**Introduction to HTML5 Canvas**

Html Canvas is just an HTML Element. We can start working on canvas by just creating an canvas element in HTML file.

<canvas></canvas>  
***<!—Default background color of canvas element is white because of the default color of the document. -->***

So, this is what canvas space where only we can interact there. We’ve to start styling our canvas to make it visible.  
canvas{  
border:1px solid black;  
}

**Four Essential skills for any HTML5 Canvas Piece:**

1. Creating and Resizing Your Canvas.
2. Drawing Elements
3. Animating Elements
4. Interacting with Elements
5. **Creating and Resizing Your Canvas**

Since, our html tag of the entire document doesn’t actually take up the full height of the screen. So, We can’t make our canvas as full and Height using CSS. Because this is going to take the full height of the html tag.

So in order to alleviate this, we’re not going to set our height and width through CSS, We’re actually going to set our height and width through our JavaScript using the **innerWidth & innerHeight** property of Window Object. So that we can assure that the height and width of the canvas will always be the entire height and width of the full browser window.

let canvas = document.querySelector('canvas');

canvas.width = innerWidth; ***//innerWidth is the property of Window Object.***

canvas.height = innerHeight; ***//innerHeight is the property of Window Object.***

1. **Drawing Elements**

Firstly, we’ve to Get the reference of 2D canvas **context**, where we'll be returning a drawing context in a variable called context variable.

This Basically means we're creating an Object, passing tons of methods and function which we can use to actually draw within our canvas.

Here, actually we’re going to draw something on a 2d space. Ex: Square, Circle etc.  
We can’t draw any 3d model something like boxes but we can draw our 2d elements that can be manipulated within a 2d space.

**Example:  
//Getting the reference of 2D canvas context, where we'll be returning a drawing context in variable 'c'.**

let c = canvas.getContext('2d');   
**// Basically we're creating an Object, passing tons of methods and function which we can use to actulaly draw within our canvas**

Once we do that, we then have access to all their properties and methods or functions which we can draw on the screen width within this variable called C.

**What Objects we can draw with the canvas:**

1. Rectangles
2. Lines
3. Arcs (which we can use to also create circles)
4. Bezier Curves
5. Images
6. Text

For the sake of time, We’re going to cover those Objects which are widely used: Rectangles, Lines & Arcs.

1. Rectangles 🡪 **fillRect()** method and **fillStyle** Property

It takes four arguments. (x value, y value, width & height).  
fillRect(x, y, width, height).

Here,X & Y are going to determine where on the screen this rectangle is going to be and the width and height is going determine the width and height of the rectangle.

Value of X & Y going to relative from the top-left of the screen of the coordinate system.

**Example:  
//Getting the reference of 2D canvas context, where we'll be returning a drawing context in variable 'c'.**

let c = canvas.getContext('2d');   
**// Basically we're creating an Object, passing tons of methods and function which we can use to actulaly draw within our canvas**

c.fillStyle="rgba(255,0,0,0.1)";   
***//Using CSS Styling to the style rectangle with the help fillStyle property***c.fillRect(100, 100, 100, 100);  
   
c.fillStyle="rgba(0,255,0,0.1)";   
c.fillRect(400, 300, 100, 100);  
  
c.fillStyle="rgba(0,0,255,0.1)";   
c.fillRect(100, 300, 100, 100);   
  
c.fillStyle="#13f4ff";   
c.fillRect(100, 300, 100, 100);

2). Lines 🡪 **beginPath(), moveTo() & lineTo()** method and **strokeStyle** Property

**Example:**c.beginPath(); ***//This basically indicate “a point” for canvas saying, Okay we want to start a path.***

c.moveTo(50,300) ***//Now we're declaring where on the canvas we want our path to start by passing X & Y co-ordinate for positioning***

**// Note: This would be a point, so it will be invisible until we call a stroke() method. As soon as we call method then we'll start to see a line from a point to point**

c.lineTo(300,100); ***// lineTo method will create another new point.*** c.lineTo(300,200);  
c.lineTo(400,300);  
c.lineTo(200,300);

c.strokeStyle = "red"; ***//Using CSS Styling to the style stroke line with the help strokeStyle property***

c.stroke(); ***//To display line or say to build connection between points.***

3). ARC 🡪

Using arcs to we can create circles. An arc method takes few argument. It takes (x-value(Int), y-value(Int), radius(Int), startAngle(Float), endAngle(Float), drawCounterClockwise (Boolean)).

**startAngle & endAngle** properties doesn’t take angle in degrees. They takes **radians**.  
startAngle property says at what angle property says that would we like to start drawing our arc while the end angle property says how long would we like the arc to go on for.

**drawCounterClockwise:** It just specifies which direction should the arc actually be drawn in. It takes Boolean value either true or false. For false it doesn’t going to create counter clockwise direction. By default anticlockwise.

**By doing all these things, we actually have created an outline foran arc. (So it will not be visible by just doing this).**

To make it visible we need to fill this outline using **stroke() method or fill property**.

**Example:**

c.beginPath(); **//We want to make sure that we're preceding any arc or any line with this beginPath() method because it's going to seprate the two and prevent them from connecting to each other.**

c.arc(300,400,30,0,Math.PI\*2,false);

c.strokeStyle = 'blue';

c.stroke();

**Start from 10:00 Video-2 (Draw the canvas)**