HTML Canvas Cheatsheet

A quick reference guide to the HTML Canvas API, covering basic setup, drawing shapes, working with images, applying styles and transformations, and creating animations.

Canvas Basics & Setup

Creating a Canvas Element

Add a `<canvas>` element to your HTML:

```
<canvas id="myCanvas" width="500" height="300"></canvas>
```

- `id`: A unique identifier for the canvas.
- `width`: The width of the canvas in pixels.
- `height`: The height of the canvas in pixels.

Accessing the canvas via JavaScript:

```
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');
```

- `getContext('2d')`: Gets the 2D rendering context.

Checking for Canvas Support:

```
if (canvas.getContext) {
  // Canvas is supported!
} else {
  // Canvas is not supported, provide a fallback.
}
```

Basic Canvas Properties

`canvas.width`	Gets or sets the width of the canvas (in pixels).
`canvas.height`	Gets or sets the height of the canvas (in pixels).
`ctx.fillStyle`	Sets the color, gradient, or pattern used to fill shapes. Example: `'red'`, `'#FF0000'`, `'rgba(255, 0, 0, 0.5)'`.
`ctx.strokeStyle`	Sets the color, gradient, or pattern used to stroke lines and shapes. Example: `'blue'`, `'#0000FF'`.
`ctx.lineWidth`	Sets the width of lines in pixels. Example: `5`.

Drawing Shapes

Rectangles

`ctx.fillRect(x, y, width, height)` - Draws a filled rectangle.

```
ctx.fillStyle = 'green';
ctx.fillRect(20, 20, 150, 100);
```

`ctx.strokeRect(x, y, width, height)` - Draws a rectangle outline.

```
ctx.strokeStyle = 'blue';
ctx.lineWidth = 5;
ctx.strokeRect(20, 20, 150, 100);
```

`ctx.clearRect(x, y, width, height)` - Clears a rectangular area, making it transparent.

```
ctx.clearRect(40, 40, 110, 60);
```

Paths (Lines, Curves, Arcs)

`ctx.beginPath()` - Starts a new path.
`ctx.closePath()` - Closes the current path by drawing a straight line back to the start.
`ctx.stroke()` - Strokes (outlines) the path with the current stroke style.
`ctx.fill()` - Fills the path with the current fill style.

`ctx.moveTo(x, y)` - Moves the starting point of a new subpath to the specified coordinates.
`ctx.lineTo(x, y)` - Adds a straight line from the current point to the specified coordinates.

```
ctx.beginPath();
ctx.moveTo(50, 50);
ctx.lineTo(200, 50);
ctx.lineTo(200, 150);
ctx.stroke();
```

`ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise)` - Creates an arc/circle.

- `x`, `y`: Center of the arc.
- `radius`: Radius of the arc.
- `startAngle`, `endAngle`: Start and end angles in radians.
- `anticlockwise`: `true` for drawing the arc anticlockwise, `false` for clockwise (default).

```
ctx.beginPath();
ctx.arc(100, 100, 50, 0, Math.PI * 2);
ctx.fill();
```

`ctx.quadraticCurveTo(cpX, cpY, endX, endY)` - Creates a quadratic Bézier curve.

- `cpX`, `cpY`: Coordinates of the control point.
- `endX`, `endY`: Coordinates of the end point.
  `ctx.bezierCurveTo(cp1X, cp1Y, cp2X, cp2Y, endX, endY)` - Creates a cubic Bézier curve.
- `cp1X`, `cp1Y`: Coordinates of the first control point.
- `cp2X`, `cp2Y`: Coordinates of the second control point.
- `endX`, `endY`: Coordinates of the end point.

Working with Images and Text

Drawing Images

`ctx.drawImage(image, dx, dy)` - Draws an image onto the canvas.

- `image`: An `<img>`, `<canvas>`, or `<video>` element.
- `dx`, `dy`: The x and y coordinates in the destination canvas at which to place the top-left corner of the image.

```
const img = new Image();
img.src = 'myImage.jpg';
img.onload = () => {
  ctx.drawImage(img, 0, 0);
};
```

`ctx.drawImage(image, dx, dy, dWidth, dHeight)` - Draws an image and scales it to fit the destination rectangle.

- `dWidth`: The width to scale the image to in the destination canvas.
- `dHeight`: The height to scale the image to in the destination canvas.

`ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)` - Draws a section of an image and scales it to fit the destination rectangle.

- `sx`, `sy`: The x and y coordinates of the top-left corner of the sub-rectangle of the source image to draw into the destination canvas.
- `sWidth`, `sHeight`: The width and height of the sub-rectangle of the source image to draw into the destination canvas.

Adding Text

`ctx.fillText(text, x, y)` - Draws filled text on the canvas.

- `text`: The text string to render.
- `x`, `y`: The x and y coordinates of the bottom-left corner of the text.

```
ctx.font = '30px Arial';
ctx.fillText('Hello Canvas', 50, 50);
```

`ctx.strokeText(text, x, y)` - Draws stroked (outlined) text on the canvas.

```
ctx.font = '30px Arial';
ctx.strokeText('Hello Canvas', 50, 50);
```

`ctx.font` - Specifies the font properties for text. Example: `'20px sans-serif'`.
`ctx.textAlign` - Specifies the horizontal alignment of text. Values: `'left'`, `'right'`, `'center'`, `'start'`, `'end'`.
`ctx.textBaseline` - Specifies the vertical alignment of text. Values: `'top'`, `'hanging'`, `'middle'`, `'alphabetic'`, `'ideographic'`, `'bottom'`.

Transformations and Compositing

Transformations

`ctx.translate(x, y)` - Moves the canvas origin to (x, y).

```
ctx.translate(50, 50);
ctx.fillRect(0, 0, 100, 100); // Draws at (50, 50)
```

`ctx.rotate(angle)` - Rotates the canvas by `angle` radians.

```
ctx.rotate(Math.PI / 4); // Rotate 45 degrees
ctx.fillRect(0, 0, 100, 100);
```

`ctx.scale(x, y)` - Scales the canvas by factors `x` and `y`.

```
ctx.scale(2, 0.5);
ctx.fillRect(0, 0, 50, 50); // Draws a rectangle twice as wide and half as tall
```

`ctx.transform(a, b, c, d, e, f)` - Multiplies the current transformation matrix with the matrix described by the arguments.
`ctx.setTransform(a, b, c, d, e, f)` - Resets the current transform to the identity matrix and then invokes `transform()` with the given arguments.

- `a` (m11): Horizontal scaling.
- `b` (m12): Horizontal skewing.
- `c` (m21): Vertical skewing.
- `d` (m22): Vertical scaling.
- `e` (dx): Horizontal moving.
- `f` (dy): Vertical moving.

`ctx.resetTransform()` - Resets the current transform to the identity matrix.

Global Composite Operations

`ctx.globalCompositeOperation = type` - Specifies how new shapes should be drawn on top of existing canvas content.

Common values:

- `source-over` (default): New shapes are drawn on top of the existing canvas content.
- `source-atop`: New shapes are drawn only where they overlap existing canvas content.
- `source-in`: New shapes are drawn only where they overlap existing canvas content. Existing content is cleared outside the new shape.
- `source-out`: New shapes are drawn where they do not overlap existing canvas content.
- `destination-over`: Existing canvas content is drawn on top of the new shapes.
- `destination-atop`: Existing canvas content is drawn only where it overlaps new shapes. New shapes are cleared outside the existing content.
- `destination-in`: Existing canvas content is drawn only where it overlaps new shapes. New shapes are cleared outside the existing content.
- `destination-out`: Existing canvas content is drawn where it does not overlap new shapes.
- `lighter`: New shapes are added to the existing canvas content.
- `copy`: New shapes replace the existing canvas content.
- `xor`: New shapes are drawn where they do not overlap existing canvas content, and existing content is cleared where it overlaps the new shape.
- `multiply`: Multiplies the colors of the new shapes with the colors of the existing canvas content.
- `screen`: Inverts the colors of the new shapes and the existing canvas content, multiplies them, and then inverts the result.
- `overlay`: Combines the multiply and screen operations.
- `darken`: Keeps the darker of the new shapes and the existing canvas content.
- `lighten`: Keeps the lighter of the new shapes and the existing canvas content.
- `color-dodge`: Divides the colors of the existing canvas content by the colors of the new shapes.
- `color-burn`: Divides the colors of the new shapes by the colors of the existing canvas content.
- `hard-light`: Combines the multiply and screen operations, but with the new shapes and the existing canvas content swapped.
- `soft-light`: Similar to hard-light, but with a softer effect.
- `difference`: Subtracts the colors of the new shapes from the colors of the existing canvas content.
- `exclusion`: Similar to difference, but with a more intense effect.
- `hue`: Preserves the hue of the existing canvas content and replaces the saturation and luminosity with the hue of the new shapes.
- `saturation`: Preserves the saturation of the existing canvas content and replaces the hue and luminosity with the saturation of the new shapes.
- `color`: Preserves the hue and saturation of the existing canvas content and replaces the luminosity with the color of the new shapes.
- `luminosity`: Preserves the luminosity of the existing canvas content and replaces the hue and saturation with the luminosity of the new shapes.