Timing Functions

setTimeout

The window object provides some useful methods for scheduling the execution of a function and for repeatedly executing functions at regular intervals.

The `window.setTimeout()` method accepts a callback to a function as its first parameter and a number of milliseconds as its second parameter. Try entering the following example into a console. It should show an alert dialog after three seconds (that's 3000 milliseconds):

```
window.setTimeout(function(){ alert("Time's Up!") }, 3000);
<< 4
```

Notice that the method returns an integer. This is an ID used to reference that particular timeout. It can also cancel the timeout using the `window.clearTimeout()` method. Try calling the code again:

```
window.setTimeout(function(){ alert("Time's Up!") }, 3000);
<< 5
```

Now quickly enter the following code before the alert pops up:

```
window.clearTimeout(5); // make sure you use the correct ID
<< undefined
```

If you were quick enough, and used the correct ID, the alert was prevented from happening.

setInterval

The `window.setInterval()` method works in a similar way to `window.setTimeout()`, except that it will continue to invoke the callback function after every given number of milliseconds.

The previous example used an anonymous function, but it is also possible to use a named function like so:

```
function hello(){ console.log("Hello"); };
```

Now we can set up the interval and assign it to a variable:

```
repeat = window.setInterval(hello,1000)
<< 6
```

This should show the message "Hello" in the console every second (1,000 milliseconds).

To stop this, we can use the `window.clearInterval()` method and the variable repeat as an argument (this is because the `window.setInterval()` method returns its ID, so this will be assigned to the variable repeat):

```
window.clearInterval(repeat);
```

requestAnimationFrame

This method of the window object works in much the same way as the `window.setInterval()` method, although it has a number of improvements to optimize its performance. These include making the most of the browser's built-in graphics-handling capabilities and not running the animation when the tab is inactive, resulting in a much smoother performance. It is supported in all major browsers, including Internet Explorer from version 10 onwards. Create and style a div element and append the id "square" to it before you try the following code.

```
var square = document.getElementById("square");
var angle = 0;
function rotate() {
  angle = (angle + 5)%360
  square.style.transform = "rotate(" + angle + "deg)"
  window.requestAnimationFrame(rotate);
}
id = window.requestAnimationFrame(rotate);
```

This is similar to the earlier code, but this time we place the rotation code inside a function called rotate. The last line of this function uses the `window.requestAnimationFrame()` method and takes the `rotate()` function as an argument. This will then call the `rotate()` function recursively. The frame rate cannot be set using `requestAnimationFrame()`; it's usually 60 frames per second, although it's optimized for the device being used.

To start the animation, we need to call the `window.requestAnimationFrame()` method, giving the `rotate()` function as an argument. This will return a unique ID that can be employed to stop the animation using the `window.cancelAnimationFrame()` method:

```
window.cancelAnimationFrame(id);
```