JavaScript - Control and Transform SVG

This project will show you how you can build interactive games using HTML5 Scalable Vector Graphics. This project follows on from ones that animated SVG elements on a page.

We will use SVG transforms to move, rotate and scale our elements around the page. Some documentation for SVG transforms can be found here:
https://developer.mozilla.org/en/docs/Web/SVG/Attribute/transform

Take a Copy of the Starter Page

I have written a template HTML5 page with a single SVG element and the open/close JavaScript tag ready for you. This file can be found in:

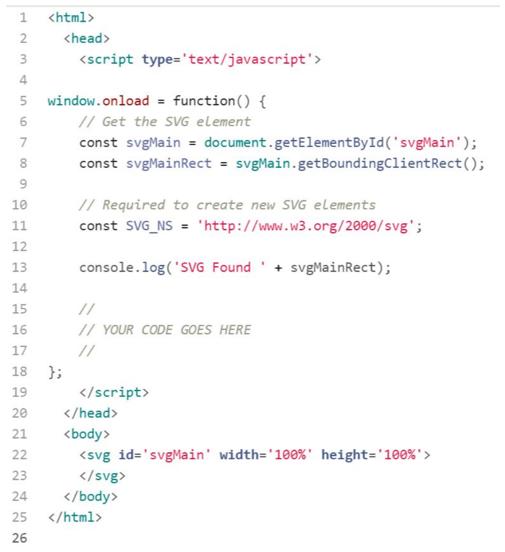
```
KidsProjects/Exercises/JavaScript/svg-game-template.html
```

Open this file, then use **Save As…** to save a copy into your folder:

```
KidsProjects/Kids/<yourname>
```

Check It Works:

You should now have a text editor open (such as Geany) with a file containing the following, but saved to a location in your own folder.

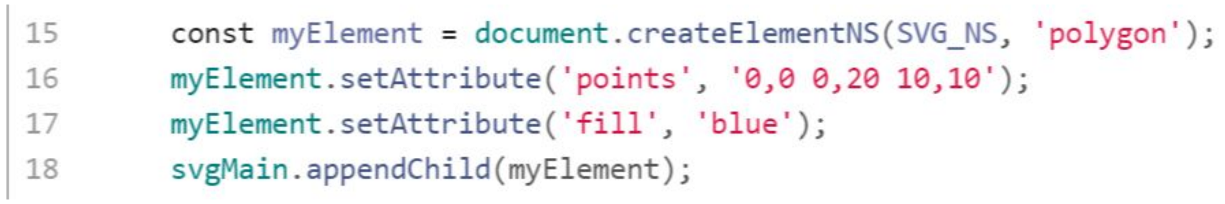


```
<html>
  <head>
    <script type='text/javascript'>

window.onload = function() {
    // Get the SVG element
    const svgMain = document.getElementById('svgMain');
    const svgMainRect = svgMain.getBoundingClientRect();

    // Required to create new SVG elements
    const SVG_NS = 'http://www.w3.org/2000/svg';

    console.log('SVG Found ' + svgMainRect);

    //
    // YOUR CODE GOES HERE
    //
};
    </script>
  </head>
  <body>
    <svg id='svgMain' width='100%' height='100%'>
    </svg>
  </body>
</html>
```

Create the SVG element

For this tutorial we will draw a triangle. SVG doesn't have a specific triangle element, but we can build one using <polygon>.

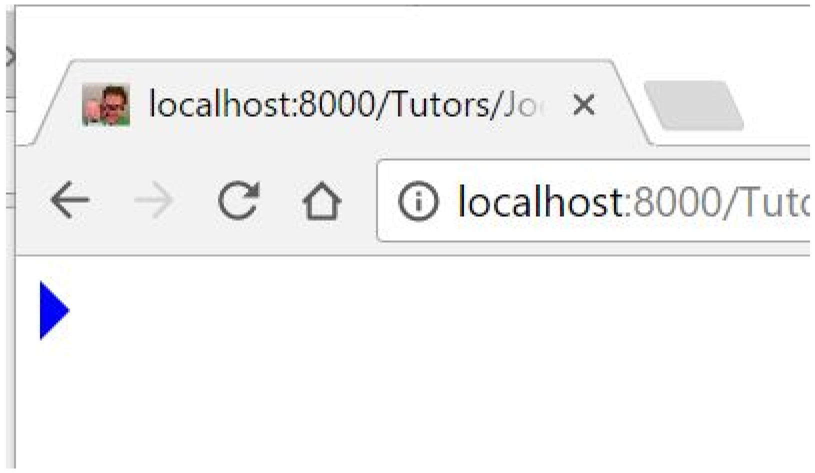
Add the following code by replacing **`// YOUR CODE GOES HERE`** with the following.



```
        const myElement = document.createElementNS(SVG_NS, 'polygon');
        myElement.setAttribute('points', '0,0 0,20 10,10');
        myElement.setAttribute('fill', 'blue');
        svgMain.appendChild(myElement);
```

Check It Works:

Refresh the open page in Chromium and look for a small blue triangle in the top left hand corner of the screen. Mine looked like this:



If you cannot see a triangle, check the **console** in the **Developer Tools** of Chromium.

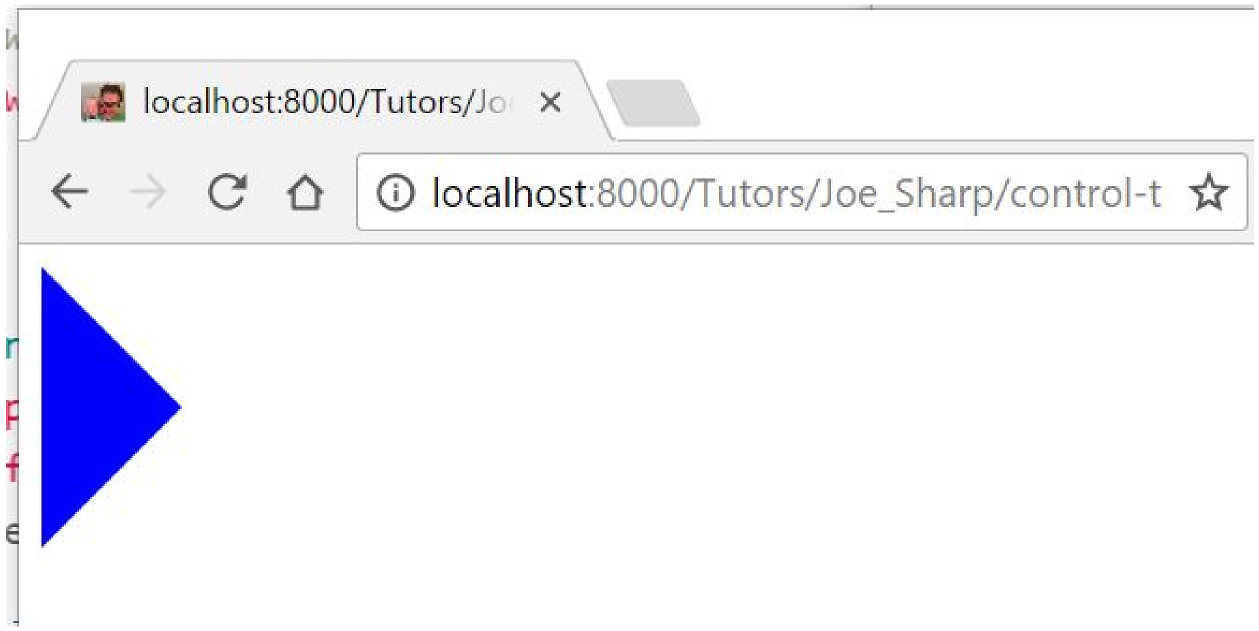
Scaling

We will now create an initial value of scaling to make our shape bigger. Construct the scaling property on our element as shown. This code goes after the initial element construction code.

```
        var scale = 5;
        var transform = 'scale(' + scale + ')';
        myElement.setAttribute('transform', transform);
```

Check It Works:

Refresh your browser and check that the triangle is 5 times the size it originally was. Mine looked like this:



Translating

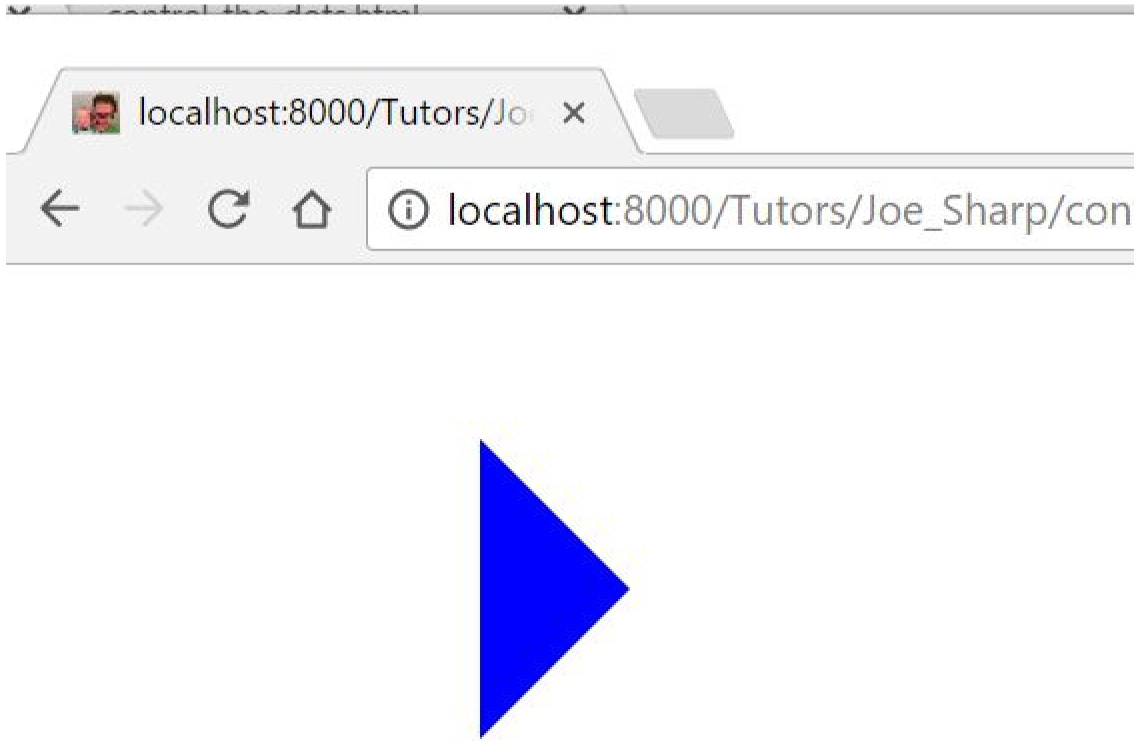
The **transform** attribute can also specify the position of the object by using translation. We will need to add the **translate** specification to our existing variable **transform**. Do the following:

- Create a variable called position to store the x,y values of our translation (lines 21 to 24)
- Append the translation onto the **transform** variable (line 26)

```
20        var scale = 5;
21        var position = {
22          x: 30,
23          y: 10
24        };
25        var transform = 'scale(' + scale + ')';
26        transform += 'translate(' + position.x + ',' + position.y + ')';
27        myElement.setAttribute('transform', transform);
```

Check It Works:

Refresh the browser, the shape should have moved as shown (check the console if this fails):



Rotating

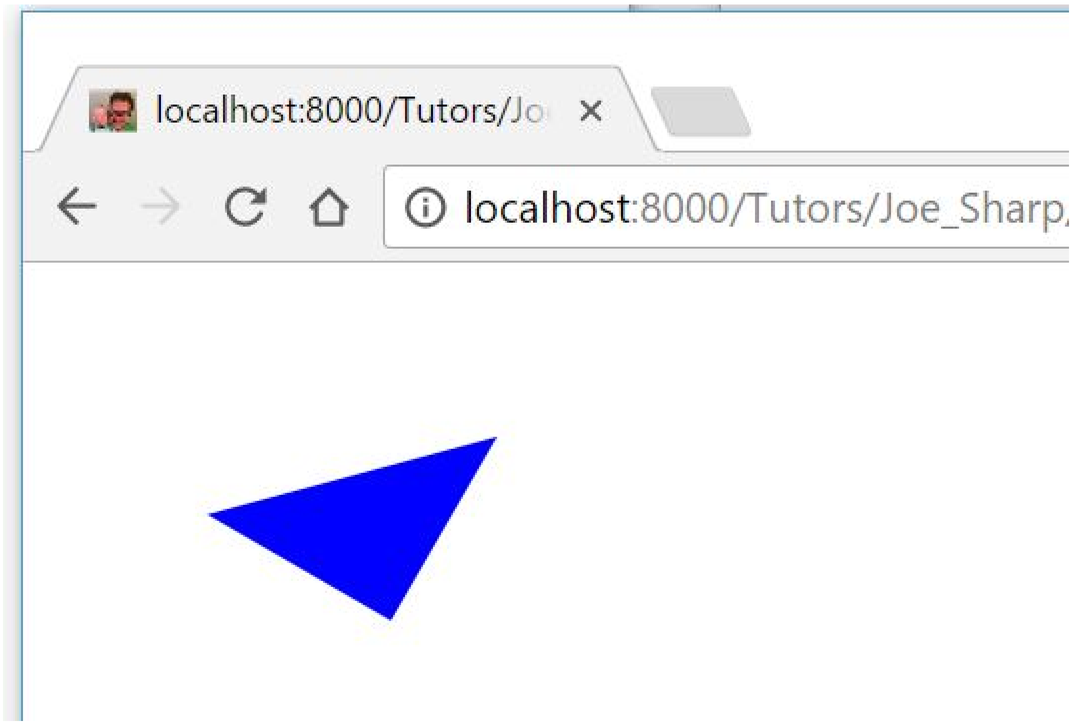
The **transform** attribute can also specify the rotation to apply to the object. This must be part of the same string that contained the scaling function. So modify the code we wrote for scaling as follows:

- Add a variable to store the rotation (line 25)
- Add the rotation to the transform (line 28)

```
20        var scale = 5;
21        var position = {
22          x: 30,
23          y: 10
24        };
25        var rotation = 75;
26        var transform = 'scale(' + scale + ')';
27        transform += 'translate(' + position.x + ',' + position.y + ')';
28        transform += 'rotate(' + rotation + ')';
29        myElement.setAttribute('transform', transform);
```

Check It Works:

Refresh the browser, check for errors (console). The shape should have rotated as shown:

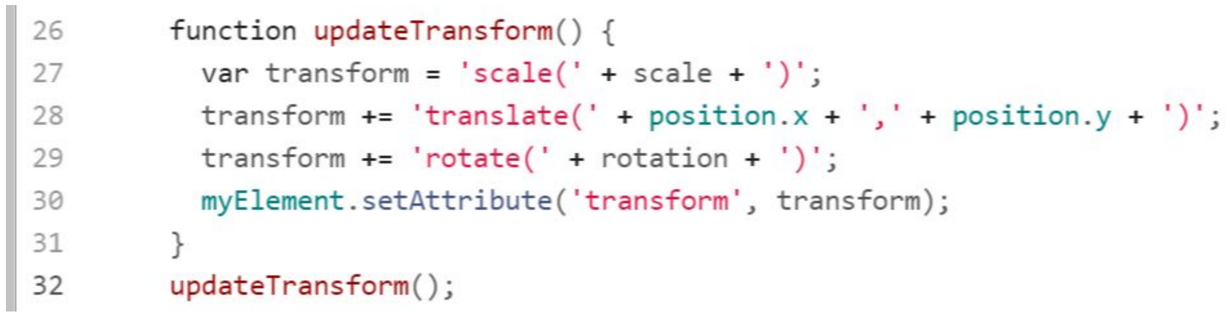


Wrap Transform Update in Function

In the next section we will start using the keyboard to change the values of scale, rotation and translation. But to make that code efficient we should first wrap up our transformation code in a function. This will allow use to call it multiple times from within key pressing code.

Wrap the code that constructs the **transform** variable and applies it to **myElement** in a function as shown (remember you have written most of this code already, just add the function start/end!):

- Indent the appropriate code and wrap it in the **function** syntax (lines 26 and 31)
- Call our **updateTransform** function to make sure the transformation is applied.



```
26      function updateTransform() {
27        var transform = 'scale(' + scale + ')';
28        transform += 'translate(' + position.x + ',' + position.y + ')';
29        transform += 'rotate(' + rotation + ')';
30        myElement.setAttribute('transform', transform);
31      }
32      updateTransform();
```

Check It Works:

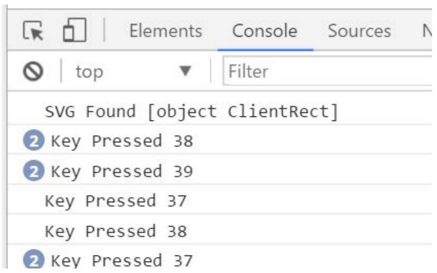
Refresh your page and nothing should have changed. If the shape has reverted to its original position, did you call the **updateTransform** function?

Handle Keys for Scaling

We will use the **window.onkeydown** function to detect key presses. Add the following code after the call to **updateTransform**:

```
    window.onkeydown = function(e) {
      console.log('Key Pressed ' + e.keyCode);
    }
```

Refresh your browser and start pressing keys on the keyboard. I did this using the arrow keys and saw events like this in the console:



```
SVG Found [object ClientRect]
2 Key Pressed 38
2 Key Pressed 39
Key Pressed 37
Key Pressed 38
2 Key Pressed 37
```

Each key has a specific code, you can use the console log to find the values you required. I will be using the UP and DOWN ARROW keys to make our object bigger (up) and smaller (down). To do this I will use a switch statement as shown:

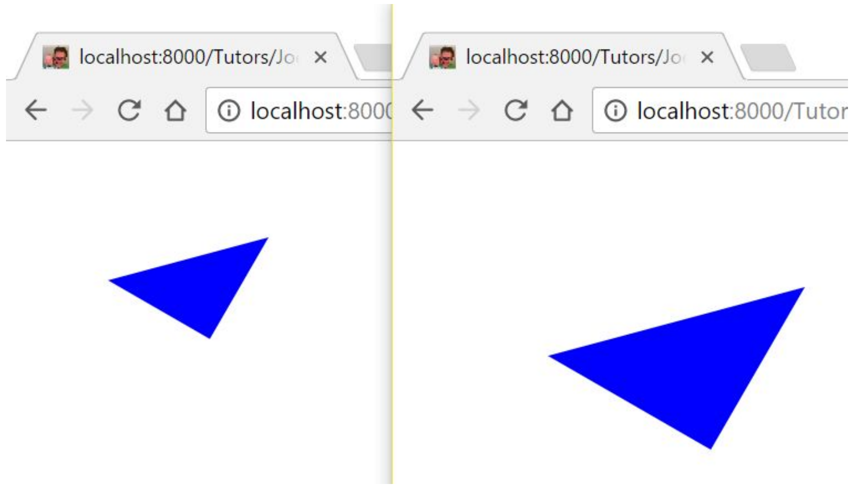
```
    window.onkeydown = function(e) {
      console.log('Key Pressed ' + e.keyCode);
      switch(e.keyCode) {
        case 38: // UP arrow
          scale += 1
          break;
        case 40: // DOWN arrow
          scale -= 1
          break;
      }
      updateTransform();
    }
```

How does this work?

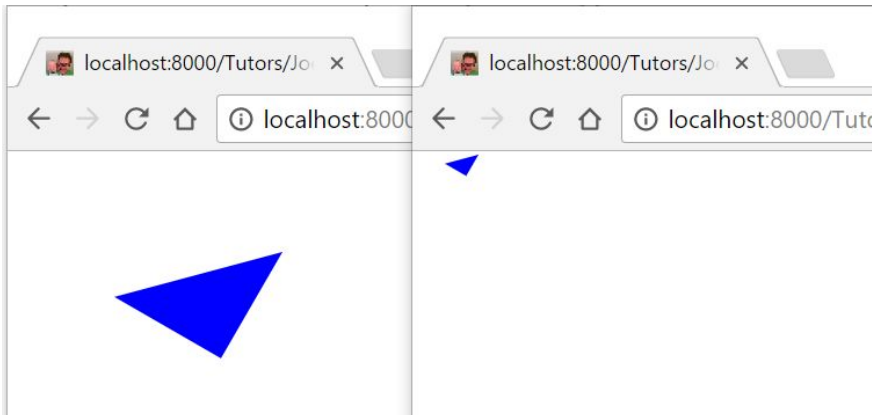
I have used a switch statement to detect the key codes for the UP and DOWN arrows (38 and 40). I can add more keys by adding further **case** statements. In both cases I have modified the **scale** variable and called **updateTransform** which uses the value of **scale** to set the size of our triangle.

Check It Works:

Refresh the browser and keep an eye on the shape. Then use the arrow keys (UP and DOWN) to modify the size of your triangle. It should look like this when grown:



It should look like this when shrunk:



Handle Keys for Rotating

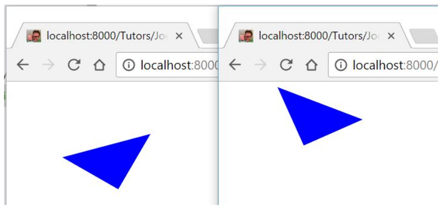
This is as simple as adding a couple of **case** statements and using the values for the RIGHT and LEFT arrows. The LEFT arrow will decrease the rotation, the RIGHT arrow will increase it. Add the case statements as shown from lines 43 to 48:

```
36        switch(e.keyCode) {
37          case 38: // UP arrow
38            scale += 1
39            break;
40          case 40: // DOWN arrow
41            scale -= 1
42            break;
43          case 37: // LEFT arrow
44            rotation -= 1
45            break;
46          case 39: // RIGHT arrow
47            rotation += 1
48            break;
49        }
50        updateTransform();
```

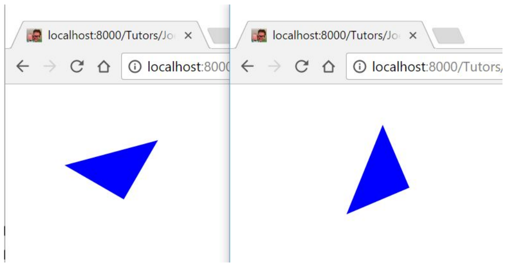
This works in the same way as the scaling. We update the value of the **rotation** variable and the call to updateTransform causes the translation to update.

Check It Works:

Refresh the page, rotation right should look like this:

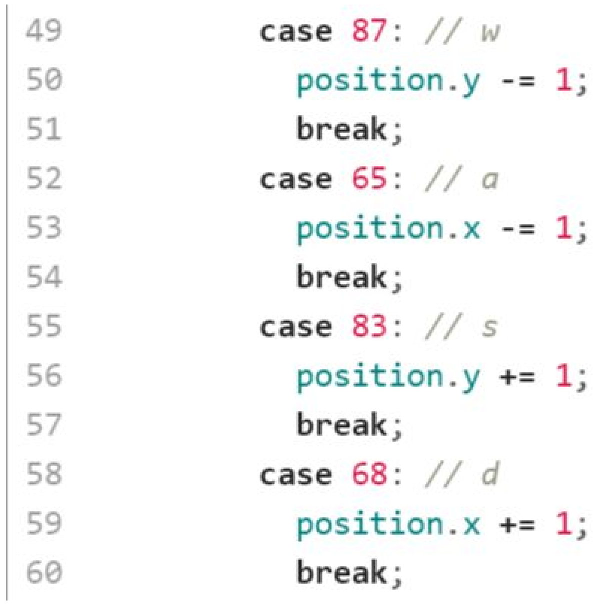


Rotating left should look like this:



Handle Keys for Translation

We will be using the WASD keys to move our shape up, down, left and right. Add the case statements as shown:



```
        case 87: // w
          position.y -= 1;
          break;
        case 65: // a
          position.x -= 1;
          break;
        case 83: // s
          position.y += 1;
          break;
        case 68: // d
          position.x += 1;
          break;
```

I used the console.log() to find the key codes for WASD. The comments in the lines above shown which is which.

In this instance I am modifying the **position.x** and **position.y** values which are then used in the **updateTransform** function.

Check It Works:

Refresh the page and try to move your shape around. It should now move around the screen when using the WASD keys.

Conclusion

This tutorial shows you how to draw polygons, and how to modify their position, size and orientation using **transform**. You have also learnt how to respond to key presses.

As you can probably see, it would take a lot of code to write a game this way.

Congratulations, you have completed the SVG control tutorial.