Performing animation using the JavaScript `setInterval(...)` function

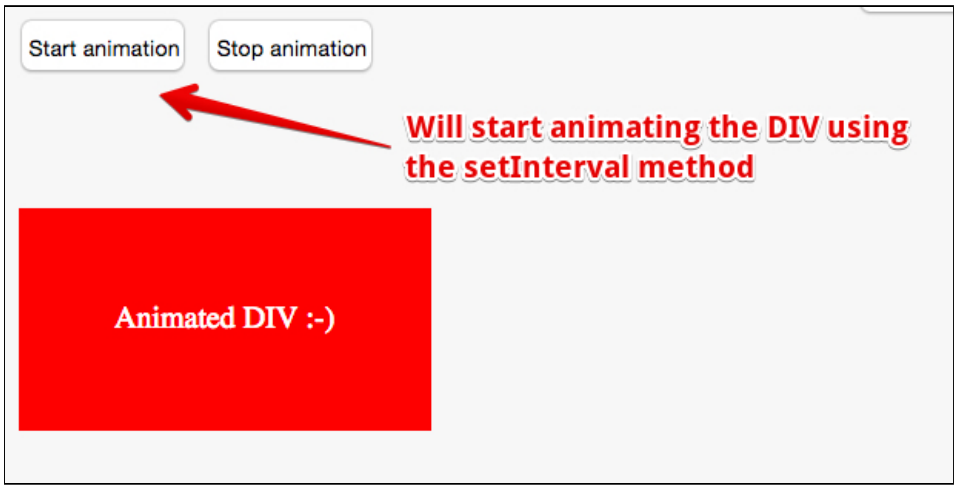
The `setInterval(...)` function is still popular on the Web, and even if this is not the recommended way to do 60 frames/second canvas animation, it's worth understanding how it works.

- Syntax: `setInterval(function, ms);`

The `setInterval(...)` function calls another function or evaluates an expression at specified intervals of time (in milliseconds), and returns the unique `id` of the action. You can always stop it by calling the `clearInterval(id)` function with the interval identifier as an argument.

BASIC EXAMPLE THAT SHOWS HOW TO ANIMATE A DIV USING THE DOM API (THIS IS HOW PRE-HTML5 GAMES WERE WRITTEN)

Please try it online: http://jsbin.com/huluhe/2/edit (open the html, JavaScript and output tabs):



Extract from the source code:

```
<body>
  <div id="animatedDIV">Animated DIV :-)</div>
  <button onclick="start()">Start animation</button>
  <button onclick="stop()">Stop animation</button>
  <script>
```

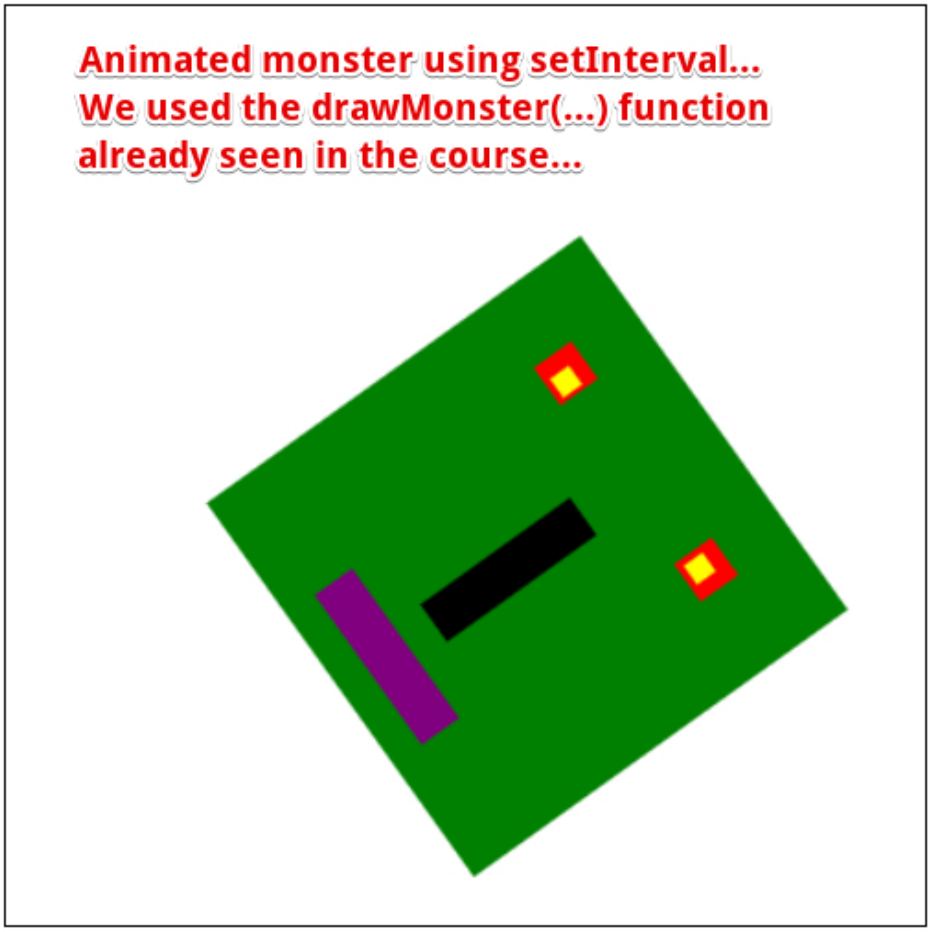
```
    var elm = document.getElementById("animatedDIV");
    var requestId;
    var x = 0;
    function render(time) {
        elm.style.left = x++ + "px";
    }
    function start() {
        requestId = setInterval(render, 10);
    }
    function stop() {
        if (requestId) {
            clearInterval(requestId);
        }
    }
  </script>
</body>
```

Here, we define a `<div>` element, (see the online source code for the CSS properties involved), and we use the `setInterval` method (line 17) to call every 10ms the`render()` method that will just increment the position of this element. Notice that since we're using the DOM, the horizontal position of the div is modified by changing its `left`CSS property.

The call to `setInterval` returns an id we can use to stop the animation, by calling`clearInterval` (line 22).

Animate the monster in a canvas, using setInterval

Online example: http://jsbin.com/lomopo/1/edit



Source code:

```
<body onload="init();">
    <canvas id="myCanvas" width="400" height="400">
        Your browser does not support the canvas tag.
    </canvas>
    <p>
    <button onclick="start()">Start animation</button>
    <button onclick="stop()">Stop animation</button>
    <script>
        var canvas, ctx;
        var monsterX=100, monsterY=100, monsterAngle=0;
        function init() {
            // This function is called after the page is loaded
            // 1 - Get the canvas
            canvas = document.getElementById('myCanvas');
            // 2 - Get the context
```

```
        ctx=canvas.getContext('2d');
    }

    function animationLoop() {
        // 1 - Clear the canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        // 2 Draw the monster using variables for pos, angle, etc.
        drawMonster(monsterX, monsterY, monsterAngle, 'green', 'yellow');
        // 3 Move the monster (change pos, angle, size, etc.)
        monsterX += 10;
        monsterX %= canvas.width
        monsterAngle+= 0.01;
    }
    function drawMonster(x, y, angle, headColor, eyeColor) {
    // BEST PRACTICE: SAVE CONTEXT AND RESTORE IT AT THE END
    ctx.save();
    // Moves the coordinate system so that the monster is drawn
    // at position (x, y)
    ctx.translate(x, y);
    ctx.rotate(angle)
    // head
    ctx.fillStyle=headColor;
    ctx.fillRect(0,0,200,200);
    ...
    // BEST PRACTICE!
    ctx.restore();
}
function start() {
    // Start the animation loop, change 20 for bigger values
    requestId = setInterval(animationLoop, 20);
}
function stop() {
    if (requestId) {
        clearInterval(requestId);
    }
}
</script>
</body>
```

- *Lines 52-61*: The code for launching and stopping the animation is similar to that from the previous example.
- *Lines 34-50*: The code that draws the monster is that which we saw earlier when we

presented the 2D transformations. Best practice consists in saving and restoring the context at the beginning and end of each function that changes the context.

- *Lines 21-32*: **The most interesting part is the animation loop** that implements the basic animation steps: *clear-draw-move*. In order to make a shape "movable", we use some "state variables" for its position and angle, and we modify them at each iteration (*lines 29-32*). We will see later on how to modify the value of these variables on user interactions (keyboard, mouse, etc.).

PROBLEMS WITH SETINTERVAL

Running several animations simultaneously

The `setInterval` function may become hard to debug, in particular if you run several animations simultaneously. For example, if you have two intervals, one running every 100 milliseconds, the other every second and if you want to debug the second one, the first one will constantly interrupt the previous one, making step by step debugging really difficult.

A single animation may be interrupted by itself to become two simultaneous animations

`setInterval` will execute the function passed as first parameter every `n`milliseconds regardless of when the function was last called or how long the function takes to execute. If the function takes longer than the interval, then `setInterval` runs a new call that will interrupt the previous one, leading to unpredictable results. The new call may overwrite the previous call data, etc.

> ***BEST PRACTICE**: AVOID using `setInterval` for animating in a canvas, except for trivial cases (change a color every second).*

KNOWLEDGE CHECK 4.2.3 (NOT GRADED)

What is the correct syntax for setInterval?

- setInterval(function, milliseconds)
- setInterval(milliseconds, function)