**Unit V**

**HTML5 - Styles and Colors**

**HTML5 - Styles and Colors, Canvas - Text and Fonts, Canvas - Pattern and Shadow, Canvas - Save and Restore States, Canvas - Translation, Canvas - Rotation, Canvas - Scaling, Canvas - Transforms, HTML5 Canvas - Composition, Canvas – Animations.**

**5.1 Canvas - Styles and Colors**

HTML5 canvas provides the following two important properties to apply colors to a shape.

|  |  |
| --- | --- |
| **Method** | **Description** |
| fillStyle | This attribute represents the color or style to use inside the shapes. |
| strokeStyle | This attribute represents the color or style to use for the lines around shapes. |

By default, the stroke and fill color are set to black. The ***fillStyle*** property sets or returns the color, gradient, or pattern used to fill the drawing. The default value is #000000 (solid black). It takes the general format,

**context.fillStyle = color|gradient|pattern**

where

color -> CSS color value that indicates the fill color of the drawing. It takes the default value black

gradient -->linear or radial gradient object used to fill the drawing

pattern --> to fill the pattern

**Example 5.1 Fill style Example - gradient**

[Live Demo](http://tpcg.io/OrKbYy)

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The createLinearGradient() Method</h2>

<canvas id="myCanvas" width="300" height="150" style="border:3px solid blue"></canvas>

<script>

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

// Create Gradient

const my\_gradient = ctx.createLinearGradient(0, 0, 0, 170);

my\_gradient.addColorStop(0, "yellow");

my\_gradient.addColorStop(1, "red");

// Fill Rectangle

ctx.fillStyle = my\_gradient;

ctx.fillRect(20, 20, 150, 100);

</script>

</body>

</html>

**5.2 Canvas - Text and Fonts**

To draw text on a canvas, the most important property and methods are

* font - defines the font properties for the text
* fillText(*text,x,y*) - draws "filled" text on the canvas
* strokeText(*text,x,y*) - draws text on the canvas (no fill)

**Example 5.2 Stroke text example**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

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

<script>

const canvas = document.getElementById("myCanvas");

constctx = canvas.getContext("2d");

ctx.font = "30px Arial";

ctx.strokeText("BCA IV SEM",10,50);

</script>

</body>

</html>

**5.3 Canvas - Pattern and Shadow**

**Canvas Pattern**

The createPattern() method repeats the specified element in the specified direction.

The element can be an image, video, or another <canvas> element.The repeated element can be used to draw/fill rectangles, circles, lines etc. It takes the general format,

***createPattern(image, repetition)***

where

image -->Specifies the image, canvas, or video element of the pattern to use

repetition-->It may take the values repeat, repeat-x, repeaty, andno-repeat. If the empty string or null is specified, repeat will be assumed

|  |  |
| --- | --- |
| Repeat | Default. The pattern repeats both horizontally and vertically |
| repeat-x | The pattern repeats only horizontally |
| repeat-y | The pattern repeats only vertically |
| no-repeat | The pattern will be displayed only once (no repeat) |

**Example 5.3 Canvas pattern example**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The createPattern() Method</h2>

<p>Image to use:</p>

<imgsrc="star.png" id="myimg" width="320" height="320">

<p>

<button onclick="draw('repeat')">Repeat</button>

<button onclick="draw('repeat-x')">Repeat-x</button>

<button onclick="draw('repeat-y')">Repeat-y</button>

<button onclick="draw('no-repeat')">No-repeat</button>

</p>

<canvas id="myCanvas" width="3000" height="1500"/>

<script>

function draw(direction)

{

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

ctx.clearRect(0, 0, c.width, c.height);

const img = document.getElementById("myimg")

const pat = ctx.createPattern(img, direction);

ctx.rect(0, 0, 1500, 2000);

ctx.fillStyle = pat;

ctx.fill();

}

</script>

</body>

</html>

**Canvas Shadow**

HTML5 canvas provides capabilities to create nice shadows around the drawings. All drawing operations are affected by the four global shadow attributes.

|  |  |
| --- | --- |
| **Property** | **Description** |
| shadowColor [ = value ] | This property returns the current shadow color and can be set, to change the shadow color. |
| shadowOffsetX [ = value ] | This property returns the current shadow offset X and can be set, to change the shadow offset X. |
| shadowOffsetY [ = value ] | This property returns the current shadow offset Y and can be set, change the shadow offset Y. |
| shadowBlur [ = value ] | This property returns the current level of blur applied to shadows and can be set, to change the blur level. |

**Example 5.4 Shadow color property**

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The shadowColor Property</h2>

<canvas id="myCanvas" width="300" height="150" style="border:3px solid green"></canvas>

<script>

const c = document.getElementById("myCanvas");

constctx = c.getContext("2d");

ctx.shadowBlur = 20;

ctx.fillStyle = "green";

ctx.shadowColor = "yellow";

ctx.fillRect(20, 20, 100, 80);

ctx.shadowColor = "red";

ctx.fillRect(140, 20, 100, 80);

</script>

</body>

</html>

**Example 5.5 Shadow offset property**

The shadowOffsetX property sets or returns the horizontal distance of the shadow from the shape.

Default value is 0.

|  |  |
| --- | --- |
| Value | Description |
| 0 | The shadow is right behind the shape |
| 20 | The shadow starts 20 pixels right from the shape's left position |
| -20 | The shadow starts 20 pixels left from the shape's left position |

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The shadowOffsetX Property</h2>

<canvas id="myCanvas" width="300" height="150" style="border:2px solid green"></canvas>

<script>

const c = document.getElementById("myCanvas");

constctx = c.getContext("2d");

ctx.shadowBlur = 10;

ctx.shadowOffsetX = 20;

ctx.shadowColor = "green";

ctx.fillStyle = "cyan";

ctx.fillRect(20, 20, 100, 80);

</script>

</body>

</html>

**Example 5.6 Shadow Blur property**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The shadowBlur Property</h2>

<canvas id="myCanvas" width="300" height="150" style="border:2px solid yellow"></canvas>

<script>

const c = document.getElementById("myCanvas");

constctx = c.getContext("2d");

ctx.shadowBlur = 50;

ctx.shadowColor = "blue";

ctx.fillStyle = "cyan";

ctx.fillRect(20, 20, 100, 80);

</script>

</body>

</html>

**5.4 Canvas - Save and Restore States**

HTML5 canvas provides two important methods to save and restore the canvas states. The canvas drawing state is basically a snapshot of all the styles and transformations that have been applied and consists of the followings −

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

Canvas states are stored on a stack every time the **save** method is called, and the last saved state is returned from the stack every time the **restore** method is called.

|  |  |
| --- | --- |
| **Method** | **Description** |
| save() | This method pushes the current state onto the stack.. |
| restore() | This method pops the top state on the stack, restoring the context to that state. |

**Example 5.7 Save and restore method**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>save and restore() Method</h2>

<canvas id="myCanvas" width="300" height="150" style="border:1px solid grey"></canvas>

<script>

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

// Draw

ctx.fillStyle = "green";

ctx.fillRect(10, 10, 50, 50);

// Save the state

ctx.save();

// Draw new

ctx.fillStyle = "red";

ctx.fillRect(100, 10, 50, 50);

// Restore saved state

ctx.restore();

// Draw new

ctx.fillRect(200, 10, 50, 50);

</script>

</body>

</html>

**5.5 Canvas - Translation**

The translate() method remaps the (0,0) position of the context. When we call a new method after translate(), the new positions are added to the x and y coordinates. It takes the general format,

***context.translate(*x, y*)***

where

x🡪 Value to add to horizontal (x) coordinate

y🡪 Value to add to vertical (y) coordinate

**Example 5.8 Canvas translate method Example**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The translate() Method</h2>

<canvas id="myCanvas" width="300" height="150" style="border:1px solid grey"></canvas>

<script>

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

ctx.fillRect(10, 10, 100, 50);

ctx.translate(70, 70);

ctx.fillRect(10, 10, 100, 50);

</script>

</body>

</html>

**5.6 Canvas - Rotation**

The rotate() method rotates the context. The rotation will affect drawings after the rotation method is called. It takes the general format,

***context*.rotate(*angle*)**

where,

angle 🡪 rotation angle in radians  
Radians = degrees\*Math.PI/180.

For example,  
To rotate 5 degrees: 5\*Math.PI/180

**Example 5.9 Canvas Rotation method**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The rotate() Method</h2>

<canvas id="myCanvas" width="300" height="150" style="border:3px solid red"></canvas>

<script>

const c = document.getElementById("myCanvas");

constctx = c.getContext("2d");

ctx.rotate(20 \* Math.PI / 180);

ctx.fillRect(50, 20, 100, 50);

</script>

</body>

</html>

**5.7 Canvas - Scaling**

The scale() method scales the current context. If we scale a context, all future drawings will be scaled. If we scale(2,2), drawings will be positioned twice as far from the 0,0 position of the canvas as we specify. It takes the general format,

***context.scale(*scalewidth, scaleheight*)***

where,

*scalewidth🡪* Scales the width (1=100%, 0.5=50%, 2=200%)

*scaleheight🡪* Scales the width (1=100%, 0.5=50%, 2=200%)

**Example 5.10 Save and restore method**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The scale() Method</h2>

<canvas id="myCanvas" width="300" height="170" style="border:1px solid grey"></canvas>

<script>

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

ctx.strokeRect(5, 5, 25, 15);

ctx.scale(2, 2);

ctx.strokeRect(5, 5, 25, 15);

ctx.scale(2, 2);

ctx.strokeRect(5, 5, 25, 15);

ctx.scale(2, 2);

ctx.strokeRect(5, 5, 25, 15);

</script>

</body>

</html>

**5.8 Canvas - Transforms**

transform() method multiplies the current transformation with the matrix described by the arguments of this method. This let us to scale, rotate, translate(move) and skew the context.

HTML5 canvas provides methods which allow modifications directly to the transformation matrix. The transformation matrix must initially be the identity transform. It may then be adjusted using the transformation methods.

|  |  |
| --- | --- |
| **Method** | **Description** |
| transform(a, b, c, d, e, f) | This method changes the transformation matrix to apply the matrix given by the arguments. |
| setTransform(a, b, c, d, e, f) | This method changes the transformation matrix to the matrix given by the arguments. |

The transform(a, b, c, d, e, f) method must multiply the current transformation matrix with the matrix described by −

a c e

b d f

0 0 1

where,

a- denotes horizontal scaling

b- denotes horizontal skewing

c- denotes vertical skewing

d- denotes vertical scaling

e- denotes horizontal moving

f- demotes vertical moving

The *setTransform*(a, b, c, d, e, f) method must reset the current transform to the identity matrix, and then invoke the *transform*(a, b, c, d, e, f) method with the same arguments.

**Example 5.11 canvas composition example**

<!DOCTYPE html>

<html>

<body>

<canvas id="estpt" width="400" height="400" style="border:3px solid blue;">></canvas>

<script>

// Get the canvas element

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

var context = canvas.getContext("2d");

// Translate

context.translate(10, 10);

// Draw the original square

context.fillStyle = "red";

context.fillRect(0, 0, 100, 100);

// Reset the transformation matrix

context.setTransform(1.5, 0, 0, 1, 0, 0);

// Rotate

context.translate(200, 10);

context.rotate((Math.PI / 180) \* 45);

// Draw the rotated square

context.fillStyle = "green";

context.fillRect(0, 0, 100, 100);

</script>

</body>

</html>

**5.9 HTML5 Canvas - Composition**

HTML5 canvas provides compositing attribute **globalCompositeOperation** which affect all the drawing operations.

We can draw new shapes behind existing shapes and mask off certain areas, clear sections from the canvas using globalCompositeOperation attribute as shown below in the example.

There are following values which can be set for globalCompositeOperation

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| source-over | This is the default setting and draws new shapes on top of the existing canvas content. |
| source-in | The new shape is drawn only where both the new shape and the destination canvas overlap. Everything else is made transparent. |
| source-out | The new shape is drawn where it doesn't overlap the existing canvas content. |
| source-atop | The new shape is only drawn where it overlaps the existing canvas content. |
| lighter | Where both shapes overlap the color is determined by adding color values. |
| xor | Shapes are made transparent where both overlap and drawn normal everywhere else. |
| destination-over | New shapes are drawn behind the existing canvas content. |
| destination-in | The existing canvas content is kept where both the new shape and existing canvas content overlap. Everything else is made transparent. |
| destination-out | The existing content is kept where it doesn't overlap the new shape. |
| destination-atop | The existing canvas is only kept where it overlaps the new shape. The new shape is drawn behind the canvas content. |
| darker | Where both shapes overlap the color is determined by subtracting color values. |

**Example 5.12 canvas composition example**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The globalCompositeOperation Property</h2>

<canvas id="myCanvas" width="300" height="150" style="border:3px solid blue"></canvas>

<script>

const c = document.getElementById("myCanvas");

constctx = c.getContext("2d");

ctx.fillStyle = "red";

ctx.fillRect(20, 20, 75, 50);

ctx.fillStyle = "blue";

ctx.globalCompositeOperation = "source-over";

ctx.fillRect(50, 50, 75, 50);

ctx.fillStyle = "red";

ctx.fillRect(150, 20, 75, 50);

ctx.fillStyle = "blue";

ctx.globalCompositeOperation = "destination-over";

ctx.fillRect(180, 50, 75, 50);

</script>

</body>

</html>

**5.10 Canvas – Animations**

HTML5 canvas provides necessary methods to draw an image and erase it completely. We can take Javascript help to simulate good animation over a HTML5 canvas.

Following are the two important Javascript methods which would be used to animate an image on a canvas

|  |  |
| --- | --- |
| **Method** | **Description** |
| setInterval(callback, time) | This method repeatedly executes the supplied code after a given *time*milliseconds. |
| setTimeout(callback, time) | This method executes the supplied code only once after a given *time* milliseconds. |