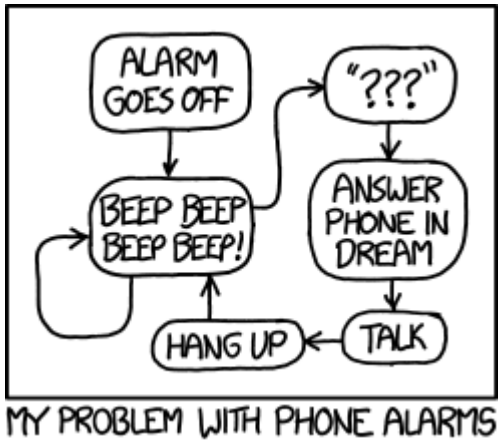
CSE 154

LECTURE 19: EVENTS AND TIMERS



Anonymous functions

```
function(parameters) {
  statements;
}
```
JS

- JavaScript allows you to declare **anonymous functions**
- quickly creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.

Anonymous function example

```
window.onload = function() {
  var ok = document.getElementById("ok");
  ok.onclick = okayClick;
};

function okayClick() {
  alert("booyah");
}
```

JS

output

- or the following is also legal (though harder to read and bad style):

```
window.onload = function() {
  document.getElementById("ok").onclick = function() {
    alert("booyah");
  };
};
```

Unobtrusive styling

```
function okayClick() {
  this.style.color = "red";
  this.className = "highlighted";
}
```
JS

```
.highlighted { color: red; }
```
CSS

- well-written JavaScript code should contain as little CSS as possible
- use JS to set CSS classes/IDs on elements
- define the styles of those classes/IDs in your CSS file

The danger of global variables

```
var count = 0;
function incr(n) {
  count += n;
}
function reset() {
  count = 0;
}

incr(4);
incr(2);
console.log(count);
```
JS

- globals can be bad; other code and other JS files can see and modify them
- How many global symbols are introduced by the above code?
- 3 global symbols: `count`, `incr`, and `reset`

Enclosing code in a function

```
function everything() {
  var count = 0;
  function incr(n) {
    count += n;
  }
  function reset() {
    count = 0;
  }

  incr(4);
  incr(2);
  console.log(count);
}

everything();
// call the function to run the code
```

- the above example moves all the code into a function; variables and functions declared inside another function are local to it, not global

- How many global symbols are introduced by the above code?
- 1 global symbol: `everything` (can we get it down to 0?)

The "module pattern"

```
(function() {
    statements;
})();
```
JS

- wraps all of your file's code in an anonymous function that is declared and immediately called
- 0 global symbols will be introduced!
- the variables and functions defined by your code cannot be messed with externally

Module pattern example

```
(function() {
  var count = 0;
  function incr(n) {
    count += n;
  }
  function reset() {
    count = 0;
  }

  incr(4);
  incr(2);
  console.log(count);
})();
```

JS

- How many global symbols are introduced by the above code?
- 0 global symbols

JavaScript "strict" mode

```
"use strict";

your code...
```

```
6  "use strict";
7
8  function calculate() {
9      abc = 42;
10
11 Uncaught ReferenceError: abc is not defined
12
13     // go get the subtotal and tip amounts from the page
14     var subtotalBox = document.getElementById("subtotal");
```

- writing `"use strict";` at the very top of your JS file turns on strict syntax checking:
  - shows an error if you try to assign to an undeclared variable
  - stops you from overwriting key JS system libraries
  - forbids some unsafe or error-prone language features
- You should *always* turn on strict mode for your code in this class!

The six global DOM objects

Every Javascript program can refer to the following global objects:

name	description
document	current HTML page and its content
history	list of pages the user has visited
location	URL of the current HTML page
navigator	info about the web browser you are using
screen	info about the screen area occupied by the browser
window	the browser window

The window object

the entire browser window; the top-level object in DOM hierarchy

- technically, all global code and variables become part of the `window` object
- properties:
  - `document`, `history`, `location`, `name`
- methods:
  - `alert`, `confirm`, `prompt` (popup boxes)
  - `setInterval`, `setTimeout` `clearInterval`, `clearTimeout` (timers)
  - `open`, `close` (popping up new browser windows)
  - `blur`, `focus`, `moveBy`, `moveTo`, `print`, `resizeBy`, `resizeTo`, `scrollBy`, `scrollTo`

Popup windows with window.open

```
window.open("http://foo.com/bar.html", "My Foo Window",
            "width=900,height=600,scrollbars=1");
```
JS

- `window.open` pops up a new browser window
- THIS method is the cause of all the terrible popups on the web!
- some popup blocker software will prevent this method from running

The document object

the current web page and the elements inside it

- properties:
  - `anchors`, `body`, `cookie`, `domain`, `forms`, `images`, `links`, `referrer`, `title`, `URL`
- methods:
  - `getElementById`
  - `getElementsByName`, `getElementsByTagName`
  - `querySelector`, `querySelectorAll`
  - `close`, `open`, `write`, `writeln`

The location object

the URL of the current web page

- properties:
  - `host`, `hostname`, `href`, `pathname`, `port`, `protocol`, `search`
- methods:
  - `assign`, `reload`, `replace`

The navigator object

information about the web browser application

- properties:
  - `appName`, `appVersion`, `browserLanguage`, `cookieEnabled`, `platform`, `userAgent`
- Some web programmers examine the `navigator` object to see what browser is being used, and write browser-specific scripts and hacks:

```
if (navigator.appName === "Microsoft Internet Explorer") { ...
```
JS

  - (this is poor style; you usually do not need to do this)

The screen object

information about the client's display screen

- properties:
  - `availHeight`, `availWidth`, `colorDepth`, `height`, `pixelDepth`, `width`

The history object

the list of sites the browser has visited in this window

- properties:
  - `length`
- methods:
  - `back`, `forward`, `go`
- sometimes the browser won't let scripts view `history` properties, for security

Setting a timer

method	description
setTimeout(*function*, *delayMS*);	arranges to call given function after given delay in ms
setInterval(*function*, *delayMS*);	arranges to call function repeatedly every *delayMS* ms
clearTimeout(*timerID*); clearInterval(*timerID*);	stops the given timer

- both `setTimeout` and `setInterval` return an ID representing the timer
  - this ID can be passed to `clearTimeout`/`Interval` later to stop the timer

setTimeout example

```html
<button id="clickme">Click me!</button>
<span id="output"></span>
```

HTML

```js
window.onload = function() {
  document.getElementById("clickme").onclick = delayedMessage;
};

function delayedMessage() {
  document.getElementById("output").innerHTML = "Wait for it...";
  setTimeout(sayBooyah, 5000);
}

function sayBooyah() {    // called when the timer goes off
  document.getElementById("output").innerHTML = "BOOYAH!";
}
```

JS

Click me!

output

setInterval example

```
var timer = null;  // stores ID of interval timer

function delayMsg2() {
  if (timer === null) {
    timer = setInterval(rudy, 1000);
  } else {
    clearInterval(timer);
    timer = null;
  }
}

function rudy() {   // called each time the timer goes off
  document.getElementById("output").innerHTML += " Rudy!";
}
```

JS

Click me!

output

Passing parameters to timers

```
function delayedMultiply() {
  // 6 and 7 are passed to multiply when timer goes off
  setTimeout(multiply, 2000, 6, 7);
}
function multiply(a, b) {
  alert(a * b);
}
```

JS

- any parameters after the delay are eventually passed to the timer function
  - doesn't work in IE; must create an intermediate function to pass the parameters
- why not just write this?

```
setTimeout(multiply(6 * 7), 2000);
```

JS

Common timer errors

- many students mistakenly write `()` when passing the function

```
setTimeout(booyah(), 2000);
setTimeout(booyah, 2000);

setTimeout(multiply(num1 * num2), 2000);
setTimeout(multiply, 2000, num1, num2);
```
JS

- what does it actually do if you have the `()` ?
  - it calls the function immediately, rather than waiting the 2000ms!