript.
*This method takes an image object as its first argument and can also accept additional arguments to specify the position, width and height of the image on the canvas.
*By using images, developers can create visually appealing graphics and animations on a canvas.
*Additionally, images can be manipulated using various canvas methods such as scaling, rotating and cropping to create dynamic effects.
* Using images on a canvas provides a

powerful tool for creating interactive web applications with rich visuals.

Ex:-

```
<!DOCTYPE html>
<html>
<head>
    <title>Canvas Image Example</title>
    <style>
        canvas {
            border: 1px solid black;
        }
    </style>
    </head>
<body>
    <canvas id="myCanvas" width="400"
        height="400"></canvas>

<script>
    const canvas = document.getElementById
                    ("myCanvas");
    const ctx = canvas.getContext("2d");

    const image = new Image();
    image.src = "path/to/image.jpg";
```

```
image.onload = function(){
    ctx.drawImage(image, 50, 50);
};

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

Canvas - Create Gradients :-

* Gradients can be used to fill rectangles, circles, lines, text etc. Shapes on the canvas are not limited, to solid colors.
* Gradients can be created on the canvas to fill shapes with color transitions.

Two types of Gradients :-

1. Linear Gradients.
2. Radial Gradients.

1. Linear Gradients :-

* A Linear Gradient is a gradient that goes from one color to another in a straight line.
* Linear gradients are created using the

createLinearGradient(x0, y0, x1, y1) method, where (x0, y0) and (x1, y1) define the start and end points of the gradient.

2. Radial Gradients:-

* A radial gradient is a gradient that goes from one color to another in a circular or elliptical shape.
* Radial Gradients are created using the createRadialGradient(x0, y0, r0, x1, y1, r1) method, where (x0, y0) and (x1, y1) define the center points of the start and end circles, and r0 and r1 define then radii.

Ex:- Linear Gradients:-

```
<!DOCTYPE html>
<html>
<head>
    <title> Canvas Gradients Example </title>
 <style>
    canvas{
        border: 1px solid black;
    }
 </style>
 </head>
```

```
<body>
  <canvas id = "myCanvas" width= "400"
    height= "400" ></canvas>

<script>
const canvas=document.getElementById
                ("myCanvas");
const ctx = canvas.getContext("2d");

//Creating a Linear Gradient
const linearGradient= ctx.creatLinearGradi
                (0, 0, 200, 0);
linearGradient.addColorStop( 0,"red");
linearGradient.addColorStop(1, "blue");

ctx.fillStyle = linearGradient;
ctx.fillRect(50, 50, 200, 100);

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

```
const ctx = canvas.getContext("2d");

//Creating a Radial Gradient
const radialGradient = ctx.createRadial
                -Gradient(150,150,50,150,150,100);
radialGradient.addColorStop(0, "green");
radialGradient.addColorStop(1, "yellow");

ctx.fillStyle = radialGradient;
ctx.beginPath();
ctx.arc(150,150,100,0,2*Math.PI);
ctx.fill();

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

Output:-