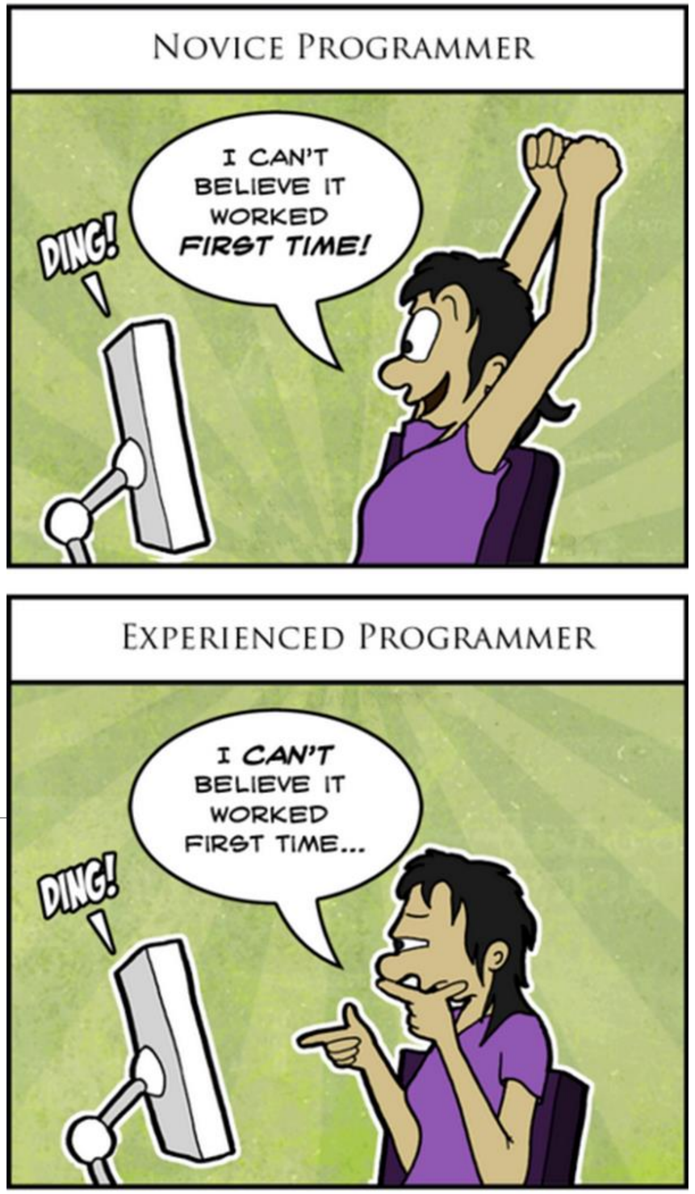
# CSc 337

LECTURE 11: KEYBOARD EVENTS


NOVICE PROGRAMMER
DING!
I CAN'T BELIEVE IT WORKED FIRST TIME!
EXPERIENCED PROGRAMMER
DING!
I CAN'T BELIEVE IT WORKED FIRST TIME...

## Multiple window.onload listeners

```
~~window.onload = function;~~
window.addEventListener("load", function);                         JS
```

- it is considered bad form to directly assign to `window.onload`
- multiple .js files could be linked to the same page, and if they all need to run code when the page loads, their `window.onload` statements will override each other
- by calling `window.addEventListener` instead, all of them can run their code when the page is loaded

## Mouse events

| | |
|---|---|
| click | user presses/releases mouse button on the element |
| dblclick | user presses/releases mouse button twice on the element |
| mousedown | user presses down mouse button on the element |
| mouseup | user releases mouse button on the element |

clicking

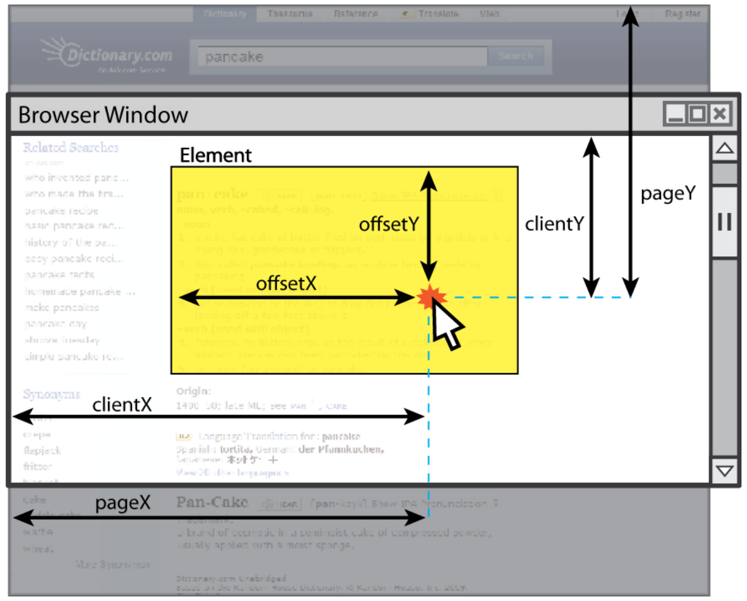
| | |
|---|---|
| mouseover | mouse cursor enters the element's box |
| mouseout | mouse cursor exits the element's box |
| mousemove | mouse cursor moves around within the element's box |

movement

# Mouse event objects

The `event` passed to a mouse handler has these properties:

| **property/method** | **description** |
|---|---|
| clientX<br>clientY | coordinates in *browser window* |
| screenX<br>screenY | coordinates in *screen* |
| offsetX<br>offsetY | coordinates in *element* (non-standard) |
| button | integer representing which button was pressed (0=Left, 1=Middle, 2=Right) |



## Keyboard/text events

| name | description |
|---|---|
| focus | this element gains keyboard **focus** (attention of user's keyboard) |
| blur | this element loses keyboard focus |
| keydown | user presses a key while this element has keyboard focus |
| keyup | user releases a key while this element has keyboard focus |
| keypress | user presses and releases a key while this element has keyboard focus |
| select | this element's text is selected or deselected |

## Key event objects

| **property name** | **description** |
|---|---|
| keyCode | ASCII integer value of key that was pressed (convert to char with String.fromCharCode) |
| altKey, ctrlKey, shiftKey | true if Alt/Ctrl/Shift key is being held |

- issue: if the event you attach your listener to doesn't have the focus, you won't hear the event
  - possible solution: attach key listener to entire page body, `document`, an outer element, etc.

## Key event example

```
document.getElementById("textbox").onkeydown = textKeyDown;
...
function textKeyDown(event) {
  var key = String.fromCharCode(event.keyCode);
  if (key == 'S' && event.altKey) {
    alert("Save the document!");
    this.value = this.value.split("").join("-");
  }
}                                                        JS
```

- each time you push down any key, even a modifier such as Alt or Ctrl, the `keydown` event fires
- if you hold down the key, the `keydown` event fires repeatedly
- `keypress` event is a bit flakier and inconsistent across browsers

## Stopping an event

| **event method name** | **description** |
|---|---|
| preventDefault | stops the browser from doing its normal action on an event; for example, stops the browser from following a link when <a> tag is clicked, or stops browser from submitting a form when submit button is clicked |
| stopPropagation | stops the browser from showing this event to any other objects that may be listening for it |

- you can also `return false;` from your event handler to stop an event

## Stopping an event, example

```
<form id="exampleform" action="http://foo.com/foo.php">...</form>
```

```
window.onload = function() {
  var form = document.getElementById("exampleform");
  form.onsubmit = checkData;
};

function checkData(event) {
  if (document.getElementById("state").length != 2) {
    alert("Error, invalid city/state.");  // show error message
    event.preventDefault();
    return false;                         // stop form submission
  }
}
```
JS

## Some useful key codes

| **keyboard key** | **event keyCode** |
|---|---|
| Backspace | 8 |
| Tab | 9 |
| Enter | 13 |
| Escape | 27 |
| Page Up, Page Down, End, Home | 33, 34, 35, 36 |
| Left, Up, Right, Down | 37, 38, 39, 40 |
| Insert, Delete | 45, 46 |
| Windows/Command | 91 |
| F1 - F12 | 112 - 123 |

## Page/window events

| name | description |
|---|---|
| contextmenu | the user right-clicks to pop up a context menu |
| error | an error occurs when loading a document or an image |
| load, unload | the browser loads the page |
| resize | the browser window is resized |
| scroll | the user scrolls the viewable part of the page up/down/left/right |
| unload | the browser exits/leaves the page |

- The above can be handled on the `window` object

## Removing a node from the page

```
function slideClick() {
  var bullet = document.getElementById("removeme");
  bullet.parentNode.removeChild(bullet);
}                                                              JS
```

- odd idiom: *obj*.parentNode.remove(*obj*);

## Getting/setting CSS classes

```
function highlightField() {
  // turn text yellow and make it bigger
  var text = document.getElementById("text");
  if (!text.className) {
    text.className = "highlight";
  } else if (text.className.indexOf("invalid") < 0) {
    text.className += " highlight";    // awkward
  }
}                                                              JS
```

- JS DOM's className property corresponds to HTML class attribute
- somewhat clunky when dealing with multiple space-separated classes as one big string

## Getting/setting CSS classes with classList

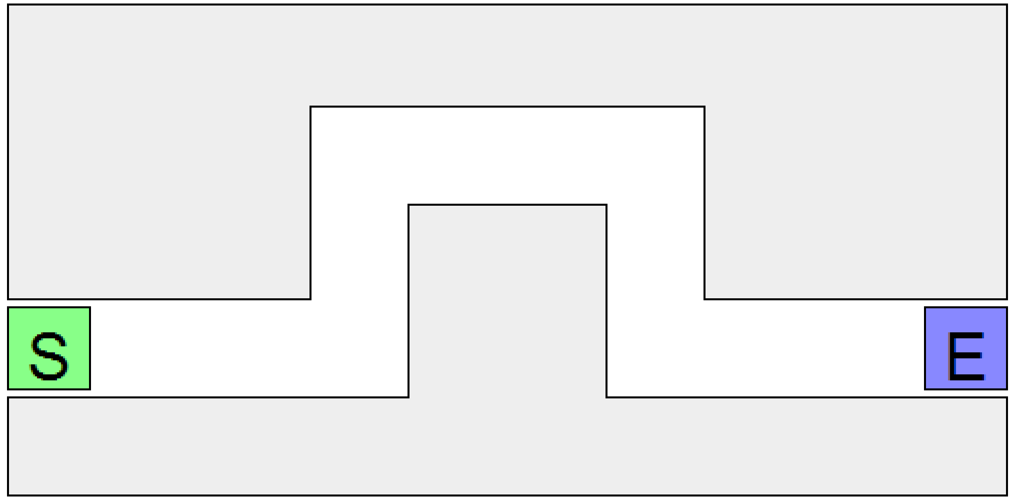
```
function highlightField() {
  // turn text yellow and make it bigger
  var text = document.getElementById("text");
  if (!text.classList.contains("invalid")) {
    text.classList.add("highlight");
  }
}                                                                    JS
```

- classList collection has methods add, remove, contains, toggle to manipulate CSS classes
- similar to existing className DOM property, but don't have to manually split by spaces

## Activity: Mouse Maze

# Mouse Maze

This lab practices unobtrusive JavaScript events and the Document Object Model (DOM). We'll write a page with a "**maze**" to navigate with the mouse. You will write maze.js to implement the maze behavior.



## Info about the maze

Download the file below (right-click, Save Target As...) to get started: maze.html

The difficulty is in having the dexterity to move the mouse through **without touching any walls**. When the mouse cursor touches a wall, all walls turn red and a "You lose" message shows. Touching the Start button with the mouse removes the red coloring from the walls.

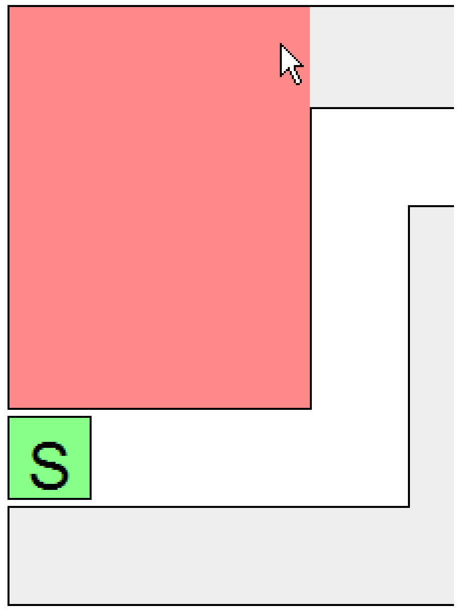
The maze walls are 5 div elements. Our provided CSS puts the `div`s into their proper places.

```
<div id="maze">
    <div id="start">S</div>
    <div class="boundary" id="boundary1"></div>
    <div class="boundary"></div>
    <div class="boundary"></div>
    <div class="boundary"></div>
    <div class="boundary"></div>
    <div id="end">E</div>
</div>
```

# Exercise : Single boundary turns red

Write code so that when the user moves the mouse onto a single one of the maze's walls (`onmouseover`), that wall will turn red. Use the top-left wall; it is easier because it has an `id` of `boundary1`.

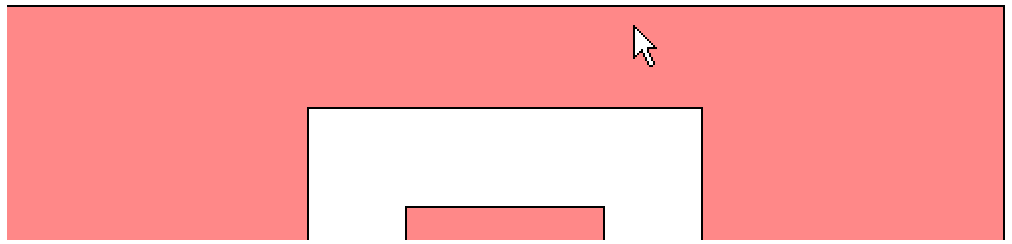
- Write your JS code unobtrusively, without modifying `maze.html`.
- Write a `window.onload` handler that sets up any event handlers.
- Handle the event on the wall by making it turn red.
- Turn the wall red by setting it to have the provided CSS class you lose, using the `classList` property.



## Exercise : All boundaries glow red on hover

Make it so that **all maze walls turn red** when the mouse enters any one of them.

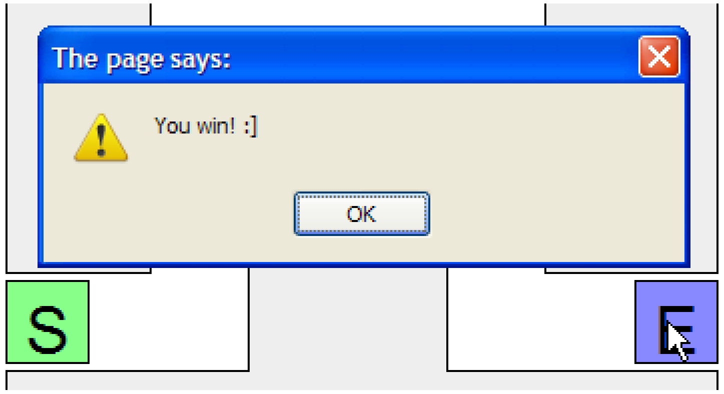
- You'll need to attach an event handler to each `div` that represents a wall of the maze.
- It is harder to select all of these `div`s, since they do not have `id` attributes.
- But they do all have a `class` of `boundary`. Use the `document.querySelectorAll` function to access them all.



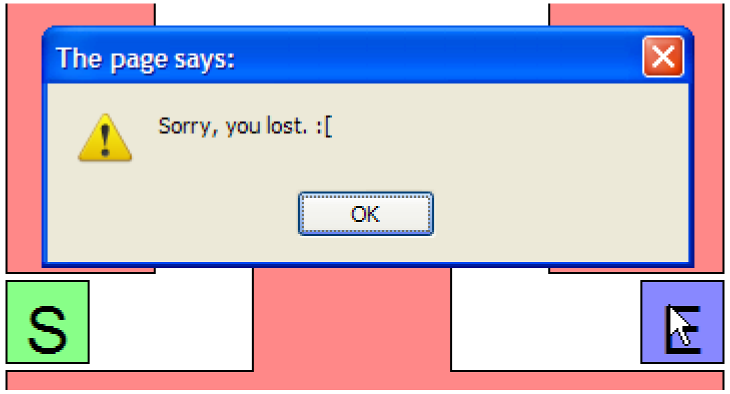
# Exercise : Alert on completion of maze

Make it so that if the user reaches the end of the maze, a "You win!" alert message appears.

- The end of the maze is a `div` with an `id` of `end`.
- Don't pop up "You win!" unless the user makes it to the end **without touching any walls**.
- Keep track of whether any walls were hit, so you'll know what to do when the end square is hit.


The page says:

You win! :]

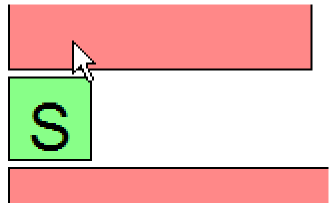

The page says:

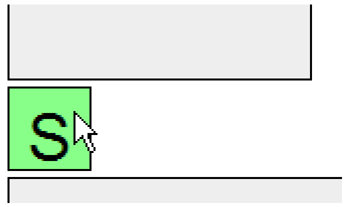
Sorry, you lost. :[

# Exercise : Restartable maze

Make it so that when the user clicks the mouse on the **Start** square (a `div` with an `id` of `start`), the maze state will reset. That is, if the maze boundary walls are red, they will all return to their normal color, so that the user can try to get through the maze again.

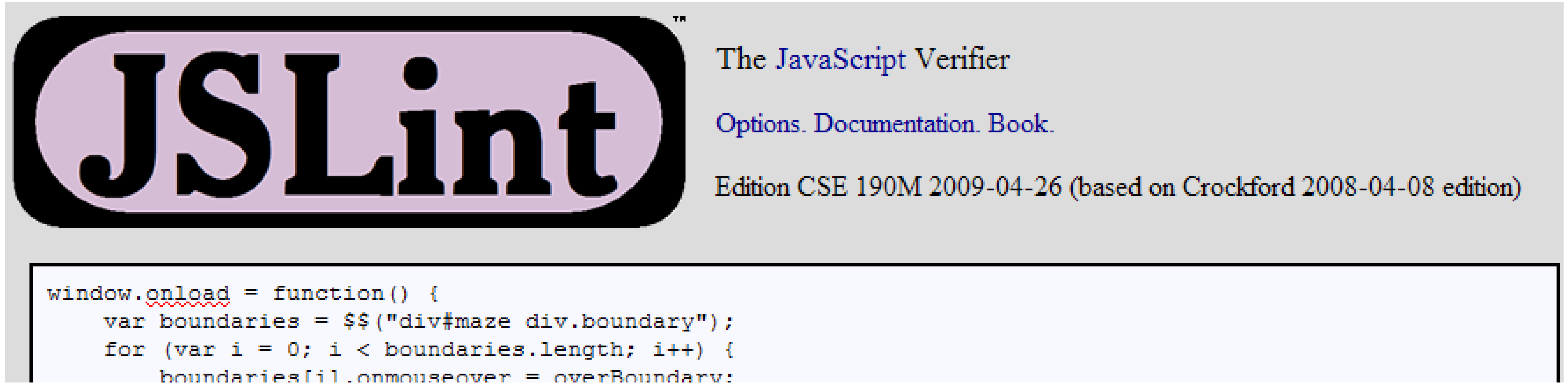
- You'll need to use the `document.querySelectorAll` function again to select all of the squares to set their color.





## Exercise : JSLint

- Verify your JavaScript code by making sure it passes JSLint with no errors.


The JavaScript Verifier

Options. Documentation. Book.

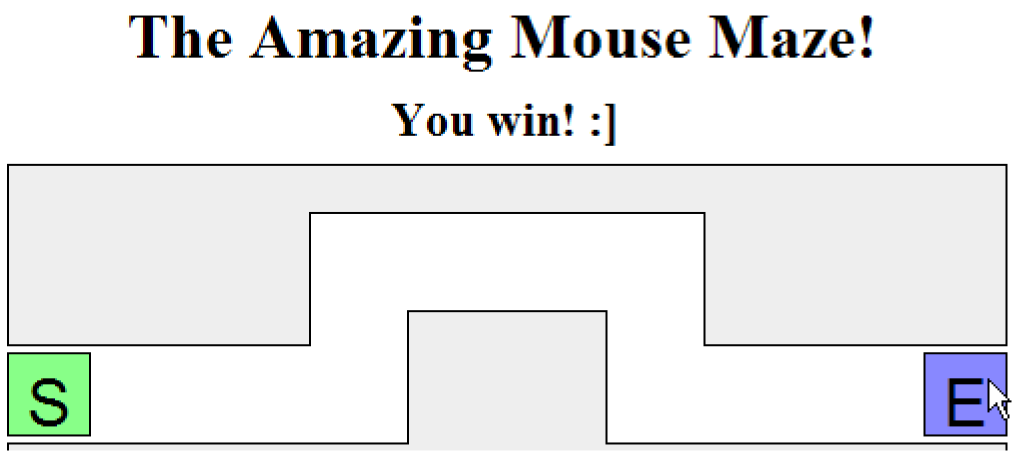
Edition CSE 190M 2009-04-26 (based on Crockford 2008-04-08 edition)

```
window.onload = function() {
    var boundaries = $$("div#maze div.boundary");
    for (var i = 0; i < boundaries.length; i++) {
        boundaries[i].onmouseover = overBoundary;
```

## Exercise : On-page status updates

Instead of an `alert`, make the "You win" and "You lose" messages appear **in the page** itself.

- The page has an (initially empty) `h2` element on the page with an id of status. Put the win/lose text into that div when the user finishes the maze.

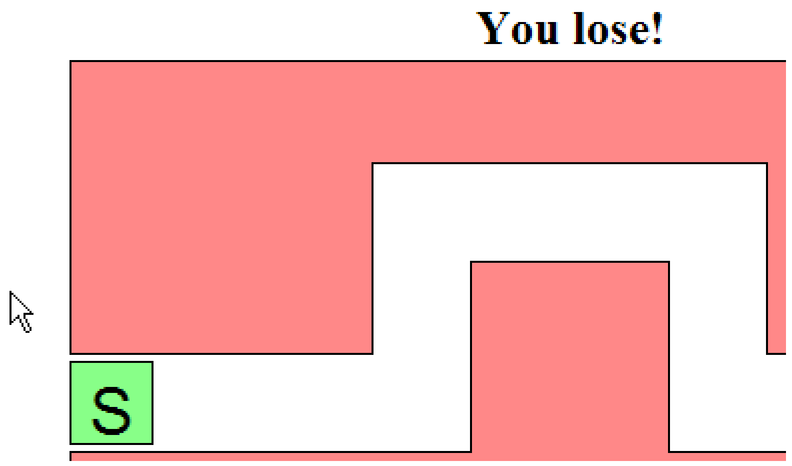

The Amazing Mouse Maze!

You win! :]

## Exercise : Disallow cheating

It's too easy to **cheat**: Just move your mouse around the outside of the maze!

- Fix this by making it so that if the user moves the mouse anywhere outside the maze after clicking the Start area, the walls will light up red and the player will lose the game.
- To do this, you'll need to listen to other kinds of mouse events on other elements.



## Exercise : Additional Features

- Add a timer to the page so that once you start playing the maze, it starts the timer, and stops it when you complete the maze. Pop up the time in an alert message to the user.
- Modify the timer so that instead of popping up an alert, the timer is displayed in the page, and updates every second. When the maze ends, the timer on the page stops.
- Implement a "lives" system - start out the user with 5 lives, and decrement each time they lose.
- Implement the Konami Code - Make the user type "Up Up Down Down Left Right Left Right B A" to unlock 999 lives.