Meet the Transformation Methods

WE'LL COVER THE FOLLOWING

The three methods you have for transforming your `canvas` are `translate`, `scale`, and `rotate`. In the following sections, let's look at how to use these methods.

Translating

If you want to shift your `canvas` and everything that gets drawn, you have the the `translate` method:

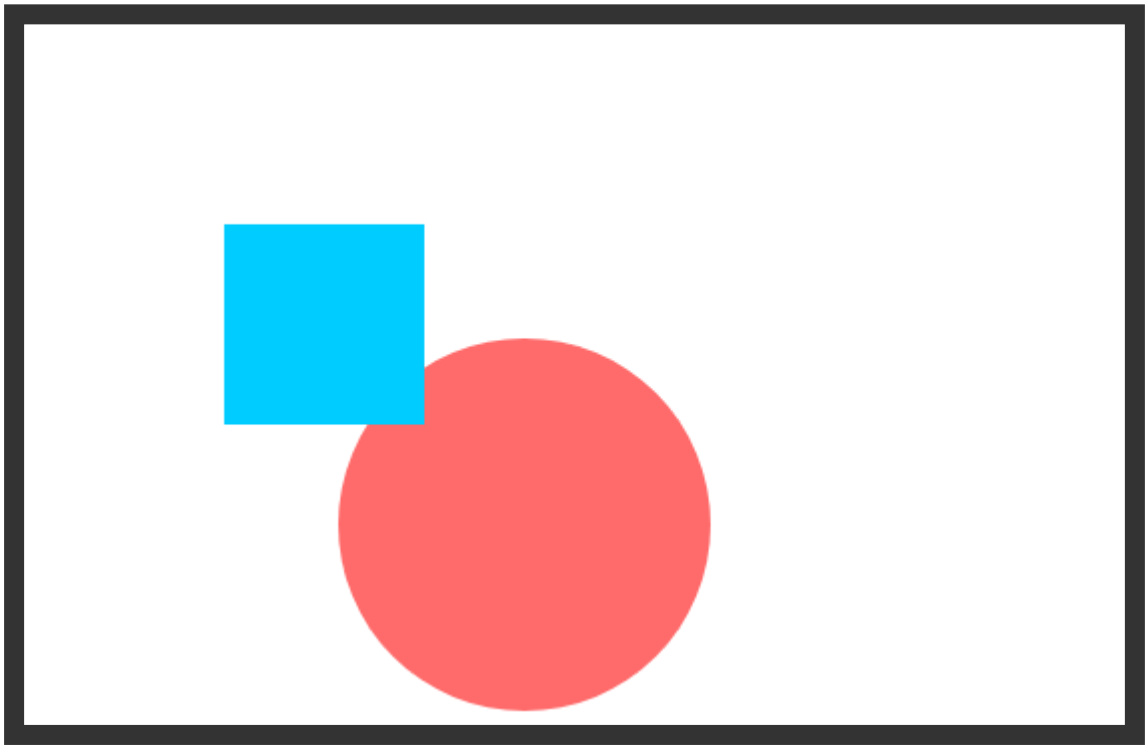
```
context.translate(x, y);
```

The x and y arguments specify the number of pixels to shift your canvas horizontally and vertically by. Below is a simple example of what this looks like:

HTML JavaScript

```javascript
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

// Transform
context.translate(50, 50);

// Circle
context.beginPath();
context.arc(200, 200, 93, 0, 2 * Math.PI, true);
```

```
context.fillStyle = "#FF6A6A";
context.fill();

// Square
context.fillStyle = "#00CCFF";
context.fillRect(50, 50, 100, 100);

```



This above code draws a circle and a square to our `canvas`. The call to the `translate` method at the top shifts both of the shapes over by 50 pixels.

The entire `canvas` and the origin (0, 0) position is shifted, **so all future drawing operations will have their positions offset automatically**. Having a transform apply to all draw operations from here on out may be undesirable, and we'll look at how to address that in a little bit. Just ignore this minor annoyance for now.

Rotating

This is probably my favorite transform, for rotating the things you draw is really hard using the drawing commands we have available today. The way you rotate is by using the `rotate` method, and it looks as follows:

```
context.rotate(angle);
```

This method takes one argument that determines the angle (in the form of radians) you wish to rotate the `canvas` by. Here is an example of us rotating some text that we draw by 45 degrees:

HTML JavaScript

```javascript
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

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

// Text
context.font = "bold 48px Helvetica, Arial, sans-serif";
context.fillStyle = "steelblue";
context.fillText("Wheeeee!", 150, 0);
```

output

I chose a text example to highlight the **rotate** transform because text is one of the things you draw that is nearly impossible to re-create using rotated angled lines and curves. Without the `rotate` method, you'd be spending a lot of time trying to get a single character to look right - much less an entire word or a series of words! By comparison, rotating geometric shapes is much easier. With that said, you should still use the `rotate` method whenever you can

instead of rotating manually...like an animal.

Scaling

The last individual transform we will look at is the `scale` method that is responsible for scaling what you draw:

```
context.scale(x, y);
```

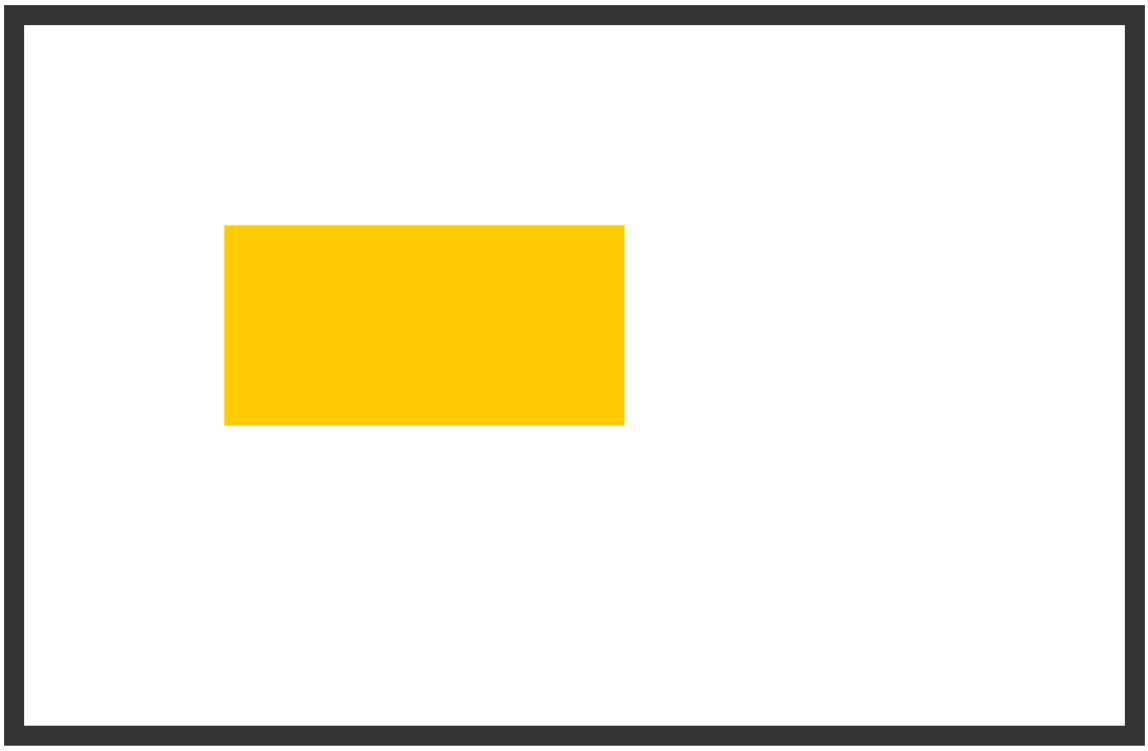
This method takes two arguments that specify the horizontal and vertical scale accordingly. You can specify the arguments in the form of decimal values with 1 representing the original scale. A number between 0 and 1 means that what you draw will be scaled down, and a number greater than 1 means that what you draw will be scaled up.

The following code highlights an example where a poor square is stretched horizontally to twice its size:

HTML JavaScript

javascript

```javascript
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

// Transform
context.scale(2, 1);

// Square
context.fillStyle = "#FFCC00";
context.fillRect(50, 100, 100, 100);
```



You can even specify negative values to flip our canvas horizontally or vertically. In the following code, we flip some text horizontally and scale it down by 50%:

HTML JavaScript

```javascript
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

// Transform
context.scale(-.5, 1);

// Text
context.font = "bold 96px Helvetica, Arial, sans-serif";
context.fillStyle = "#CC6699";
context.fillText("Confused", -700, 100);
```

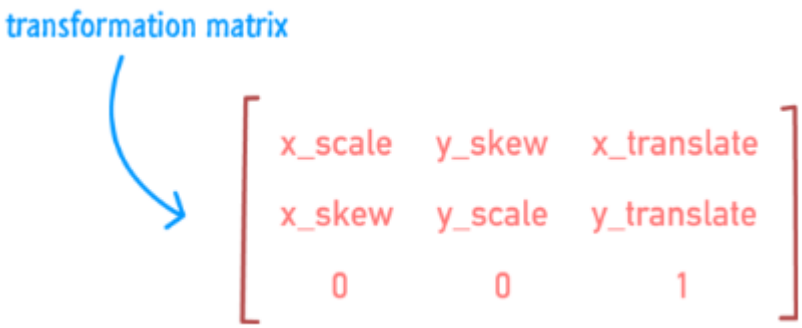
The negative value for the `scale` method's x argument flips our `canvas` horizontally. The value of .5 squishes things by 50%.

Combining Transforms

You aren't limited to using only a single transform to torture your canvas with. You can apply multiple transforms very easily:

```
context.scale(-.5, 1);
context.rotate(45 * Math.PI / 180);
context.translate(40, 10);
```

The reason this is possible has to do with how these transforms are implemented. There is a transform matrix that represents all of the transform values you can use:



> These values aren't dependent on any other values, so you can independently set multiple transforms without stepping on any numerical toes. Don't worry if that doesn't make any sense. Just remember that all the `translate`, `rotate`, and `scale` methods end up affecting are the values stored by this matrix. You can set this matrix directly by using the `setTransform` method, but covering that goes beyond the scope of what you would use frequently in the real world.