Dynamic web interfaces

- **forms are a limited interface**

```
<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

- **how can we make web interfaces more interactive and responsive?**
- **dynamic HTML: HTML + CSS, DOM, Javascript**
- **asynchronous partial update: XMLHttpRequest / Ajax**
- **plugins like Flash, Silverlight, Quicktime, ...**

Javascript source materials

- **Bob Dondero's Javascript summary from Spring 2011**
  - http://www.cs.princeton.edu/courses/archive/spring11/cos333/reading/javascriptsummary.pdf
- **"official" Javascript documentation:**
  - https://developer.mozilla.org/en/JavaScript/Reference
  - https://developer.mozilla.org/en/Gecko_DOM_Reference
- **tutorials and reference material:**
  - http://www.w3schools.com/js/
  - http://www.javascriptkit.com/javatutors/index.shtml
  - http://www.functionx.com/javascript
  - http:///www.codecademy.com
- **books:**
  - Javascript, The Definitive Guide (David Flanagan)
  - Javascript: The Good Parts (Douglas Crockford)

Javascript

- **client-side scripting language** (by Brendan Eich at 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 on`***SomeEvent*** `='`***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 change a page without completely redrawing it**
- **lots of incompatibilities among browsers**
  - HTML, DOM, Javascript all potentially vary

Javascript constructs

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

Constants, variables, operators

- **constants**
  - doubles [no integer], `true/false`, `null`
  - `'string'`, `"string"`,
    no difference between single and double quotes; interprets \ within either
- **variables**
  - hold strings or numbers, as in Awk
    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 (!)
- **operators**
  - mostly like C
  - use === and `!==` for testing equality
  - string concatenation uses +
  - regular expressions in `/.../`

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***

```
if-else, switch
while, do-while, break, continue
for ( ; ; ) …
for (var in object) …
try {…} catch(…) {…} finally {…}
```

Example: Find the largest number

```
<html>
<body>
<script>
 var max = 0;
 var num;
 num = prompt("Enter new value, or empty to end");
 while (num != null && num != "") {
    if (parseFloat(num) > max)
        max = num;
    num = prompt("Enter new value, or empty to end");
 }
 alert("Max = " + max);
</script>
</body>
</html>
```

- **needs parseInt or parseFloat to coerce string value to a number**

Functions

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

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

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

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

- **libraries for math, strings, regular expressions, date/time, ...**
- **plus browser interface: dialog boxes, events, ...**

Example: ATM checksum

```
function atm(s) {
   var n = s.length, odd = 1, sum = 0;
   for (i = n-1; i >= 0; i--) {
      if (odd)
         v = parseInt(s.charAt(i));
      else
         v = 2 * parseInt(s.charAt(i));
      if (v > 9)
         v -= 9;
      sum += v;
      odd = 1 - odd;
   }
   if (sum % 10 == 0)
      alert("OK");
   else
      alert("Bad.  Remainder = " + (sum % 10));
}

<form name=F0 onsubmit="">
 <input type=text name=num >
 <input type=button value="ATM"
    onClick='atm(document.forms.F0.num.value);'>
</form>
```

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, …

```
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

```
var arr = [point, 1, “somewhere”, {x:1, y:2}];
```

- **array operators:**
  - sort, reverse, join, push, pop, slice(start, end), …

Object literals

```
var course = {
  dept: "cos",
  numbers: [109, 333],
  prof: {
    name1: "brian", name2: "kernighan",
    office: { bldg: "cs", room: "311" },
    email: "bwk"
  },
  toString: function() {
    return this.dept + this.numbers + " "
       + this.prof.name1 + " " +  this.prof.name2 + " "
       + this.prof.office.bldg + this.prof.office.room
       + " " + this.prof.email;
  }
}
```

JSON : Javascript Object Notation (Douglas Crockford)

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

- **see json.org**

Prototype property

- each object has a prototype property that is used to make new instances
- changing the prototype affects all subsequent ones

```
function Point(x,y) {
   this.x = x; this.y = y;
}
Point.prototype.dist = function(that) {
   var dx = this.x - that.x;
   var dy = this.y - that.y;
   return Math.sqrt(dx*dx+dy*dy);
}
Point.prototype.toString = function() {
  return '(' + this.x + "," + this.y + ')';
}
Point.ORIGIN = new Point(0,0);
var p = new Point(3,4);
var d = p.dist(Point.ORIGIN);
var msg = "Dist to " + p + " is " + d;
```

Formatter in 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]);
}
println();

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 = ""
}
```