Software Engineering
ECE444

Client-side Processing

Seven additional languages

- HTML
- XML
- DOM
- CSS
- Javascript
- JSAPI
- JSON
- JQuery
- . . .

HTML I

- Markup language to represent a Web page
- Presentation Language
- Describes structure and content
- Key idea: let browser determine look and feel
- Tags content with HTML elements; e.g.

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

HTML II

- Additional elements include:
  - headers <h1> . . <h6>
  - table <table>
  - forms <form> with many elements: input, radio button, etc.
  - blockquote <blockquote> + code <code>
  - images <img> and videos <video>
  - unordered and ordered lists <ul> + elements
  - frames <frame>
  - structural ,
  - horizontal line/ thematic break <HB>, line break
  - links <a>
- Content attributes:
  , , <b>, <i>, <big>, ,
  <id>, <class> . . .

HTML III

- You can control much of the layout and look & feel with HTML;
  e.g., using tables with lots of attributes

- You should resist the temptation!

  Just use HTML to define the structure of the document/Web page

- If you are unfamiliar with HTML basics:
  - consult https://www.w3schools.com/html/ and
  - create a simple small Web page

XML

- a data description language
- generalization of HTML
- but allows you to define your own tags
  ➔ much more flexible

```
<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>
```

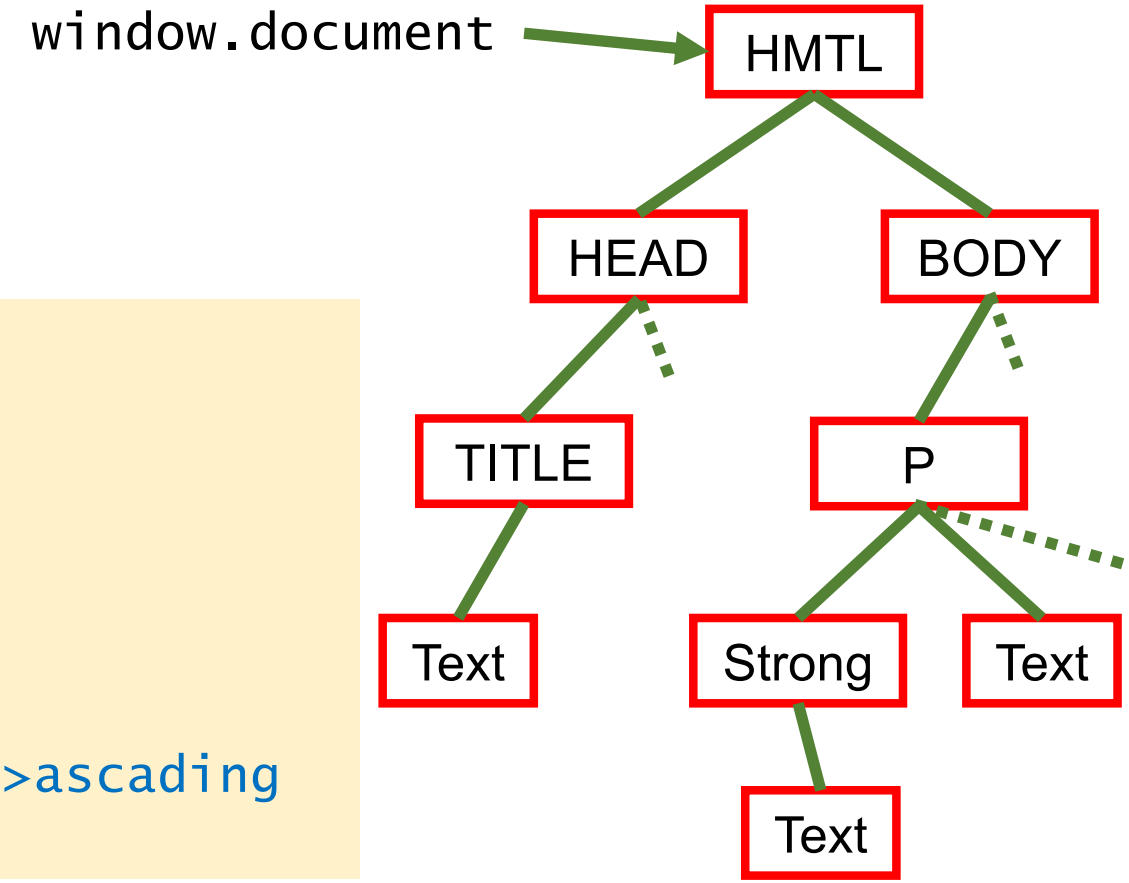
DOM: Document Object Model

- a WWW Consortium standard
- a platform and language-neutral interface to allow programs to dynamically access and update documents
  - content,
  - structure, and
  - style

DOM II

- When a browser loads an HTML tree, the page is parsed into a DOM tree:

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



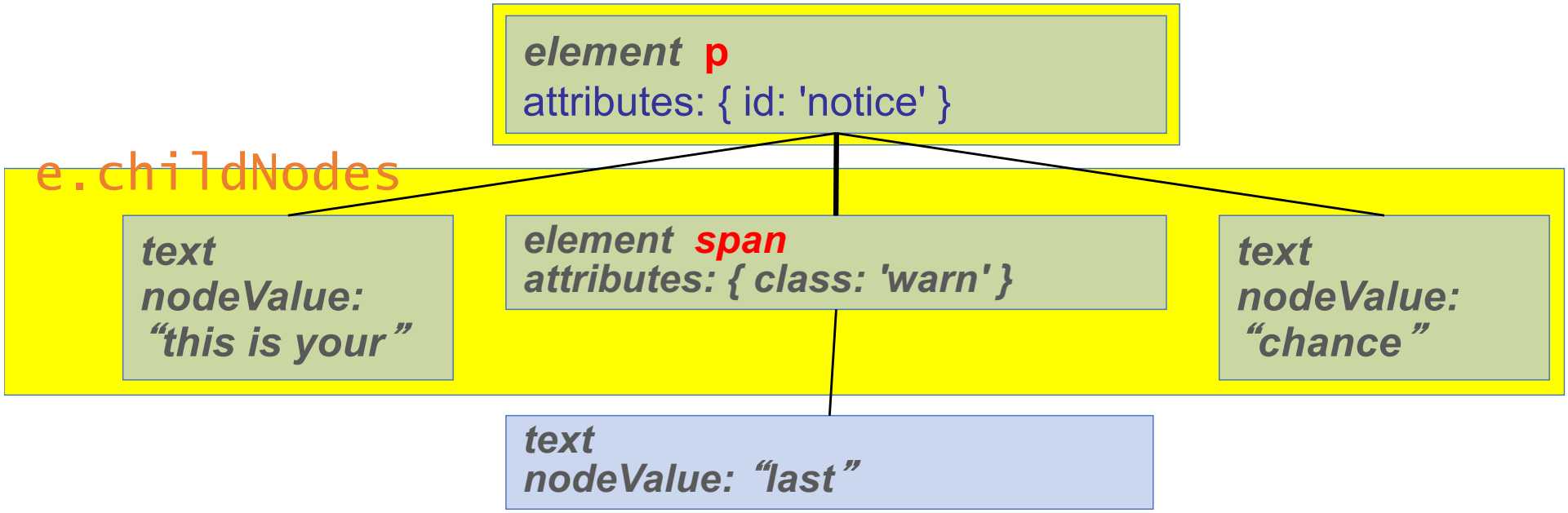
DOM III

- More detailed example:

```
<p id="notice">
 This is your <span class="warn">last</span> chance!
</p>
```

- Javascript routines are available to search and navigate tree

```
e = document.getElementById('notice)';
```



DOM IV

- HTML is parsed by browser layout engines; e.g.,
  - Gecko
    - Firefox
  - Blink
    - Chrome
    - Opera
    - Microsoft
  - Webkit
    - Safari
    - all browsers that run on IOS

CSS: Cascading Style Sheets

- language that allows us to associate styling and layout instructions with HTML elements

- Basic idea: separate styling and layout (CSS) from structure (HTML)

- E.g.,

selector attribute value

```
strong {color: red;}
```

- E.g.,

```
p { color: blue; text_decoration: underline }
```

declaration declaration

property

CSS II

- HTML attributes `class` and `id` are often used with CSS

  `<p id="fred">` or `<p class="important">`

  must be unique

  many elements may share same class
  even if element tag is different

- Examples of 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

CSS III

- Note that HTML and class typically used for CSS purposes
- Further CSS example:

```
body { background-color: lightblue; }

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 12px;
}
```

See: https://www.w3schools.com/css/

recommended

- You can place CSS in doc <head>, inline, or use separate file:

```
<head>
  . . .
  <link rel="stylesheet" type="text.css" href="style.css">
  . . .
</head>
```

CSS IV

- Final style for an element can come from several places (that interact in complex ways):
  - browser's default styles
  - styles specified by end-user; e.g.,
    - set in Preferences
    - userContent.css (for Firefox)
  - styles linked to document by author
    - external file
    - at beginning of doc
    - within element

- Styles are inherited; e.g., for

```
<body><p><strong>C</strong>ascading . . . </p> . . .
```

  style of "C" is a combination of
  - browser strong style
  - CSS strong def
  - inherited style of <p>
  - inherited style of <body> . . .

➔ use tools such as Dom Inspector (Firefox) to see what exactly is going on

- Recommendation: use Bootstrap CSS Framework (Twitter)

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)

```
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 no
  - `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

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.

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), so ➔ use JQuery instead

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": 3,
    "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"
```