Lecture 8 (7.5?): Javascript
Dynamic web interfaces

- **forms are a limited interface**
  
  ```html
  <FORM METHOD=GET
      ACTION="http://bwk.mycpanel.princeton.edu/cgi-bin/hello1.cgi">
    <INPUT TYPE="submit" value="hello" >
  </FORM>
  ```

  - limited interaction on client side  
    e.g., Javascript for simple validation
  - form data sent to server for processing
  - synchronous exchange with server  
    potentially slow: client blocks waiting for response
  - recreates entire page with what comes back  
    even if it's mostly identical to current content

- **making web interfaces more interactive and responsive**
  
  - "dynamic HTML": HTML + CSS, DOM, Javascript
  - asynchronous partial update: XMLHttpRequest / Ajax
  - plugins like Flash, Silverlight, Quicktime, ...  
  - HTML5 (reduces need for plugins)
Javascript

- **client-side scripting language** (Brendan Eich, Netscape, 1995)
  - C/Java-like syntax
  - weakly typed, basic data types: double, string, array, object
  - object-oriented, very dynamic
    - unusual object model based on prototypes, not classes

- **usage:**
  - `<script> javascript code </script>`
  - `<script src="url "></script>`
  - `<sometag onSomeEvent = 'javascript code'>`

- **can catch events from mouse, keyboard, ...**
- **can access browser's object interface**
  - window object for window itself
  - document object (DOM == document object model) for entities on page
- **can modify ("reflow") a page without completely redrawing it**

- **incompatibilities among browsers**
  - HTML, DOM, Javascript all potentially vary
  - but it's getting much better: ECMA standard is being followed
Javascript constructs

- constants, variables, types
- operators and expressions
- statements, control flow
- functions
- arrays, objects
- libraries
- prototypes
- asynchrony, promises
- etc.
Constants, variables, operators

• constants
  – doubles [no integer], true/false, null
  – ‘string’, “string”,
    no difference between single and double quotes; interprets \ within either
  – 16-bit Unicode characters

• variables
  – hold strings or numbers, as in Awk, but not both simultaneously
    no automatic coercions; interpretation determined by operators and context
  – var declaration (optional; just names the variable; always use it)
  – variables are either global or local to a function
  – only two scopes; block structure does not affect scope
    (changed in newer versions)

• operators
  – mostly like C
  – use === and !== for testing equality
  – string concatenation uses +
  – string[index] but no slices
  – regular expressions in / . . . /
Unicode  (www.unicode.org)

• universal character encoding scheme
  > 120,000 characters

• UTF-16: 16 bit internal representation
  – encodes all characters used in all languages
    numeric value, name, case, directionality, …
  – expansion mechanism for > $2^{16}$ characters

• UTF-8: byte-oriented external form
  – variable-length encoding, self-synchronizing within a couple of bytes
  – ASCII compatible: 7-bit characters occupy 1 byte
    0bbbbbbb
    110bbbb 10bbbbbb
    1110bbbb 10bbbbbb 10bbbbbb
    11110bbb 10bbbbbb 10bbbbbb 10bbbbbb

• Javascript supports Unicode
  – char data type is 16-bit Unicode
  – String data type is 16-bit Unicode chars
  – \uhhhh is Unicode character hhhh (h == hex digit); use in "..." and '!'
Statements, control flow

• statements
  – assignment, control flow, function call, …
  – braces for grouping
  – semicolon terminator is optional (but always use it)
  – // or /* ... */ comments

• control flow almost like C, etc.
  if-else, switch
  while, do-while, break, continue
  for ( ; ; ) …
  for (var in array) …
  try {...} catch(...) {...} finally {...}
Functions

- **functions are objects**
  - can store in variables, pass to functions, return from functions, etc.
  - can be “anonymous” (no name)
  - heavily used for callbacks

```javascript
function name(arg, arg, arg) {
    var ... // local if declared with var; otherwise global
    statements
}

function sum(x, y) { return x + y; }

var sum = function (x, y) { return x + y; }
sum(1,2);
```

- standard libraries for math, strings, regular expressions, date/time, ...
- browser DOM interface: dialog boxes, events, ...
Closures

- A closure is a function that has access to its parent scope, even after the parent function has closed.
  (based on https://www.w3schools.com/js/js_function_closures.asp)

```javascript
var incr = (function () {
    var counter = 0;
    return function () {
        return counter += 1;
    }
})();

incr();
incr();
incr();
incr();
console.log(incr());
```
Objects and arrays

- **object**: compound data type with any number of components
  - very loosely, a cross between a structure and an associative array
- **each property is a name-value pair**
  - accessible as `obj.name` or `obj["name"]`
  - values can be anything, including objects, arrays, functions, …

```javascript
var point = {x:0, y:0, name: "origin"};
point.x = 1; point["y"] = 2;
point.name = "not origin"
```

- **array**: an object with numbered values 0..len-1
  - elements can be any mixture of types
  ```javascript
  var arr = [point, 1, "somewhere", {x:1, y:2}];
  ```
- **array operators**: 
  - sort, reverse, join, push, pop, slice(start, end), …
JSON : Javascript Object Notation  (Douglas Crockford)

- **lightweight data interchange format**
  - based on object literals
  - simpler and clearer than XML, but without checking
  - parsers and generators exist for most languages

- **two basic structures**
  - **object**: unordered collection of name-value pairs (associative array)
    ```
    { string: value, string: value, ... }
    ```
  - **array**: ordered collection of values
    ```
    [ value, value, ... ]
    ```
  - **string** is "..."
  - **value** is string, number, true, false, object or array

- **Javascript eval function can convert this into a data structure:**
  ```
  var obj = eval(json_string)  // bad idea!
  ```
  - potentially unsafe, since the string can contain executable code
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Spectre and Meltdown in the context of Node.js.

Download

8.9.4 LTS
Recommended For Most Users

9.6.1 Current
Latest Features
Formatter in Javascript

```javascript
var fs = require('fs');
var line = ''; var space = '';
var buf = fs.readFileSync(process.argv[2], 'utf-8');
buf = buf.replace(/\n/g, ' ').replace(/ +/, ' ').trim();
words = buf.split(/ +/);
for (i = 0; i < words.length; i++) {
    addword(words[i]);
}
printline();

function addword(w) {
    if (line.length + w.length > 60)
        printline();
    line = line + space + w;
    space = " ";
}
function printline() {
    if (line.length > 0)
        console.log(line);
    line = space = ""
}
```