Undoing Transforms

WE'LL COVER THE FOLLOWING

This may be the part you have been eagerly waiting for. As you probably realized by now, transforming the `canvas` isn't an operation that resets itself with each thing you draw. It's not like a `fillStyle` or `strokeStyle`. The transformation is always there for any draw operation you perform in the future. That isn't always desirable, right?

To handle this, you need to explicitly turn the transforms off. There are several ways you can do with this. We'll look at two approaches in this section and focus on a slightly different (and heavy-handed) approach in a future tutorial where we look at how to save and restore state.

Resetting the Transform...the Easy Way

The easiest way to reset a transform is to call the `resetTransform` method:

HTML JavaScript

javascript

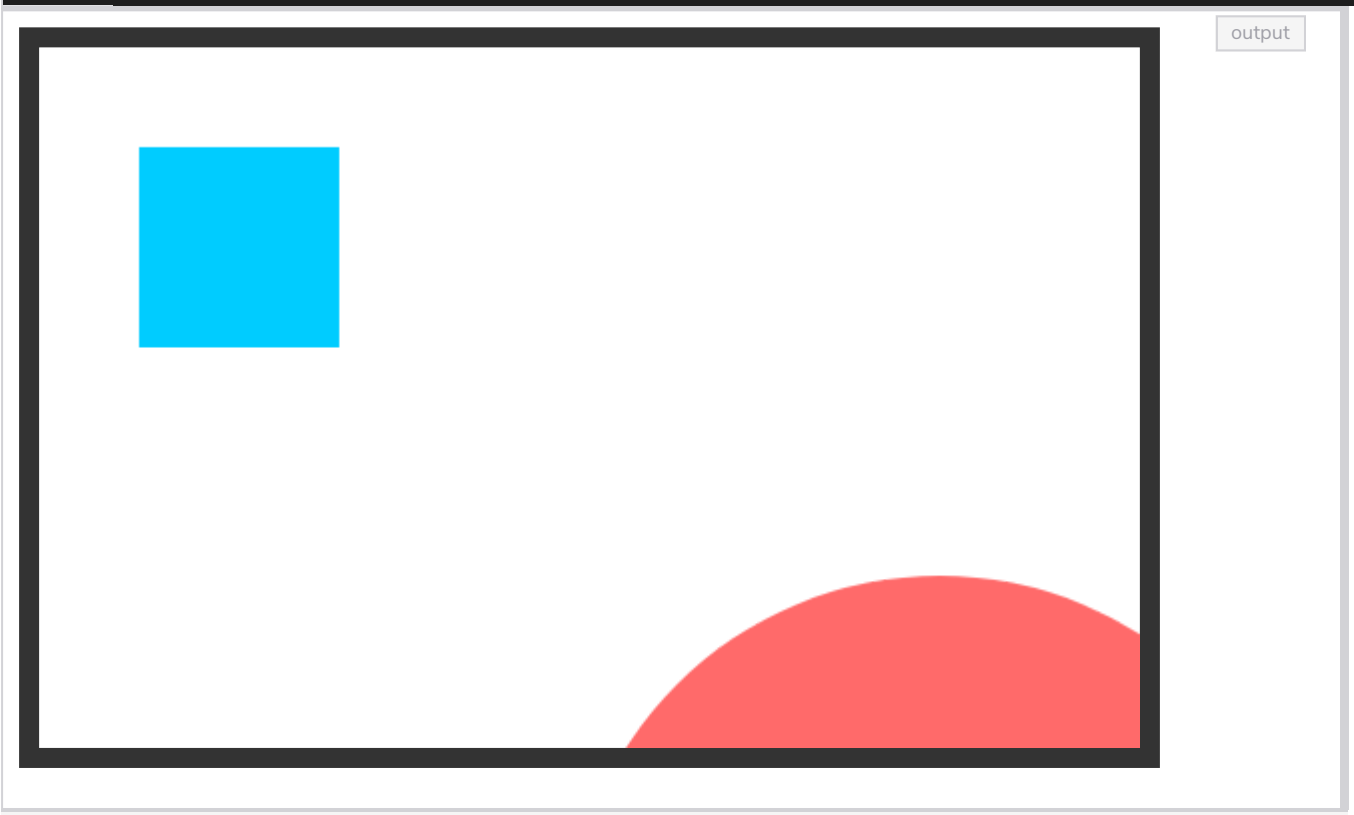
```
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

// Transform
context.translate(50, 50);
context.scale(2, 2);

// Circle
context.beginPath();
context.arc(200, 200, 93, 0, 2 * Math.PI, true);
context.fillStyle = '#FF6A6A';
context.fill();

// Reset the Transform
```

```
15  context.resetTransform();
16
17  // Square
18  context.fillStyle = '#00CCFF';
19  context.fillRect(50, 50, 100, 100);
20
```



The `resetTransform` method performs the magic needed to the transformation matrix you saw earlier to get everything back to how it was before a transform was even applied. In our example, the circle will be drawn on the transformed canvas. The square will be drawn on the untransformed canvas. Because of how drawing on the canvas works, untransforming the canvas with our circle already on it won't affect how the circle displays. Only **future** draw operations after `resetTransform` will be impacted.

Manually Resetting the Transform

> TL;DR: Just use `resetTransform`. Skip this section. Tell your friends.

Before we go on, I should mention this upfront: I don't recommend you reset the transform with using the approach I am about to show you. The only

reason I am showing you this is to give you a better understanding of how

transforms affect the `canvas`. Plus, it inflates the length of this article and helps make all of us look really smart by learning about this.

The more tedious way to reset your `canvas` to its untransformed state involves setting new transforms to undo what your earlier transforms did. That seems straightforward, but as you will see in a few seconds, there are some complications here that you'll need to deal with.

Here is an example of what this madness looks like:

HTML JavaScript

javascript

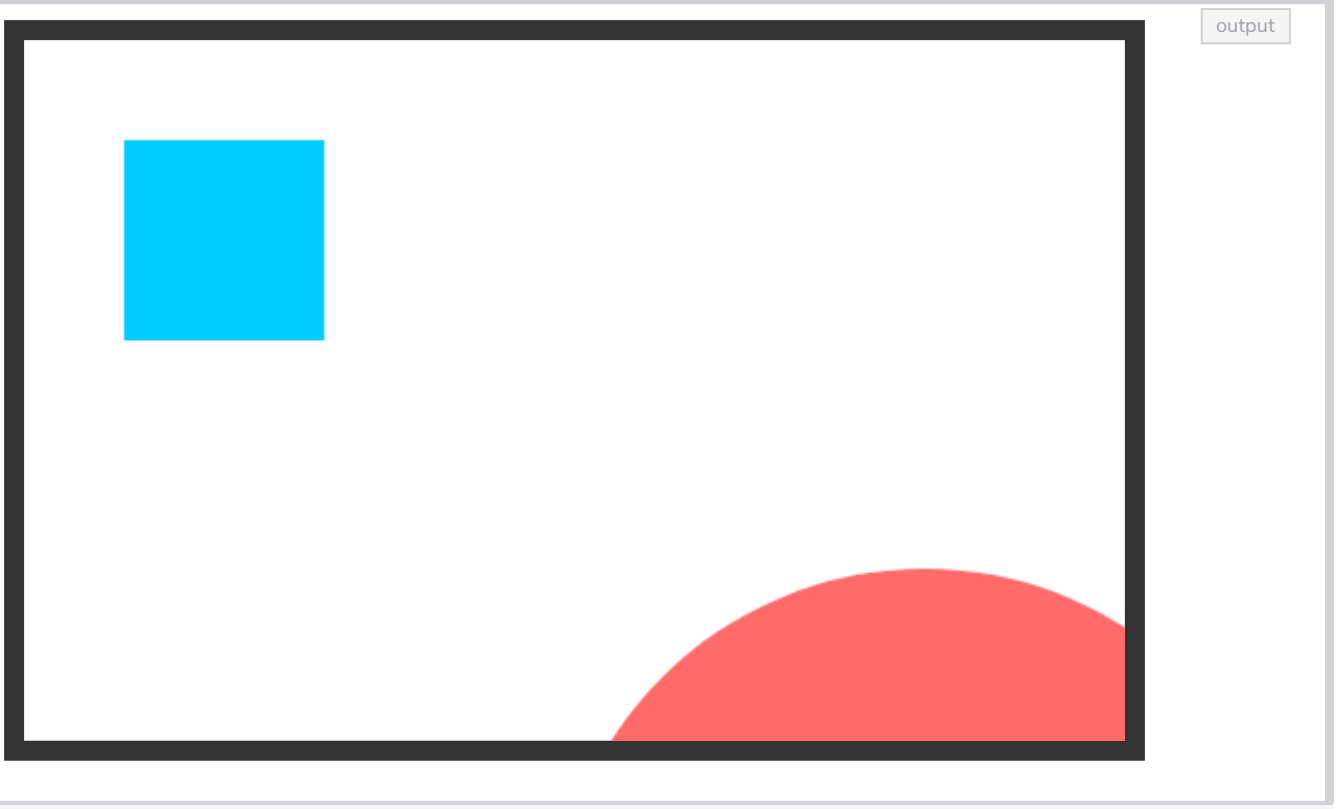
```javascript
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

// Transform
context.translate(50, 50);
context.scale(2, 2);

// Circle
context.beginPath();
context.arc(200, 200, 93, 0, 2 * Math.PI, true);
context.fillStyle = '#FF6A6A';
context.fill();

// Reset the Transform
context.scale(.5, .5);
context.translate(-50, -50);

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



Pay attention to the highlighted lines where we set the transform first and then reset the transform next. Resetting a transform in this approach isn't as simple as specifying the default transform values for `translate` and `scale`:

```
context.scale(1, 1);
context.translate(0, 0);
```

That seems like the logical thing to do, but that only works in a world where the `canvas` can intelligently access its previous state. The moment our canvas gets transformed, it only sees the world through its transformed lenses. Setting a `scale` value of 1 or a `translate` value of 0 means that you just stay at the current transformed state. The fix is where the tediousness comes in:

You have to account for the earlier transform that has been applied and negate it.

If the original transform called for everything to be scaled by 200%, you need to reset the scale by scaling everything by 50% instead. If your `translate` transform shifted everything by 50 pixels horizontally and vertically, you undo this by translating back by 50 pixels in the horizontal and vertical

directions.

That’s what our code highlights:

```
context.scale(.5, .5);
context.translate(-50, -50);
```

There is one more wrinkle. The order you perform this reset is important. Notice that we first undo the scale before resetting the position. If you didn’t do this, you will have changed the position of an element that will then be repositioned again as a result of the scale operation. Getting the position right at this point will require more calculations, and that isn’t particularly fun. This whole section isn’t fun!