Improved Keyhandling Logic

WE'LL COVER THE FOLLOWING

- Why we aren't using requestAnimationFrame

When talking about our current logic for dealing with the keyboard events, I mentioned that we are using a less-than-ideal solution. To see why, go back to your example and **press and hold** the Up and Right arrow keys at the same time. What you would expect to see is your triangle moving diagonally. What you actually see is your triangle moving in only one direction - either right or up. That isn't what we want!

The reason for this bizarre behavior is because we are using a `switch` statement to figure out which arrow key was pressed. There is nothing wrong with this general approach, for `switch` statements are far less verbose than `if`/`else-if` statements for checking which condition happens to equate to **true**. For many general cases, this is fine. How often are people going to hold down multiple keys at the same time? As it turns out, for the interactive/game-ey things that we are doing, pressing multiple keys will be a common occurrence. Pressing the Up and Right arrows on the keyboard at the same time is the equivalent of pushing a joystick diagonally. Don't tell me you've never done that before!

The solution is to change how we check for which key was pressed. Replace your existing `addEventListener` call and `moveSomething` function with the following instead:

```
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
    if (keys[40]) {
      deltaY += 2;
    }

    e.preventDefault();

    drawTriangle();
}

function keysReleased(e) {
    // mark keys that were released
    keys[e.keyCode] = false;
}
```

This code change looks pretty massive, but it's not that invasive. What we've done is simply take our existing code and combine it with the Detecting Multiple Key Presses code you saw in the Keyboard Events tutorial. This change ensures that we can press multiple keys and get the desired behavior that we want. You can see that if you test your page out again and try pressing multiple arrow keys. Win!

> ### Why we aren't using requestAnimationFrame
>
> Earlier, when having a circle follow our mouse cursor around, we gave `requestAnimationFrame` the responsibility of drawing (and re-drawing) our circle. We didn't have our draw code be part of the **mousemove** event handler that fires each time your mouse cursor moves. That is in stark contrast to what we did here where our `moveSomething` (and `keysPressed`) event handlers are directly responsible for calling our

`drawTriangle` function. Why did we do two different things in what looks like an identical situation?

The reason has to do with how chatty some events are with respect to your frame rate. The goal is to do work to update our screen at a rate that is consistent with our frame rate - ideally 60 times a second. Your **mousemove** event fires waaaaaaay too rapidly. Forcing things to get drawn with each **mousemove** fire would lead to a lot of unnecessary work with only a fraction of that work actually showing up on screen. That is why we defer drawing our circle in the **mousemove** case to `requestAnimationFrame` and its insane ability to stay in sync with the frame rate. Our **keydown** event (kinda like all keyboard events) is not very chatty at all. Having our **keydown** event handler (`moveSomething`) be responsible for drawing and shifting our triangle by two pixels is totally an OK call.

With all of this said, we will look at an example later where we will use `requestAnimationFrame` to **smoothly** move things using our keyboard!