12 - More on p5.js

Introducing couple of more p5.js specific subjects like transformations

WE'LL COVER THE FOLLOWING

Rotate & Translate

At this point, we are almost ready to work on our final project; an interactive game built using JavaScript and p5.js! Before we do that, I would like to demonstrate couple more useful p5.js functions to extend the realm of things that we can build.

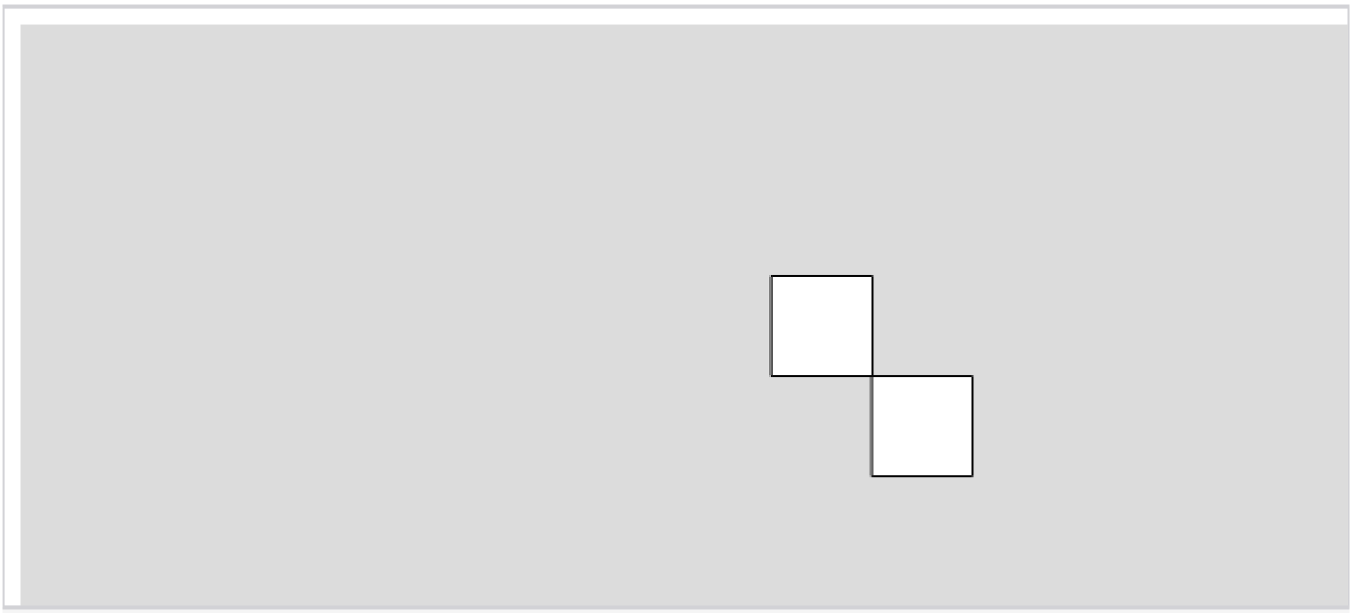
Notice how we can draw shapes on the screen using our current knowledge, but we can't really transform them like rotating them around their center. That's a big blocker on the kinds of visuals that we can build, so let's learn how to do transformations in p5.js to enhance our abilities.

Having used other kinds of drawing libraries, I should say that doing transformations such as scaling, rotating and translating shapes can be a bit unintuitive in p5.js. Here is an example that demonstrates how to use the p5.js **rotate** function that would allow us to rotate shapes.

Output

JavaScript

HTML

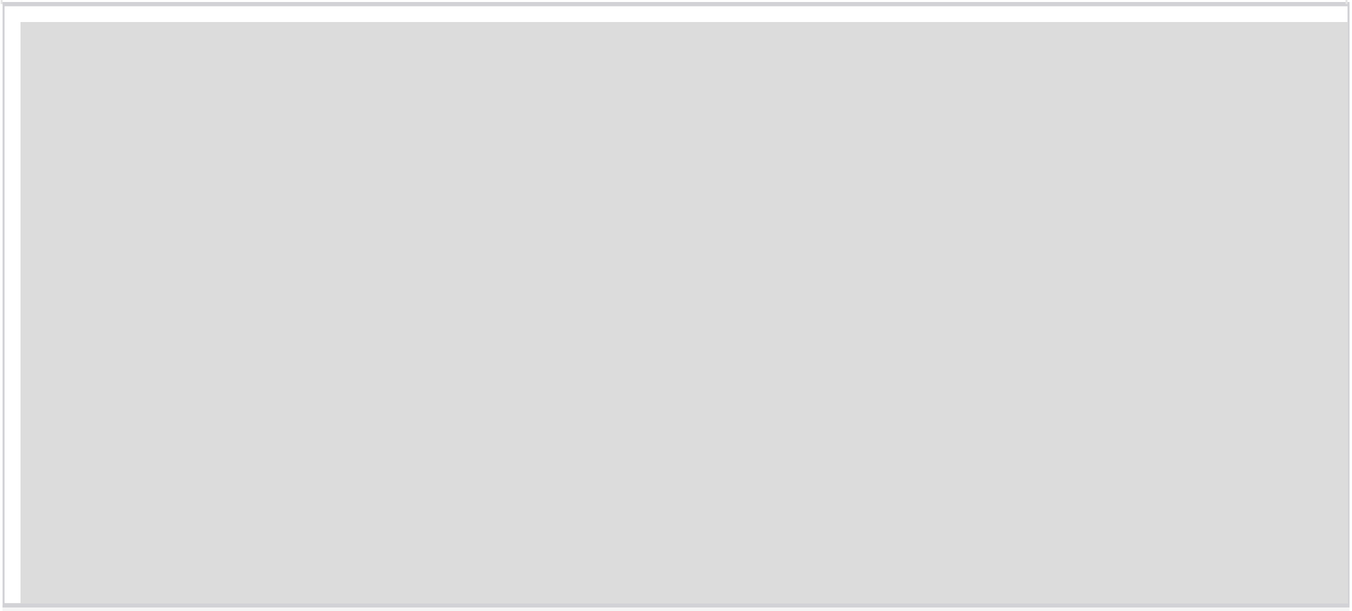


Currently, we are drawing two rectangles that are diagonal to each other. Let's make use of the **rotate** function to see what's going to happen.

Output

JavaScript

HTML



You will notice that both of the shapes disappeared from the screen. If you were expecting the shapes to move by only 5 degrees, this must be a confusing result. This happens because the default units that **rotate** function works result. This happens because the default units that **rotate** function works within p5.js is *radians*. We can make this function work using degrees instead

by using the **angleMode** function with the **DEGREES** p5.js variable. Make this declaration inside the **setup** function.

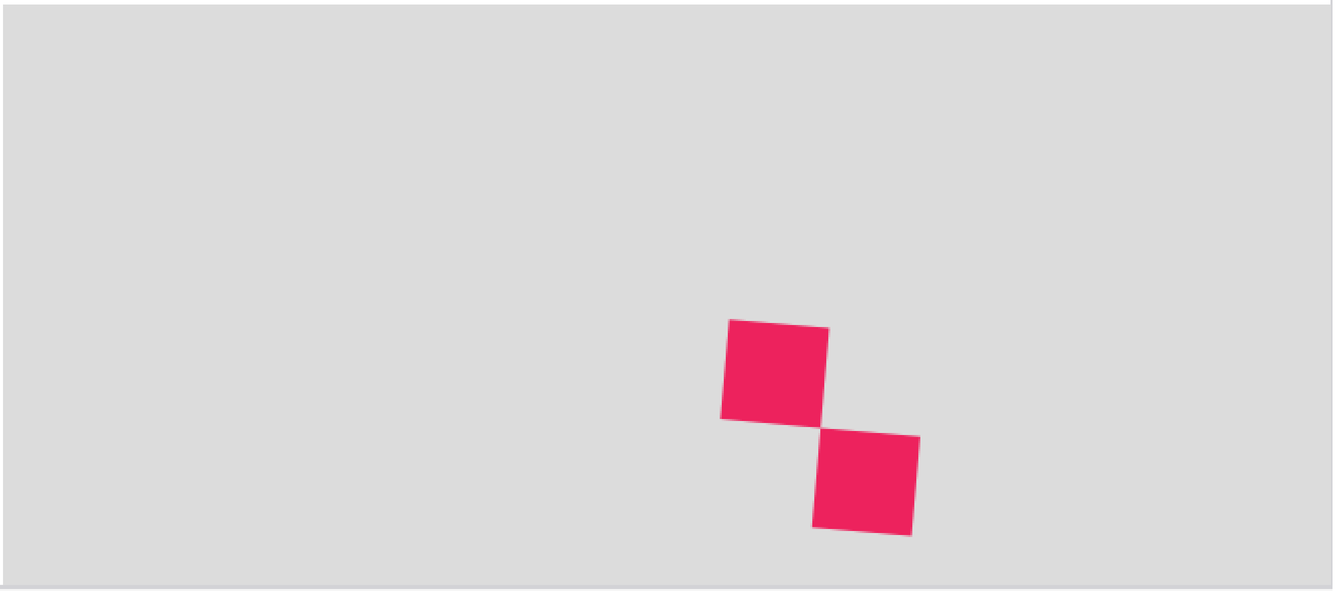
```
    angleMode(DEGREES);
```

Now things work in a way that is more or less expected. We can now observe that when we call the **rotate** function, we end up rotating every shape that comes after the function call.

Output

JavaScript

HTML

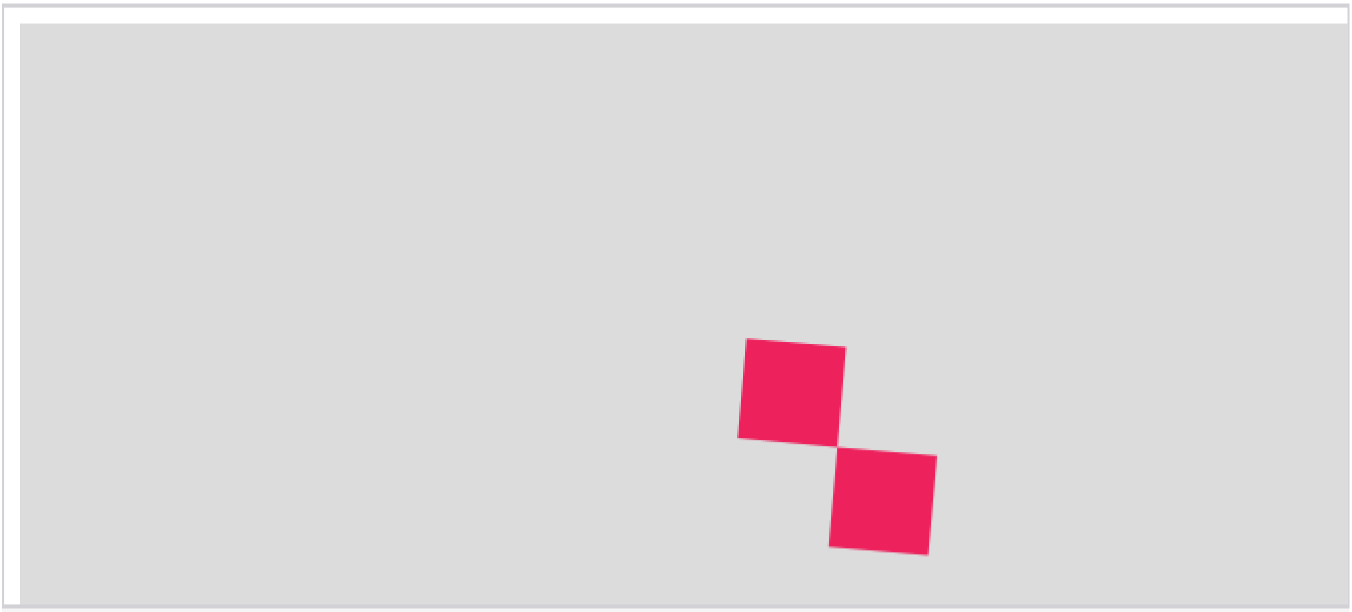


Another thing to notice is that the rotation happens around the origin point, the top left corner of the canvas. Whereas when we are controlling shapes, we would usually like to have them rotate around their origin. So this function, as is, doesn’t seem to be extremely useful.

Output

JavaScript

HTML

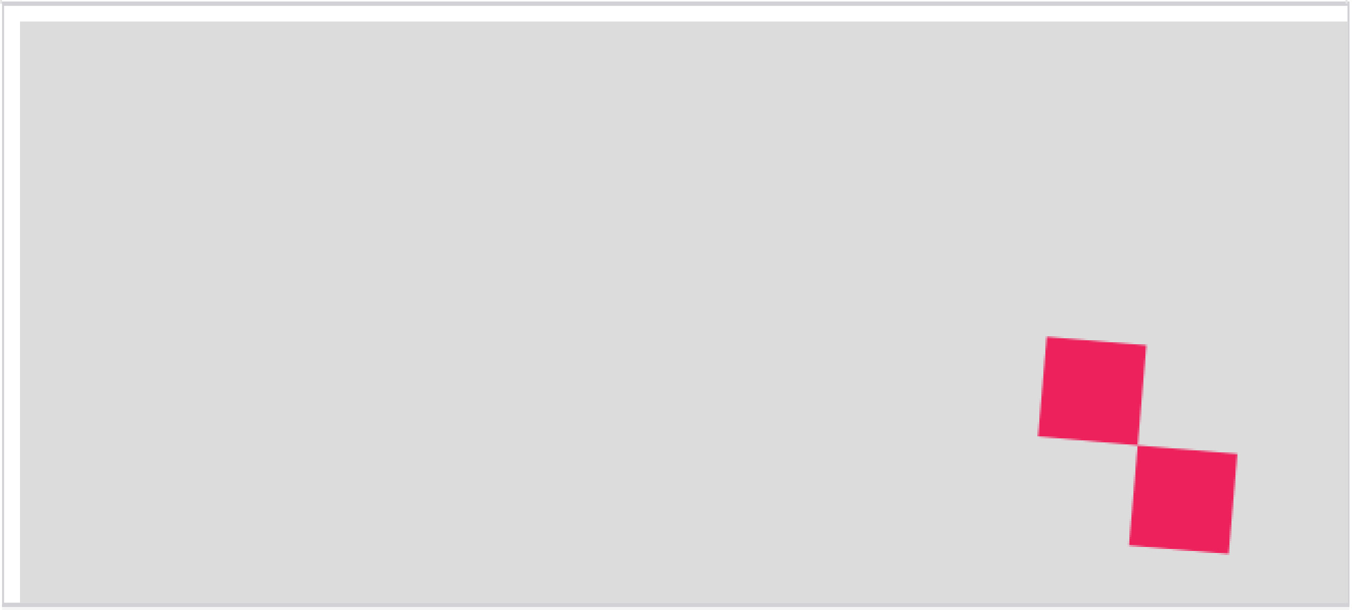


To be able to have better control over the **rotate** function, we should look into the **translate** function. **translate** function moves the object for the given x and y translation amount from the origin point. Let’s make use of it inside our current setup.

Output

JavaScript

HTML

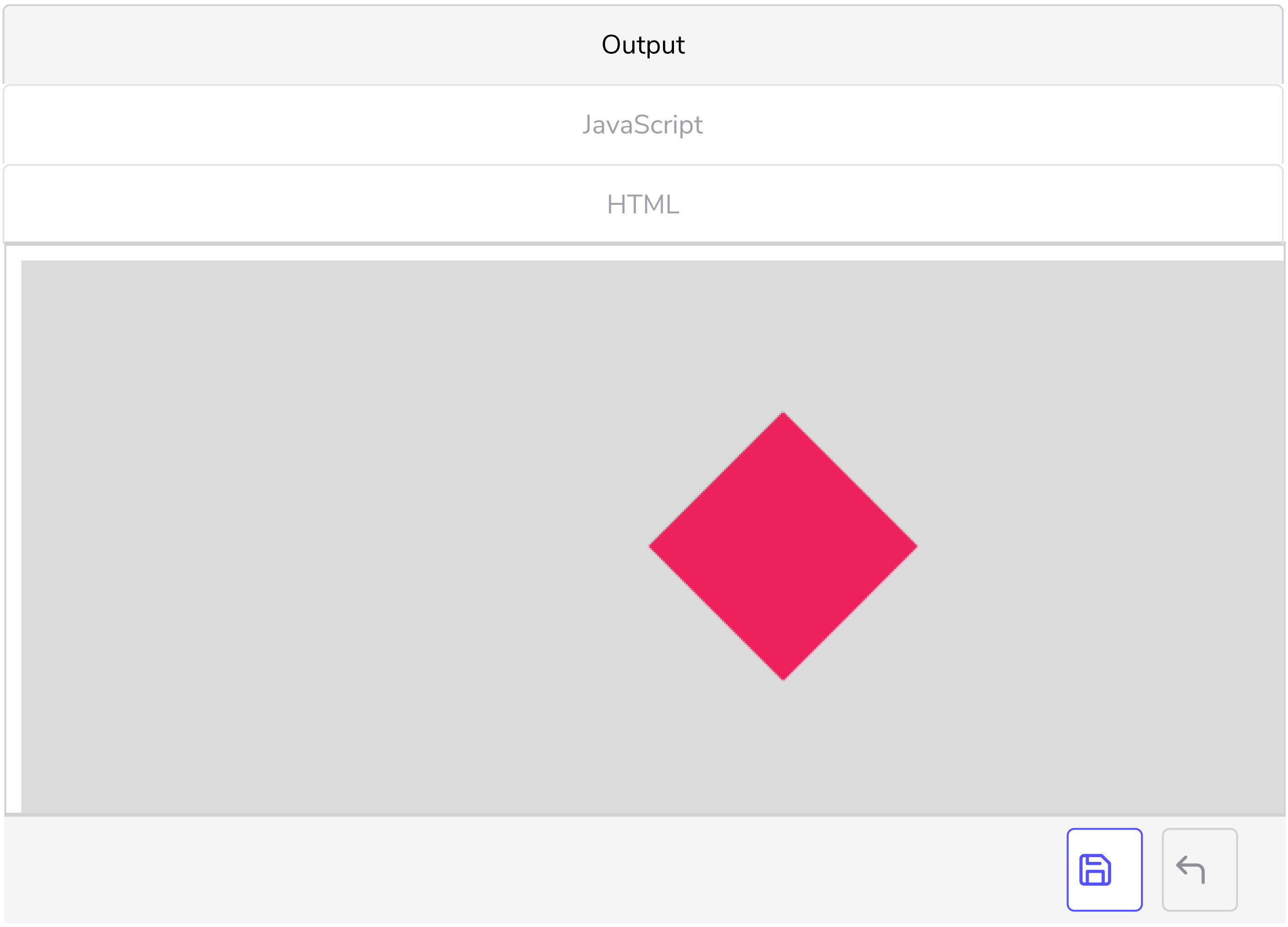


What’s happening right now is that the **translate** function moves everything

What's happening right now is that the **translate** function moves everything inside the canvas 150 pixels to the right. It moves the entire coordinate system

as the rotation is also happening around 150px right-hand side of the origin instead of happening from the origin.

Without further ado, here is how to rotate things around their origin. I think it is easier to show how it is done than to explain it. We will work with a single shape for now.



In this example, we are drawing a shape as we usual, but using the **translation** function to set its x and y coordinates instead of feeding those values directly into the shape drawing function. Doing this, when coupled with using the **rectMode** function allow us to draw the shape with its center located at the origin. Basically, we start off by drawing the shape at the origin point as all the transformation functions work relative to that point. Then we use the **translate** and **rotate** functions to move the shape to desired position and angle. Using this approach, we need to remember to call **rotate** after **translate** function or the rotation will still be happening relative to the original origin point which is probably not desired.

The current shortcoming with this approach and with the usage of transformation functions, in general, is that everything that we draw from

this point onward will be using this new origin point. The way to fix this is using **push** and **pop** functions.

Push & Pop

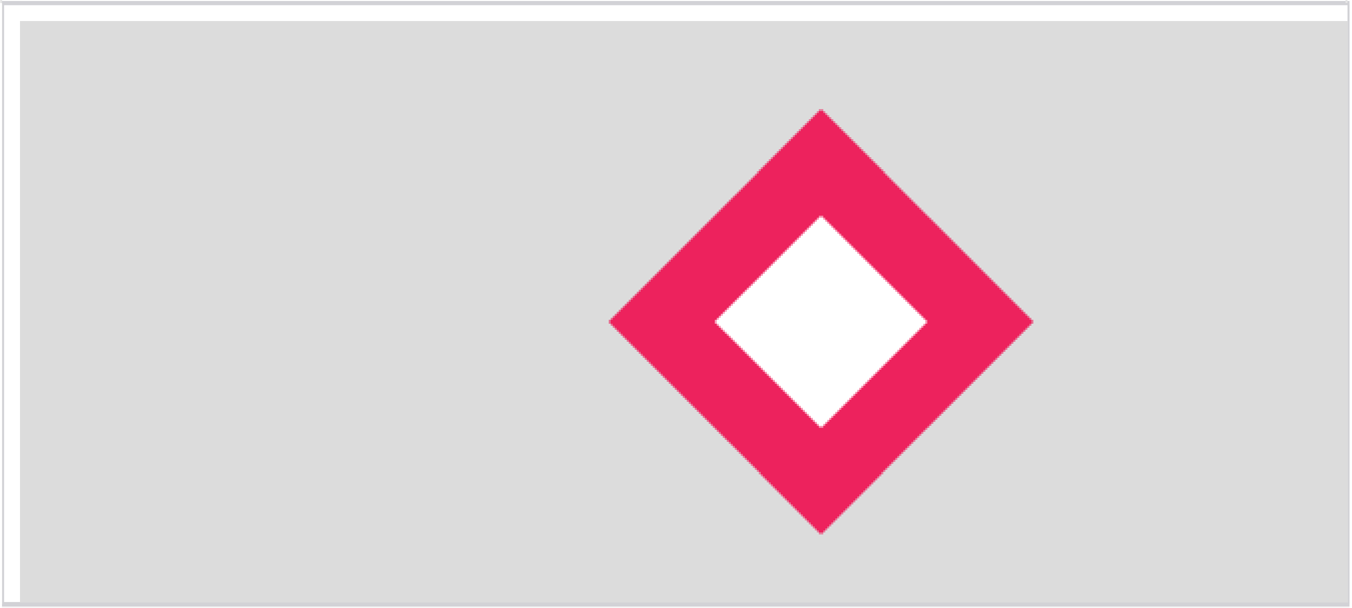
The p5.js **push** function allows us to create a new state and **pop** function restores the state to the previous conditions. This allows us to have completely different settings applied to individual objects without worrying if those settings will affect the shapes that come after as long as we do everything in between a **push** and a **pop** call. Again it is easier to see this in an example.

With our current setup, everything that we draw after the **translate** and **rotate** functions will have that 45 degrees of rotation applied to them.

Output

JavaScript

HTML

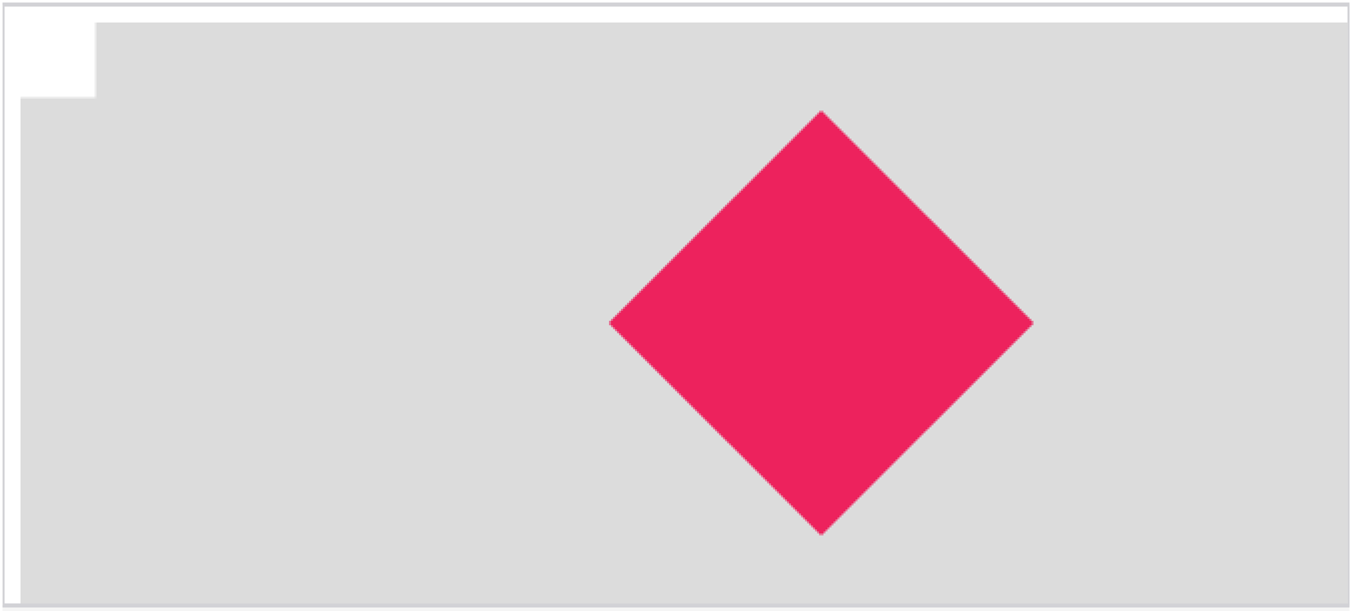


Let’s implement the **push** and **pop** functions here so that we can isolate the transformation that we are applying to the bigger rectangle.

Output

JavaScript

HTML



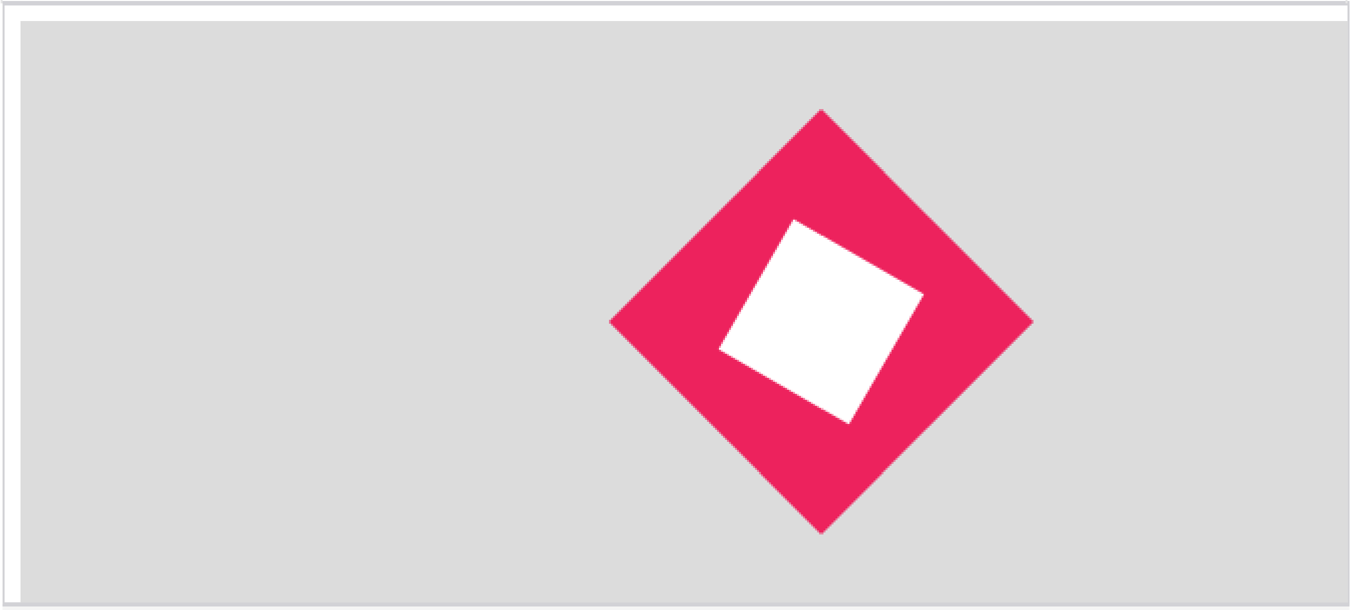
Brilliant! Whatever we are doing in between the **push** and **pop** functions don’t end up affecting anything else that is outside these function calls. It is important to note that we would always call **push** and **pop** functions together. Using one but not the other doesn’t make any sense.

Let’s update our example so that we can still translate the pink rectangle to the middle but apply a different rotation value to it.

Output

JavaScript

HTML



If you find yourself wishing that p5.js transformations weren't this complicated, you can try building your own functions to handle and abstract away the complexity. Here is an example rectangle function that takes a fifth argument, which is the rotation parameter.

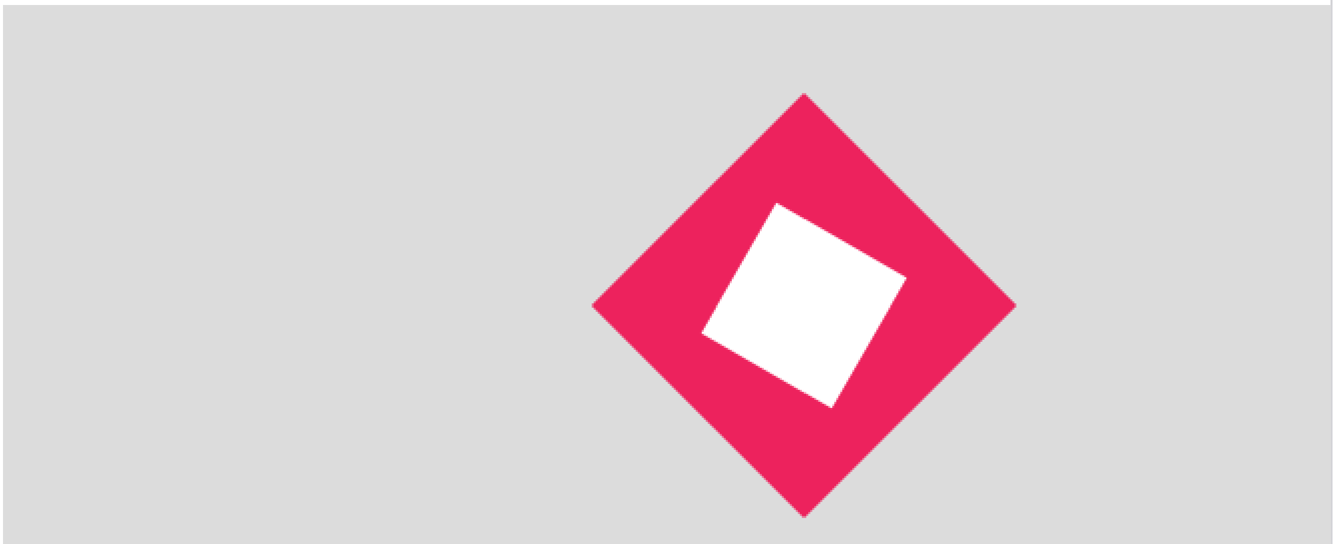
```
    function rectC(x, y, width, height, rotation) {
        if (rotation === undefined) {
            rotation = 0;
        }
        push();
        translate(x, y);
        rotate(rotation);
        rect(0, 0, width, height);
        pop();
    }
```

Here, we are creating our rectangle drawing function called **rectC** that wraps the original **rect** function but uses the **push** and **pop** internally to save state and set transformation, and accepts an optional rotation parameter. If the rotation argument is not provided, then it will assume the value **undefined**. If that's the case, I can just set the rotation value to be 0 instead. Here is the above example refactored to make use of this function. Notice it is much more concise this time.

Output

JavaScript

HTML



Summary

When working with a drawing library, it gets pretty important to be able to transform shapes. In this chapter, we have seen how p5.js transform functions work. We learned about the **translate** and **rotate** functions. We also learned about the **angleMode** function which lets us set the units that would be used by the **rotate** function.

We then learned about the **push** and **pop** functions and found out how they can be used in conjunction with the transformation functions to isolate the state and apply transformations to individual shapes.

Learning these functions aren’t really crucial to learning JavaScript, but I find that knowing about them is pretty essential when using p5.js.

Practice

Try building something cool on your own before moving on to the next chapter where we will be building an interactive game together!