Adjusting the Position

It is time to tie together the triangle we've drawn with the code we just added for dealing with the arrow keys. What we are going to do is define two counter variables called `deltaX` and `deltaY`. What these variables will do is keep a count of how far to move our triangle as a result of arrow key presses. This may sound a bit confusing right now, but hang on tight!

First, let's go ahead and define our `deltaX` and `deltaY` variables and put them to use inside our `moveSomething` function. Add lines 1-2, 7, 10, 13 and 16 to your code:

```
var deltaX = 0;
var deltaY = 0;

function moveSomething(e) {
    switch(e.keyCode) {
        case 37:
            deltaX -= 2;
            break;
        case 38:
            deltaY -= 2;
            break;
        case 39:
            deltaX += 2;
            break;
        case 40:
            deltaY += 2;
            break;
    }
}
```

Depending on which arrow key was pressed, either the `deltaX` or `deltaY` variable will be increased or decreased. These variables changing in isolation has no effect on our triangles. We need to modify our `drawTriangle` function to actually use the `deltaX` and `deltaY` variables. Guess what we are going to do next?

Go ahead and make changes (line 2, 5-7) to the `drawTriangle` function:

```
function drawTriangle() {
  context.clearRect(0, 0, canvas.width, canvas.height);
  // the triangle
  context.beginPath();
  context.moveTo(200 + deltaX, 100 + deltaY);
  context.lineTo(170 + deltaX, 150 + deltaY);
  context.lineTo(230 + deltaX, 150 + deltaY);
  context.closePath();

  // the outline
  context.lineWidth = 10;
  context.strokeStyle = "rgba(102, 102, 102, 1)";
  context.stroke();

  // the fill color
  context.fillStyle = "rgba(255, 204, 0, 1)";
  context.fill();
}
```

The code changes should be pretty straightforward to make sense of. The call to `clearRect` ensures we clear our `canvas` before attempting to re-draw our triangle. The additions to the `context.moveTo` and `context.lineTo` methods take the `deltaX` and `deltaY` values into account. This ensures our triangle is always drawn with an offset that is determined by the number of times you pressed each arrow key. Putting that last sentence into human terms, this means you can move your triangle around using the keyboard.

At this point, if you preview your page now, our example still won't work. The reason is because there is one more thing you need to do. We need to call `drawTriangle` each time a key is pressed to actually draw our triangle in the new position. To make this happen, go back to the `moveSomething` function and add a call to `drawTriangle` towards the bottom:

```
function moveSomething(e) {
    switch(e.keyCode) {
        case 37:
            deltaX -= 2;
            break;
        case 38:
            deltaY -= 2;
            break;
        case 39:
            deltaX += 2;
            break;
        case 40:
            deltaY += 2;
            break;
    }

    drawTriangle();
}
```

Give your `canvas` element focus by clicking on the triangle, and then use your arrow keys. You'll see our triangle moving around the screen!

HTML JavaScript

javascript

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

var deltaX = 0;
var deltaY = 0;

window.addEventListener("keydown", keysPressed, false);
window.addEventListener("keyup", keysReleased, false);

var keys = [];

function keysPressed(e) {
    // store an entry for every key pressed
    keys[e.keyCode] = true;

    // left
    if (keys[37]) {
      deltaX -= 2;
    }

    // right
    if (keys[39]) {
      deltaX += 2;
    }

    // down
    if (keys[38]) {
      deltaY -= 2;
    }

    // up
```

output