Software Engineering
ECE444

Client-side Processing

Recall: HTML

- Presentation language that describes structure and content

```
<HTML>
  <head>
    <title> . . . </title>
    . . .
  </head>
  <body>
    <h1> . . . level 1 header . . </h1>
    <p> . . . paragraph . . . </p>
    <img> . . . image . . . </img>
  </body>
</HTML>
```

Recall: XML

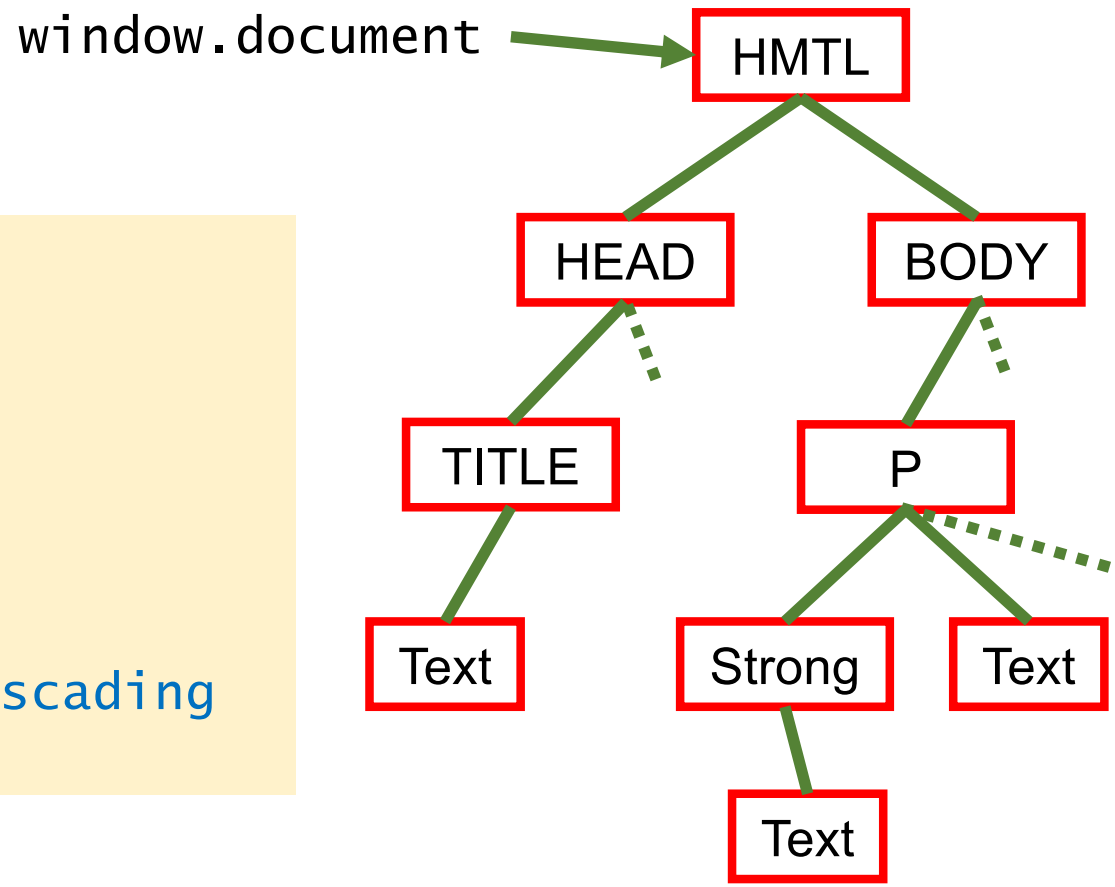
- an HTML-generalized data description language

```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>
    Two of our famous Belgian Waffles with plenty of real maple syrup
    </description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>
    Light Belgian waffles covered with strawberries and whipped cream
    </description>
    <calories>900</calories>
  </food>
  . . .
</breakfast_menu>
```

Recall: DOM

- a platform and language-neutral interface to allow programs to dynamically access and update documents w.r.t content, structure, andstyle

```
<HTML>
  <HEAD>
    <TITLE> .. </TITLE>
    . . .
  </HEAD>
  <BODY>
    <P>
      <strong>C</strong>ascading
      . . .
```



Recall: CSS

- Selectors:

`h1`	any `h1` element
`div#message`	`div` with `id="message"`
a.lnk	`a` element with `class="lnk"`
`.red`	any element with `class="red"`
`div.red, h1`	`div` with `class="red"` or any `h1`
`div#message h1`	`h1` element that is child of `div#message`
`a.lnk:hover`	"pseudo class": `a.lnk` when hovered over

- and properties; e.g.,

```
body { background-color: lightblue; }
h1 {
    color: white;
    text-align: center;
}
p {
    font-family: verdana;
    font-size: 12px;
}
```

Javascript

- dynamic, interpreted scripting language built into all modern browsers
- unrelated to Java (LiveScript → JavaScript to get traction)
- Bad reputation
  - many download, copy and modify poor code
  - incompatibilities between interpreter implementations
  - browsers have restricted dev environments

(I'm not a fan… but its popularity is increasing… and you have no choice!)

- Three things you need to know for your JS interview:
  - meaning of `===` operator and how it is different than `==`
  - closures
  - the real meaning of `this`

Javascript uses

- to enhance user experience:
  - client-side JS works together with HTML and CSS:
  - can interpret "events" like typing, mouse over, mouse movement, etc. and take app-specific actions to change the DOM
  - client-side checking of form input
- AJAX: Asynchronous JS and XML:
  - make HTTP requests to Web server without triggering page reload, then use returned info to change DOM
  - goal: more responsive user experience
  - create single-page apps
- Client-side apps like Google Docs.
  - as complex as server-side apps, if not more so
  - e.g., Angular framework uses MVC architecture
- Server-side apps
  - e.g., node.js: popular server-side JS framework

Some Javascript language features

- supports meta-programming and introspection
- typing is dynamic
- everything is an object
- object looks like a Ruby hash: KV-pairs
  (except keys must be strings or valid JS var names)

```
var person = {fName:"John", lName:"Doe", age: 50}
```

  or define object with constructor:


```
function person( fn, ln, age ) {
    this.fName=fn ;
    this.lName=ln ;
    this.age=age ;
}
```

  then `var student = new person("Suzy", "Hacker", 34)`

- JS doesn't really have classes

Functions are closures

- Functions are *first class objects* & *closures*
  - A function is a lambda expression

```
var make_times = function(mul) {
  return function(arg) { return arg * mul; }
}
// or:  function make_times(mul) { ... }

times2 = make_times(2)
times3 = make_times(3)
times2(5) ➔ 10
times3.call(null, 5) ➔ 15
```

- A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment).
- ➔ A closure gives you access to an outer function's scope from an inner function.
- In JavaScript, closures are created every time a function is created, at function creation time.

Basic JavaScript Constructs I

- Objects
  - like a hash; can be nested
    `stud = {name:{fname:"Billy", lname:"Jean"}, age: 50,…}`
  - access with `stud.age` or `stud[age]` (if property name not legal or not def'd until runtime).
  - `for( var in stud ) { . . . }` iterator
- Types
  - objs have types, vars do not
  - `typeof x` returns string representation of type: "`object`", "`array`", "`boolean`", "`function`", "`undefined`"
- Arrays `var a = [1, {two: 2}, 'three']; a[0] == 1 ;`
- Numbers `+ - / % +=… ++ -- Math.round(n), Math.ceil(n) …`
- Control flow `while(), for(;;), if…else switch/case return`
- Naming `localVar, local_var, ConstructorFunction, GLOBAL`

JavaScript Pitfalls I

- interpreter inserts ';' you might have forgotten
  → sometimes guess wrong → unexpected results
- Syntax suggests block scope, but not true; e.g.,
  `for(i=0; i<10; i++) { var m; … }`: `m` is visible to entire fct.
- Array is just an obj with integer keys
  → a[2.1] becomes a["2.1"]
- `==` and `!=` perform type conversions automatically
  → `'5' == 5.0` is true!
  but `'5' === 5.0` is false (different than Ruby's `===`)
- Equality for arrays and hashes based on identity, not value. → `[1,2,3]==[1,2,3]` is false

JavaScript Pitfalls I|

- Beware: JavaScript doesn't have classes:

```
var Student = function( fn, ln, age ) {
  this.fname = fn ;
  this.lname = ln ;
  this.age = age ;
  this.full_name = function() // "instance method"
     return( this.fname + " " + this.lname ) ;
} ;
function Student( fn, ln, age ) { // looks familiar, eh?
  this.fname = fn ;
  . . .
}
// 'new' creates new instance
sue = new Student( 'Suzy', 'Smith', 98 ) ;
sue.full_name ;     // => function(){…}
sue.full_name() ;   // => "Suzy Smith"
// BAD: without 'new', 'this' bound to global object, not inst.
suzy = Student( ' 'Suzy', 'Smith', 98 ) ;
suzy ;              // undefined
suzy.age ;          // error: undefined has no properties
suzy.age() ; // error: undefined has no properties
```

Use this:
- can pass it around

declared in glob namespace

With 'new', 'this' refers to instance.
Without 'new', function returns nothing

Including JavaScript in HTML page

- within <script> HTML tag in header:

```
<head>
   <title>Today's Date</title>
      <script type="text/javascript">
         let d = new Date();
         alert("Today's date is " + d);
      </script>
 </head>
```

- within <script> HTML tag in body:

```
<body>
   <script type="text/javascript">
      let d = new Date();
      document.body.innerHTML =
                  "<h1>Today's date is " + d + "</h1>"
   </script>
</body>
```

- include external source file in header:

```
<script type="text/javascript"
         src="path-to-javascript-file.js"></script>
```

Including JavaScript in HTML page II

if you place one or more .js files in app/assets/javascripts

then Rails:

1. concatinates all JS files in directory
2. compresses result
3. places result in public subdirectlry

- use `javasript_include_tag 'app'` to generate appropriate tag

JSAPI

- Interface between JavaScript and DOM
- with JavaScript: global var: `window`
  - exists for each loaded page (can't share data across pages)
  - key property: `window.document` -- root element of DOM
  - other properties to query, traverse, modify DOM, etc.
  - E.g.,

```
const list = document.getElementById('list1');
const children = list.childNodes;
for( let i=0; i<children.length; i++) {
   if( children[i].id === 'three' ){   //remove
      children[i].parentNode.removeChild( children[i] ) ;
   }
   console.log(children[i].nodeName);
}
```

- However: huge compatibility problems!!! → unusable! (see quirksmode.org/) ➔ use JQuery instead

Trivial example

```
<!DOCTYPE html>
<html>
  <head>
    <script>
       function myFunction() {
          document.getElementById("demo").innerHTML =
                                 "Paragraph changed.";
       }
    </script>
  </head>
  <body>
     <h1>A Web Page</h1>
     <p id="demo">A Paragraph</p>
     <button type="button" onclick="myFunction()">
                 Try it</button>
  </body>
</html>
```

More convoluted example I

```
<H1>Michael Stumm: Selected Papers</H1>
Show:
<form action="">
   <input type="checkbox" name="cat" value="OS" checked
                                onclick="filter();"/>Operating Systems 
   <input type="checkbox" name="cat" value="SE" checked
                                onclick="filter();"/>Software Engineering 
   . . .
</form>

<p id="p1" class="filter-OS">
        <strong>The scariest paper ever: OS</strong><br/>
        Reza Azimi, David Tam and Michael Stumm,<br/>
        In <em>Proceedings of the Funniest Conference Ever</em>,
        Anaheim, CA, ACM, New York, NY, USA, January, 2013, pp. 45-55.
</p>
<p id="p2" class="filter-SE">
        <strong>The sexiest paper ever: PC</strong><br/>
        Reza Azimi, David Tam and Michael Stumm,<br/>
        In <em>Proceedings of the Worst Conference Ever</em>,
        San Francisco, CA, Usenix, New York, NY, USA, January, 2018, pp. 45-55.
</p>
<p id="p3" class="filter-OS filter-SE"> . . .
```

More convoluted example II

```
function filter() {
   var chk_arr =  new Array() ;
   chk_arr = document.getElementsByName("cat");
   var chklength = chk_arr.length;

   /* now go through all the papers and hide those whose category is not checked */
   var papers = new Array() ;
   papers = document.getElementsByTagName('p') ;

   for( var i=0; i<papers.length; i++ ) {
       var hidden = true ;
       for( var j=0; j<chklength; j++ ) {
           if( chk_arr[j].checked )
               if( papers[i].className.match(chk_arr[j].value)) {
                     hidden = false ;
                     papers[i].style.display = "block" ;
                     break ;
               }
           }
        }
        if( hidden == true ) {
           papers[i].style.display = "none" ;
        }
   }
}
```

JavaScript Recommendations

- To deal with compatibility issues:
  - restrict yourself to language features in ECMAScript 3 standard, which all browsers support
  - use jQuery library (described later) to interact with HTML docs
- Your Web pages should give a good experience even if JavaScript not supported or disabled.
- JavaScript code should be kept completely separate from page markups – separation of concerns
- Avoid namespace clutter: create one object with one name associated with your app, and make all functions be values of properties of this one object.

JSON: JavaScript Object Notation

- Language independent way to represent data
- For exchanging data between browser and server
- The most popular data exchange format to "serialize"/"marshal" internal data formats.
- Similar to XML in concept, but
  - no end tags
  - shorter → faster
  - quicker and easier to read and write
  - can use arrays (where order matters)

JSON II

- Similar to JS object def, but keys must be strings; e.g.,

```
{
    "number Employees": 7,
    "employees": [
        {"fname": "Suzy", "lname": "Smith", "age": 98, . . . },
        {"fname": "John", "lname": "Doe", "age": 23, . . . },
        . . .
        { . . .}
    ]
 }
```

- JS Conversion functions:

```
var myObj = {fname: "John", lname: "Johny", age: 31, …} ;
var myJSON = JSON.stringify( myObj ) ;

var myJSON = {"fname": "John", "lname": "Johny", "age": 31, … ];
var myObj = JSON.pars( myJSON ) ;
myObj.fname ;  // valid: "John"
```

JQuery

- A powerful framework for DOM manipulation
  - adds many useful features over browsers' built-in JSAPI
  - homogenizes incompatible JSAPI's across browsers
- used by Google, IBM, Netflix, and just about everyone else
- has exactly one function: `jQuery()`, aliased as `$()`:

```
$(selector).action()
```

to be performed on elements

as in CSS to find HTML elements

E.g.,

```
$(this).hide()      // hides current elem
$("p").hide()       // hides all <p> elems
$(".text").hide()   // hides all elems with class="test"
$("#test").hide()   // hides all elems with id=="test"
```

JQuery II

- `$()` returns node set with each element wrapped in jQuery's DOM element representation
  - not an array: `$()[0]` won't work
  - use `each` iterator instead
- elem representation gives it abilities beyond JSAPI:
  - `is()` // test if `elem :checked, :selected, :enabled, …`
  - `addClass()`, `removeClass(), hasClass() // CSS classes`
  - `css()` // e.g., `css("color", "red")` or `css()` to query
  - `insertBefore()`, `insertAfter()`
  - `remove()`
  - `clone()`
  - `val()` // e.g., value of a form element
  - `hide()`, `show()`, `toggle()`, `fadeOut()`
  - `html()`, `text()` // query or set content
  - `attr()` // e.g., `$("img").attr("src", http://imgur.com/xyz)`
  - many, many more…

Jquery III

- Select with multiple selectors:
  `$('p  .myclass')`
- Wrap DOM elements to give them secret jQuery powers:
  `this` ➔ `$(this)`
  `document.window` ➔ `$(document.window)`
- Create elements:
  `var elt = $("<span>Hola, mundo</span>")`
- Run a function when document ready:
  `$(RP.setupFunc)`
- Chaining: do one after the other; e.g.,

```
$("#p1").css("color","red").slideup(2000).slid
edown(2000)
```

JQuery events

- JSAPI allows attaching Javascript event handlers to the user interface
  - when user performs certain actions on a DOM element a designated JS function is called.
  - actions on any element:
    - click, dblclick, mousedown/mouseup, mouseenter/mouseleave, keypress, focus/blur, focusin/focusout
  - actions on user-editable controls (forms, checkboxes, radio buttons, text boxes, text fields, menus):
    - change, select, submit
- JQuery makes it convenient to bind handler to action: e.g.,

```
$("#p1".mouseenter(function() {
        alert( "you entered p1") ;
        // JS "this" would refer to elem #p1
}) ;
```

JQuery widgets

- Use for fancy interfaces with little work:
  - Datepicker
  - Acordion
  - Autocomplete
  - Menu
  - Progressbar
  - Slider
  - Spinner
  - Tabs
  - Tooltips
  - etc.

Regular expressions (Regex)

- strings defining string patterns
- Regex appear between slashes; e.g.

0 or 1 of prev | 0 or 1 of prev | any of | ignore case

```
r = "/^\d\d?:\d\d\s*[ap]m$/i"
```

beg of string | digit | white space | end of string

```
x = "8:45pm" ; x =~ r  # → 0  (pos in x were r matched)
y = " 8:45pm"; y=~r    # → nil
"08:45  pm" =~ r       # -> 0
"47:45am" =~ r         # -> 0
```

Regex symbols

.	any char	
[...]	any char in set	[ap] [a-z] [0-9]
[^...]	any char not in set	[^0-9]
\d	digit	== [0-9]
\D	non-digit	== [^0-9]
\s	white space	
\S	non-white space	
\w	word char	
\W	non-word char	
*	zero or more (of previous)	
+	one or more (of previous)	
?	zero or one (of previous	
|	or	(It's|it is)
^ $	beginning of line end of line	

Regex grouping

```
r = "/^(\d\d)?(\d\d)\s*([ap]m)$/i"
```

$1 $2 $3

```
x =~ r
puts $1    # → 8
puts $2    # → 45
puts $3    # → pm
```

Email regex

- Simplified:

```
r = "/^[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+$/i"
```

- Reality in Ruby:

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\"
.\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@
,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r
\n)?[\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?
[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t]
)*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]
|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@
,;:\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?
:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]|\\.
)*\](?:(?:\r\n)?[\t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+
|\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:
[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[\t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]|\\.)*
\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|
\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[
^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|"(?:[^\"\r\\]|\\.|(?
:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t
])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000
-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)
)*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\"
.\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()
<>@,;:\".\[\]\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:
\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\".\[\] \000-\031]+(?:(?:(?:\
r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)
)*@(?:(?:\r\n)?[ \t]
```

That's about 1/3 of it… See emailregex.com

- Use: `URI::MailTo::EMAIL_REGEXP`