LING 408/508: Programming for Linguists

Lecture 11
October 5th
Topics

• Web:
  – HTML: page content and structure
  – CSS: page presentation
  – Javascript
    • programming language inside your browser
  – DOM: Document Object Model
    • allows a Javascript program to read (and modify) your page content
Debugging

• In Safari:
  – Preferences > Advanced
  > Show Develop menu in menu bar
Debugging

• In the Develop menu:
  – Allow JavaScript from Smart Search Field
  – javascript: ok in address bar
  – Show Page Source
  – Show Web Inspector
Pop-up notification

- For browser programming:
  - `alert(string)`
  - `no print function`
- Example (from address bar):
  - `javascript:alert(4+5)`

- Changing the html document:
  - `document.write(string)`  \(\text{overwrites html document}\)
  - `html_element.innerHTML = "string"`  \(\text{modifies html_element}\)
Pop-up notification

- Google Chrome
Firefox
Javascript

• Javascript:
  – invented in 1995 as a browser programming language
  – not the same as Java (also appeared in 1995)

• Javascript code:
  – <script> .. </script>
  – <script src="filename.js"></script>

• HTML element events:
  – onclick="js code"
  – onmouseover="js code"
  – onkeydown="js code"

• Reference:
  • http://www.w3schools.com/jsref/dom_obj_event.asp

  code snippets can be placed in
  <head>..</head>
  or
  <body>..</body>
Variables

- **Declared variables:**
  - `var x = 9;` // number variable
  - `var name = "String";` // string variable
  - **Note:** names are case sensitive, no $ like the bash shell, // for in-line comments

- **function local variables:**
  - `function f() {
    var x = 99;
    }
  }

- **Not declared:**
  - i.e. just used without declaration
  - treated as a global variable

- **Strict:** prevents use of undeclared variables etc.
  - "use strict"; quotes for backwards compatibility with older versions

- **Comments:**
  - `//` until the end of the line
  - `/* ... */` multiline (same as CSS)
Strings

• Use either **double quotes** or **single quotes**
  – escape using backslash, e.g. ", \n, \\
• Properties:
  – "string".length
• Methods (like a function but object-based):
  – .indexOf(string) returns position (0...length), -1 if not found
  – .search(regex) returns position
  – .replace(regex,string) replaces with string, returns result
  – .slice(index,index) substring from position to position
  – .substr(index,length) substring
  – .toLowerCase() case conversion
  – .toUpperCase() case conversion
  – .trim() remove whitespace from both ends
  – .charAt(index) single character at position
  – .split(separator) returns array, single characters if separator=""
Numbers

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

<table>
<thead>
<tr>
<th>Value (aka Fraction/Mantissa)</th>
<th>Exponent</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>52 bits (0 - 51)</td>
<td>11 bits (52 - 62)</td>
<td>1 bit (63)</td>
</tr>
</tbody>
</table>

- Recall the discussion in lectures at the beginning of the semester about number representation?
  - Integers are considered accurate up to 15 digits.
Numbers

• Pi to 20 decimal places
  – 3.14159 26535 89793 23846
  – 3.14159 26535 89793 (Javascript)
Operators

Arithmetic Operators:
• +, -, *, /, % (mod), ++, --

Assignment Operators:
• =, +=, -=, *= /=, %=

String Operators:
• +  concatenation
• +=  append to string

Comparison Operators:
• ==  can be made equal (type coercion)
• !=  not equal
• ===  same type and equal (no type coercion)
• !==  not equal (no type coercion)
• >, <, >=, <=
• &&  logical and  (Javascript: Boolean true/false)
• ||  logical or
• !   negation

Note: & , !, ~ (not), ^ (xor) are bitwise operators

```javascript
alert("string" + 5)
```
produces string5
Conditionals

• if-then
  – if (condition) { ... } else if (condition) { ... } else { ... }

• switch
  – switch (expression) { case value: ... break; ... default: }

Idea: compute expression first
2nd stage: compare computed value with each case
Loops

• for loop (just like C):
  – example:
    • for (i = 0; i < 100; i++) { ... }

• for/in loop (object properties):
  – for (x in object) { ... }

• while loop:
  – example:
    • while (true) { ... }

• do/while loop (like traditional repeat/until):
  – do { ... } while (condition)

• loop exit:
  – break (jump out of loop immediately)
  – continue (skip rest of current iteration)
Miscellaneous

• Random number $[0, n-1]$:  
  – Math.floor(Math.random() * n)

• Swapping two variables (normally):  
  – var a; var b; var temp;  
  – temp = a;  
  – a = b;  
  – b = temp;

• Javascript arithmetic (Calculation priority):  
  – var a; var b;  
  – b = a + (a = b) - b;  
  – a = b + (b = a, 0);

Arithmetic priority: 
Set a to the value of b first,  
(a = b) evaluates to b,  
then evaluate rest of expression
DOM
Document Object Model (DOM)

- HTML document

```html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<title></title>
</head>
<body>
<h1></h1>
<br/>
<br/>
<br/>
<br/>
<br/>
</body>
</html>
```

- Tree representation

```
html: document.documentElement
body: document.body
```
Document Object Model (DOM)

Properties for traversing the DOM:
• `e.childNodes`
  – children of element el as an array, e.g. `childNodes[0]`
• `e.children`
  – element nodes only (excludes text nodes)
• `e.firstChild`
• `e.lastChild`
• `e.parentNode`
• `e.nextSibling`
• `e.previousSibling`

Object properties:
• `e.nodeType`
  – 1 = element, 3 = text
• `e.nodeName`
  – uppercase
• `e.innerHTML`
  – for element nodes
  – value is html as text
  – writeable
• `e.nodeValue`
  – for text nodes
  (null: for element nodes)
  – writeable
New content:
• `document.createElement(tag)`
  – `tag` = ‘div’, ‘p’ etc.
  – creates new DOM element
• `document.createTextNode(text)`
  – creates new DOM element of type text

For non-HTML elements:
• `document.createElementNS(NS,tag)`
  – `NS` = Namespace URL identifier
  – e.g. [http://www.w3.org/2000/svg](http://www.w3.org/2000/svg) and tag “rect” (rectangle)

Place new_el:
• `e.appendChild(new_el)`
  – `new_el` is inserted as last child of `el`
• `e.insertBefore(new_el,next_el)`
  – `new_el` inserted as previous sibling of `next_el`
  – `el` is common parent
• `e.removeChild(child_el)`
  – `child_el` is deleted
  – `el` is parent
• `e.replaceChild(new_el,child_el)`
  – `new_el` replaces `child_el`
  – `el` is parent

Old way:
• `document.write(text)`
• `document.writeln(text)`
  – adds a newline
**Document Object Model (DOM)**

Locating an element:
- `document.getElementById(id)`
  - useful if you have named the document element using the id='Name' property
- `document.getElementsByTagName(tag)`
  - all document elements of type tag
  - returns an array
- `e.getElementsByTagName(tag)`
  - all elements of type tag under el
  - returns an array
- `document.getElementsByName(name)`
  - useful for elements that support name='Name'
- `(document|e).getElementsByClassName(class)`
- `(document|e).querySelector(query)`
  - example query ‘BODY > UL > LI’
  - ‘>’ means immediately dominates
  - returns first matching element
- `(document|e).querySelectorAll(query)`
  - returns an array