Web Applications

CSE183

Fall 2020

Introduction

Baskin Engineering UC SANTA CRUZ

Notices

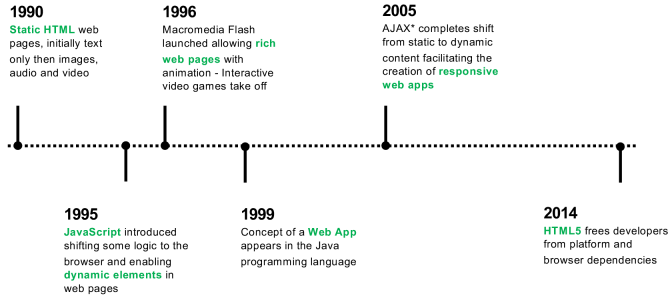
- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 1** due 23:59 **Thursday October 8**

Today's Lecture

- Introduction
- Resources
- Course Outline
- Assessment
- Grading
- Academic Integrity

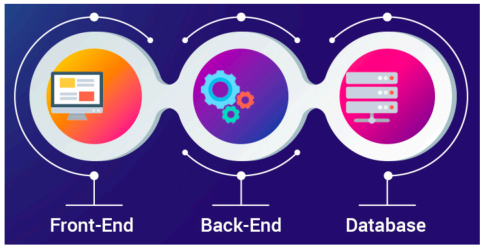
- Assignment 1
- Questions

Early History of Web Applications



* Asynchronous **JavaScript** and **XML**

Full Stack Web Applications



https://nigeriainformatics.com/wp-content/uploads/2019/04/Full-Stack-Development-Services-1.jpg

UC Berkeley Extension Academic Areas Student Services Blog Berkeley Global

Introduction to Full Stack Web Development
COMPSCI X443.1

Learn the fundamentals of full-stack web development using some of the most popular open-source libraries. The stack is comprised of Angular.js/Handlebars (front-end), Node.js/Express (back-end) and MongoDB (NoSQL), MySQL or Postgres (Relational) (DB technologies). In addition, you learn HTML5/CSS/JavaScript/jQuery and DevOps/Software Development skills. Full-stack web development is an important skill set to build fully-functional web applications. Learn more by e-mailing the instructor at mkremer@berkeley.edu.

coursera Explore What do you want to learn?

Browse > Computer Science > Mobile and Web Development

Full-Stack Web Development with React Specialization

Build Complete Web and Hybrid Mobile Solutions. Master front-end web, hybrid mobile app and server-side development in four comprehensive courses.

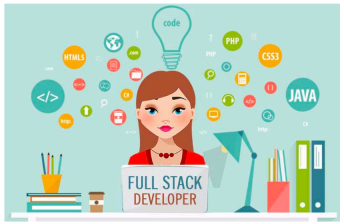
4.7 9,606 ratings

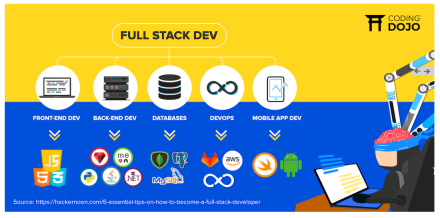
Jogesh K. Muppala

Enroll for Free
Starts Sep 24

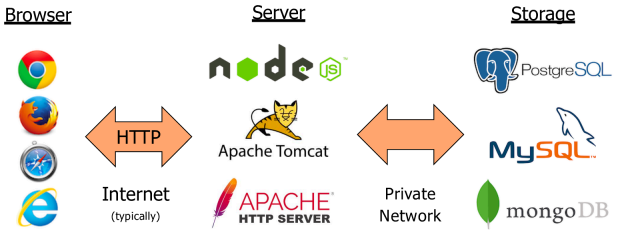
Financial aid available

76,713 already enrolled





Full Stack Web Applications



Software Requirements

- A Modern Browser
  - Chrome strongly recommended, Firefox is good too
  - Try to avoid Internet Explorer and Safari
- Node.js
  - JavaScript runtime environment
  - Installs easily on Windows, MacOS, Linux
  - Includes `npm` (Node Package Manager)
- PostgreSQL
  - "The worlds most advanced open source database"
  - Derived from UC Berkeley Ingres project
  - Team Leader (Michael Stonebreaker) won Turing Award in 2014

The NERP* Stack

Node.js	JavaScript runtime environment
Express.js	Web Application framework for Node.js
React	JavaScript library for building user interfaces
PostgreSQL	Database management system

* I totally made that up, not a recognized acronym ☺

Resources

- People
  - David (Instructor) dcharris@ucsc.edu
  - Alex (TA) arinaldi@ucsc.edu
  - Hayden (TA) hchen222@ucsc.edu
- Course pages
  - https://canvas.ucsc.edu/courses/36577
  - Lecture handouts & Zoom recordings available shortly after each lecture
- Instructor office hours
  - 16:30 to 18:00 every weekday, excluding holidays
  - Zoom link on Canvas
  - Waiting room if it gets crowded
  - Arrive prepared
  - Invite your study buddies to join at the same time

Course Outline

- **Browsers**
  - Markup, separation of content & style, reuse, scripting
  - The Document Object Model
- **User Interface**
  - Model View Controller, React, Single Page Applications, Responsive Web Design
- **App Servers**
  - Browser/Server Comms., Web Servers, Node.js, Express.js
- **Storage**
  - PostgreSQL, Cookies, Session Data
- **Miscellaneous**
  - Input Validation, State Management, Security, Scaling

Assessment

Administration x 2		**(not graded, mandatory)**
Individual Assignments x 8	**50%**	**(4% x 2 & 7% x 6, 1 week each)**
Group Assignment	**20%**	**(3 weeks)**
In-Class Quizzes x 5	**10%**	**(2%, 25 minutes each)**
Final Examination	**20%**	**(3 hours in finals week)**

NOTE: Administration tasks are mandatory, and you must pass every component (assignments, quizzes, and examinations) to pass the class. E.g., doing well on the final and quizzes but submitting too many poor assignments will see you fail the class.

HOWEVER: Doing badly on one or two assignments or quizzes but well on the others, is fine so long as the aggregate grade for the assignments/quizzes is a pass.

REMEMBER: If you do not pass the final, you do not pass the class.

HOWEVER: Failing any single quiz or assignment is *not* an instant fail.

Assignments, Quizzes, Final

- **Individual Assignments**
  - Single new concept coding exercises
- **Group Assignment**
  - Multi concept "realistic" full stack Web app
  - Teams of two
  - Equitable distribution of work must be demonstrated
- **Quizzes**
  - Small number of small questions
  - Reinforces lecture material
  - Encourages you to produce your own study guides
- **Final**
  - Larger number of larger questions
  - Sample examination with solutions provided two weeks before final

Submissions

- **Individual Assignments**
  - Released 08:00 Fridays
  - Due 23:59 Thursdays
- **Group Assignment**
  - Released 08:00 Friday November 20
  - Due 23:59 Thursday December 10
- **In-Class Quizzes**
  - Approximately every second Friday
  - During lecture slot
  - Random start times
  - Randomized Canvas Quiz
  - Zoom Proctoring

Grade Bands

Grade	Range	Characterisation
A+	**95-100**	**Outstanding**
A	**88-94**	**Excellent**
A-	**81-87**	**Excellent in most respects**
B+	**75-80**	**Very good**
B	**70-74**	**Good**
B-	**65-69**	**Good overall, but some weaknesses**
C+	**60-64**	**Satisfactory to good**
C	**55-59**	**Satisfactory**
* C-	50-54	Adequate evidence of learning
* D	45-49	Some evidence of learning
F	0-44	Below the required standard; fail

* Pass, but cannot be used to satisfy a major requirement or a general education requirement, and cannot satisfy a prerequisite for another course. https://registrar.ucsc.edu/navigator/section4/performance/letter-grades.html

Academic Integrity

- Assignments are individual tasks, but…
  - We encourage you to work, study, and think together
  - But ensure the final product is yours and yours alone
- All course work is submitted electronically
  - No late submissions allowed
  - Checked for collaboration & plagiarism
    - If you copy another student, you fail the assignment ☹
    - If you copy a solution found on the web, you fail the assignment ☹
    - In both cases you may get kicked off the class, or even out of school ☹ ☹
- Examinations (quizzes and final) are open book
  - Laptop camera proctoring
  - Canvas quiz monitoring
  - Randomised question selection

Assignment 1

- Approximately 1,500-word paper
- "A Brief History of Web Applications"
  - Entirely Free format
    - Idea is for you to do some research
  - Possible sections:
    - Major milestones
    - Uptake charts/graphs
    - Browser market share
    - Development framework popularity
    - Etc.
  - Use charts / images you find on-line
  - Include details of a few connections to Santa Cruz and/or UCSC
  - Must include references in IEEE style

Next Lectures

- Monday: Hypertext Markup Language (HTML)
- Tuesday: Cascading Style Sheets (CSS)

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 1** due 23:59 **Thursday October 8**

Web Applications

CSE183

Fall 2020

Hypertext Markup Language

Baskin Engineering
UC SANTA CRUZ

Notices

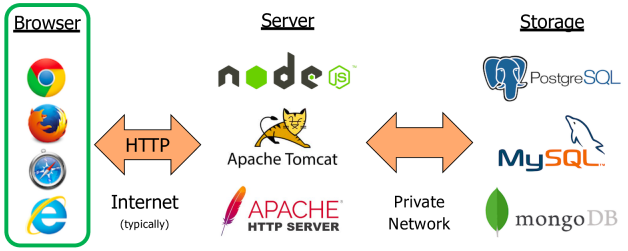
- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 1** due 23:59 **Thursday October 8**

Today's Lecture

- Browser Delivery
- Hypertext Markup Language (HTML)
- XML-Based HTML (XHTML)
- HTML5

- Questions

Full Stack Web Applications

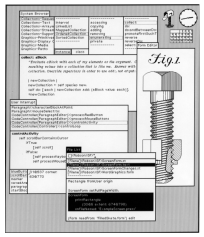


Early User Interfaces

- Early applications were “green screen” / character-based displayed on “dumb” terminals with limited capabilities
- App ran on a remote mainframe or mini-computer

Graphical User Interfaces (GUIs)

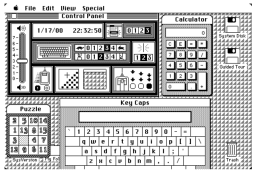
- Since the 1970’s applications have been able to access bitmapped displays via RGB framebuffers and make use of high-level GUI toolkits of buttons, tables, menus, etc.
- App ran on personal computer



Xerox Alto 1973

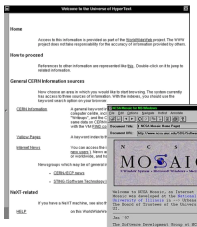
Microsoft Windows 1985

Apple MacOS 1984

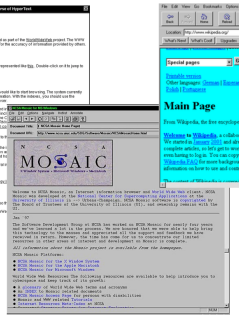


Browsers Changed the Game

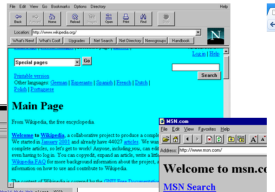
- Since 1990 browsers display documents described in HTML



CERN 1990



Mosaic 1993



Netscape 1994

Internet Explorer 1995

Chrome 2008

Browsers Changed the Game

- Since 1990 browsers display documents described in HTML
  - Until HTML5’s canvas region, couldn’t write pixels direct
- Applications constructed from documents
  - Initially multiple documents per app
  - Now (typically) single document manipulated with JavaScript
  - App typically runs partially in the browser, partially on a remote server

Hypertext Markup Language (HTML)

- Markup Languages include directives with content
  - Directives describe content or make presentation demands
  - Originated in the 1960's typesetting (printing) industry
  - Regarded as declarative languages
- General HTML App Approach
  - Start with your content
  - Annotate with tags (directives)
  - Leave the browser to render the content as directed
- Examples:
  - `<i>Something in italics</>`
  - `<title>Some Title</title>`

HTML Tags

- Layout:
  - `<h1>` "display text inside tag as a top-level heading"
  - `<p>` "start a new paragraph"
  - `<ul><li>` "start an unordered, bulleted list"
- Formatting:
  - `<i>` "display text inside tag in *italics*"
  - `<b>` "display text inside tag in **bold**"
- External Information:
  - `<img>` "display an image"
- Should be terminated:
  - `<i>Italic</i>` is correct, `<i>Italic` is not
- Can be nested:
  - `<i><b>Bold Italic</b></i>` displays ***Bold Italic***

HTML Example - Raw Text

```
The Hard Sell
Reasons for taking CSE183 include:
You need it to graduate.
You'll learn some cool stuff.
It'll give you interesting things to talk about in
job interviews.
```


HTML Example - With Tags

```
<h2>The Hard Sell</h2>
<p>
Reasons for taking
<b>CSE183</b>
include:
<ul>
<li>
You need it to graduate.
</li>
<li>
You'll learn some cool stuff.
</li>
<li>
It'll give you interesting things to talk about in job
interviews.
</li>
</ul>
</p>
```

HTML Example - Equivalent

```
<h2>The Hard Sell</h2><p>Reasons for taking<b>CSE183</b>
include:<ul><li>You need it to graduate.</li><li>You'll
learn some cool stuff.</li><li>It'll give you
interesting things to talk about in job interviews.</li>
</ul></p>
```


HTML Example - Formatted

```
<h2>The Hard Sell</h2>
<p>
  Reasons for taking
  <b>CSE183</b>
  include:
  <ul>
    <li>
      You need it to graduate.
    </li>
    <li>
      You'll learn some cool stuff.
    </li>
    <li>
      It'll give you interesting things to talk about
      in job interviews.
    </li>
  </ul>
</p>
```

- Browsers do not care
- Programmers do
- IDEs will format for you

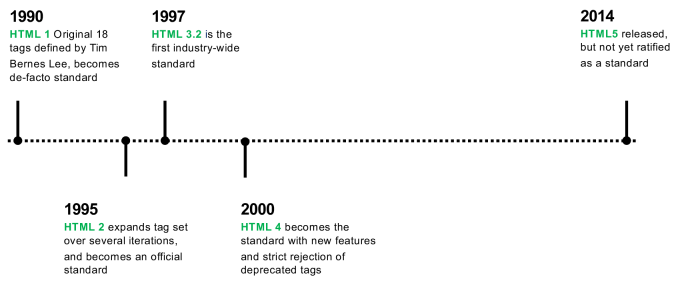
HTML Example - Rendered by Browser

The Hard Sell

Reasons for taking **CSE183** include:

- You need it to graduate.
- You'll learn some cool stuff.
- It'll give you interesting things to talk about in job interviews.

HTML Evolution

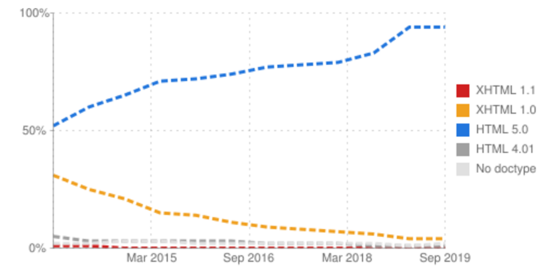


http://www.mediaprobe.com/frontend-engineering/engineer/html/html-top-history.html

HTML Evolution

- Influenced by browser peculiarities
  - What should the browser do if HTML has no paragraph closing tags?
    `<p>Whatever` instead of `<p>Whatever</p>`
  - Refuse to render the HTML?
  - Work out what tags are missing and act is if they were there?
  - Each browser had its own way of handling poorly formed HTML
  - Programmers started to rely on these browser quirks ☹
- XHTML (Extensible Hypertext Markup Language) grew out of HTML starting as an extension to HTML 4 in 1998
- Using XHTML can be thought of as requesting the browser honor a strict definition of HTML
- We will use XHTML in CSE183 assignments to ensure you write well-formed HTML

HTML - Version Adoption



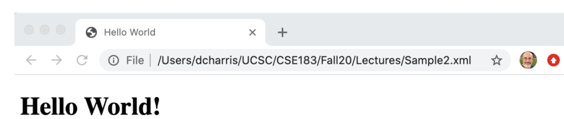
https://try.powermapper.com/Stats/HtmlVersions

XHTML - Basic Syntax

- Each file contains a single Document, a hierarchy (tree) of elements with the `<html>` at the top tree
- Elements may be nested
- Elements should have matching start and end tags
  - `<i>Italic</i>` is correct, `<i>Italic` is not
  - `<p/>` is valid shorthand for `<p></p>`
- Start tags can have attributes
  - `<input type="text">`
  - `<img src="mugshot.png">`

XHTML - Example

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```



XHTML - Example

Declares this is an XHTML document conforming to version 1.0 of the standard

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Outermost element of the document

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

Data about the document

```
  <head>
    <title>Hello World</title>
  </head>
```

Document content

```
  <body>
    <h1>Hello World!</h1>
  </body>

</html>
```


HTML - Common <body> Tags

`<!- Comment -->`	Comments
`<p>`	New paragraph
` `	Force a line break within the same paragraph
`<h1> <h2> ...`	Headings
`<b> <i>`	Boldface and italic
`<pre>`	Preformatted, whitespace is significant
`<img>`	Images
`<a href="…">`	Links to another Web page
`<table> <tr> <td>`	Tables
`<ul> <li>`	Unordered list (bulleted)
`<ol> <li>`	Ordered list (numbered)
`<div> <span`	Group related elements on a single line with (`<div>`) or without (`<span>`) pre and post line breaks
`<form> <input> <select> …`	For constructing data input forms

https://www.tutorialspoint.com/html/html_tags_reference.htm

HTML - Common <head> Tags

`<title>`	Page title, appears in the browser window title bar
`<meta>`	Document keywords to help search engines
`<link>`	Include Cascading Stylesheet
`<script>`	Add JavaScript to a page - can also be used in `<body>`

https://www.tutorialspoint.com/html/html_header.htm

HTML - Markup Characters

Can't use markup characters like < and > in document text so use entities by name or number, basic ones are:

Symbol	Description	Entity Name	Number Code
"	quotation mark	`"`	`"`
'	apostrophe	`'`	`'`
&	ampersand	`&`	`&`
<	less-than	`<`	`<`
>	greater-than	`>`	`>`

https://www.tutorialspoint.com/html/html_entities.htm

HTML - ISO 8859-1 Symbol Entities

Long list of special characters, including:

Result	Description	Entity Name	Number Code
	non-breaking space		
®	registered trademark	®	®
™	trademark	™	™

https://www.tutorialspoint.com/html/html_entities.htm

HTML 5

- Newest version, but not yet a true standard
- New tags:
  - Content definition:
    - `<article> <section> <header> <footer> <summary>`
    - `<mark> <figcaption> <figure>`
    - `<nav> <menuitem>`
  - Drawing:
    - `<svg>` Scalable Vector Graphics - Draw shapes
    - `<canvas>` Draw from JavaScript - 3D with WebGL
  - Timed media playback:
    - `<video>` and `<audio>`

https://www.htmldoodles.com/tutorials/html5/new-tags-in-html5.html

Next Lecture

- Wednesday: Cascading Style Sheets (CSS)
- Friday: Resource Locators and Links

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 1** due 23:59 **Thursday October 8**

Web Applications

CSE183

Fall 2020

Cascading Style Sheets

Baskin Engineering
UC SANTA CRUZ

Notices

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 1** due 23:59 **Thursday October 8**

- **Assignment 2** available 08:00 **Friday October 9**
- **Quiz 1** during lecture **Friday October 16** (Wrong on Zoom Recording)

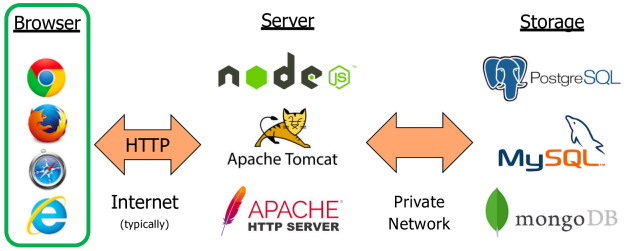
- All assignments now listed on Canvas
  - Locked until availability date

Today's Lecture

- HTML Weaknesses
- Separation of style from content
- Cascading Style Sheets

- Assignment 1 Secret Sauce 
- Questions

Full Stack Web Applications



An HTML Weakness

Say we have an HTML table:

```
<font face = "arial,sanserif" size = "6">
  <table>
    <tr>
      <th>Reasons for taking CSE183</th>
    </tr>
    <tr>
      <td>You need it to graduate.</td>
    </tr>
    <tr>
      <td>You'll learn some cool stuff.</td>
    </tr>
    <tr>
      <td>It'll give you interesting things to talk about in job interviews.</td>
    </tr>
</table>
</font>
```

```
<th align = "left">Reasons for taking CSE183</th>
```

Override defaults with an attribute

Center alignment ☹

Reasons for taking CSE183
You need it to graduate.
You'll learn some cool stuff.
It'll give you interesting things to talk about in job interviews.

Left alignment ☺

Reasons for taking CSE183
You need it to graduate.
You'll learn some cool stuff.
It'll give you interesting things to talk about in job interviews.

An HTML Weakness

- In our HTML example we had: `<h2>The Hard Sell</h2>`
  - Q: What font should the browser use for the level 2 heading?
  - A: Some default from the browser
- Basic Principle:
  - HTML says "what" browser says "how"
- But what if we'd like to say what font to use?
- Early HTML approach: The `<font>` tag

```
<font face = "arial,sanserif" size = "10">
    The Hard Sell
</font>
```

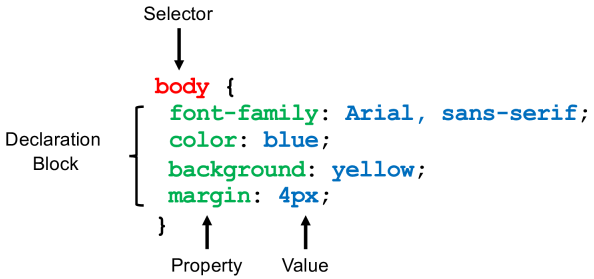

Pros and Cons of HTML Approach

- Can override the browser defaults ☺
- Becomes problematic if you have many elements you want to change the same attribute on ☹

Solution: Style Sheets

- Principle 1 : Separate Style from Content
  - Content (the text to display) is in HTML files
  - Style (how to format that text) is in a separate file
- Principle 2 : Don't Repeat Yourself
  - Define style once, us it in many places
- Implementation:
  - Add the `class` attribute to all tags; links style and content
  - `<th class = "table-header">`

Style Sheet Rules



Selectors

Selector	Style Sheet	HTML
Tag Name	`h2 { color: blue; }`	`<h2>The Hard Sell</h2>`
Class Attribute	`.large { font-size: 16pt; }`	`<div class = "large"/>`
Tag & Class	`p.large { font-size: 20pt; }`	`<p class = "large"/>`
Element ID	`#tr24 { font-weight: bold; }`	`<tr id = "tr24"/>`

Pseudo Selectors

- Add special effects to some selectors without needing JavaScript to generate the effect
- E.g. apply rule when:
  - `hover` : mouse pointer over element
  - `link` & `visited` : link has not been visited or has

```
p:hover {
  background-color: yellow;
}
a:visited {
  color: green;
}
a:link {
  color: blue;
}
```


https://www.tutorialspoint.com/css/css_pseudo_classes.htm

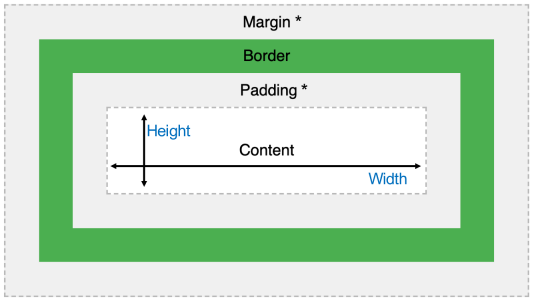
CSS Properties

- Text
  e.g. `p { text-decoration: line-through; }`
- Colour
  - Predefined literals: `red`, `white`, etc.
  - 3 x 8-bit hexadecimal intensities for red, green, blue: `#ff0000`
  - Decimal intensities via function: `rgb(255,255,0)`
  - % intensities via function: `rgb(0%,100%,100%)`

  e.g. `p:hover { background-color: rgb(255,128,255); }`

- Visibility, Size & Position
  - See later slides
- Plus many more
  - Self-study

The CSS Box Model



* Transparent

https://www.w3schools.com/css/css_boxmodel.asp

CSS Distance Units

- Absolute
  - `px` pixel ($^1/_{96}$ of an inch)
  - `mm` millimeter
  - `cm` centimeter
  - `in` inch
  - `pt` printer point ($^1/_{72}$ of an inch)
- Relative
  - `em` 2 x element's current font size
  - `rem` 3 x root element's current font size

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```


Size Properties

- Element

  `width, height`

- Box

  `padding, padding-top, padding-right, padding-bottom, padding-left`

  `margin, margin-top, margin-right, margin-bottom, margin-left,`

  `border, border-top border-right border-bottom border-left`

- Border

  `border-color, border-top-color,`

  `border-style border-top-style`

  etc. etc.

- Border shorthand

```
div { border: 5px solid red; }
```

size style color

https://www.w3schools.com/css/css_border_shorthand.asp

Position Property

- Default position in document

  `position: static`

- Via `top`, `right`, `bottom`, and left properties:
  - Relative to default position

    `position: relative`
  - Relative to fixed position in the document

    `position: fixed`
  - Relative to nearest positioned ancestor element

    `position: absolute`
  - Relative to position of cursor

    `position: sticky`

- Fixed position (0,0) is top left of document

https://www.w3schools.com/css/css_positioning.asp

Visibility Properties

`display: none;`	Element not displayed and takes no space
`visibility: hidden;`	Element hidden but space allocated
`visibility: visible;`	Element is normally displayed

Plus others (self study)

Other Common Properties

- Set image for element's background
  `background-image`
- Font settings
  `font, font-family, font-size, font-weight, font-style`
- Text and sub element alignment
  `text-align, vertical-align`
  Values `center, left, right`
- Set the cursor when over element
  `cursor`
- And many, many more
  - Self study

Flexbox and Grid Layout

- Items "flex" (stretch) to fill additional space and shrink if space is limited
  - Useful for page layout ☺ but can be a little tricky to get the hang of ☹
- Flexbox - Layout in one dimension (row or column)
- Grid - Layout in two dimensions (rows and columns)

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  background-color: #2196F3;
  padding: 10px;
}
.grid-item {
  background-color: rgb(255, 255, 0);
  border: 1px solid rgb(0, 0, 0);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
```

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
</div>
```

1	2	3
4	5	6

https://www.w3schools.com/css/css_grid.asp

CSS Peculiarities

- Inheritance
  - Some properties (e.g. font-size) are inherited from parent elements
  - Others (border, background) are not inherited
- Multiple rule matches
  - Most specific rule wins
  - If we have this CSS:
    ```
    .mult { color: red }
    div.mult { color: green }
    div { color: blue }
    ```
  - And this HTML:
    ```
    <div class="mult">Hello World</div>
    ```
  - What gets rendered in the browser?
    Hello World

Adding Styles

- Link to separate file (recommended approach)

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

- Page specific styles

```
<head>
<style type="text/css">
  body {
    font-family: Arial, sans-serif;
  }
 </style>
</head>
```

- Element specific styles

```
<div style="padding:2px; ... ">
```


CSS Example

```
<h1>Heading One</h1>
<p>
  Paragraph one.
</p>
<div class="shaded">
  <h2>Heading Two</h2>
  <p>
    Paragraph two.
  </p>
</div>
```

```
body {
  font-family: Arial, sans-serif;
  font-size: 20px;
}
h1 {
  font-size: 42px;
  border-bottom: 2px solid black
}
.shaded {
  background: rgb(0%,50%,100%);
}
```

Heading One

Paragraph one.

Heading Two

Paragraph two.

Heading One

Paragraph one.

Heading Two

Paragraph two.

CSS In Practice

- In realistic projects Style Sheets proliferate and start to exhibit the similar problems to the ones HTML had with style
  - Same style written in many places, even within a single Style Sheet
- One solution is to use a CSS pre-processor
  - Adds variables, macros and functions
  - Allows for scoping with naming conventions
  - Examples: Sass, less, Stylus
    - Self Study: https://ravgun.com/blog/css-preprocessors-examples/
- Problems when multiple Style Sheets are included
  - It's often hard to work out where a style is coming from ☹

Assignment 1 - Secret Sauce

- 1993 The Internet Underground Music Archive (IUMA)
  - "The birthplace of on-line music" founded by three guys from UCSC

IUMA Home Page

Assignment 1 - Secret Sauce

- 1994 Pizza Hut takes first on-line pizza order in Santa Cruz
  - The Santa Cruz Operation provided the technology

Next Lecture

- Friday: Resource Locators, Links, Assignment 2
- Monday: Introduction to JavaScript

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 1** due 23:59 **Thursday October 8**

Web Applications

CSE183

Fall 2020

Universal Resource Locators

Baskin Engineering
UC SANTA CRUZ

Notices

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**

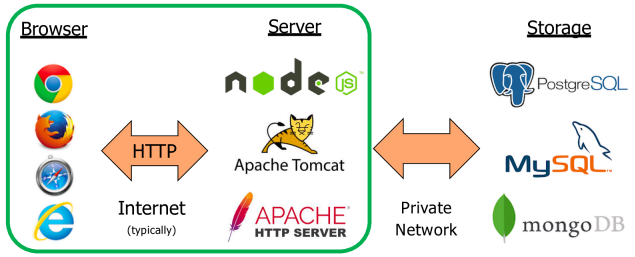
- **Quiz 1** during lecture **Friday October 16**

Today's Lecture

- Hypertext
- URLs
- Query Parameters
- HTML Links
- URL Encoding

- Assignment 2
- Questions

Full Stack Web Applications



Hypertext

- Text with links to other text
  - Original Idea:
    - Click on a link to go to another location in a document
  - Expanded to include:
    - Click on a link to go to another document
  - Originated in the early 60's
    - Find the 1968 "Mother of All Demos" by Doug Englebart and others from SRI
- HTML adopted the idea
  - Links in the same document:

```
<a href="#ref1">Click me!</a>
<p id="ref1">Some paragraph...</p>
```

  - Links to other documents:

```
<a href="https://en.wikipedia.org/wiki/URL">URL</a>
```

Uniform Resource Locator

URL Component parts

`scheme://host:port/path?query#fragment`

Scheme:	Web protocol, usually HTTPS
Host:	The internet Protocol (IP) host that holds the resource E.g. `www.ucsc.edu` or `localhost`
Port:	IP port number for hosts running multiple services "Well known" (traditionally used) ports exist for common schemes HTTPS = 443, HTTP = 80
Path:	"Location" of resource on the host No longer always a direct mapping to an HTML file on disk
Query:	If present, passed on to resource when retrieved
Fragment:	Not sent to server, browser will scroll to anchor tag

URL Example

`https://www.exp.com/staff/detail.html?name=Alice&dept=Sales#tr2`

Scheme:	HTTPS
Host:	www.exp.com
Port:	443 (the "well known" port for HTTPS)
Path:	/staff/detail.html
Query:	Two parameters: name=Alice and dept=Sales
Fragment:	Browser will scroll to tag with id tr2 after retrieval

URL Path

- Passed to server for interpretation
- Early Web Servers:
  - Path to static HTML file (e.g. `/staff/index.html`)
  - Path to HTML generation script (e.g. `/customer/orders.php`)
- Web Apps:
  - Contains routing information
    - Map path to HTML generation function and (possibly) parameters
- Application Programming Interfaces (APIs):
  - Map to Create/Retrieve/Update/Delete (CRUD) operations:

```
/customer/create
/customer/list
/customer/ec4b51c8e91b469891e62dba8577b86f
/customer/delete/f85eba2ac36b43478bc8496b69a6d78c
```

Query Parameters

- Originally:
  - Pass arguments to an operation
    `https://www.exp.com/order.php?id=1234`
  - Send form data to server
    `?field1=value1&field2=value2&…`
- Recently:
  - User tracking instead of cookies
    - Add unique id as param to every generated link
      `<a href="foo.html?e0a72cb2a2c7">Do foo</a>`

HTML Links

- Browser maintains a notion of current location
  - The URL the current HTML document was loaded from
- Links:
  - Content in a page which, when clicked, requests the browser fetch a new HTML document from a URL
  - Implemented with the `<a>` tag:
    `<a href="http://www.demo.com/sales/2020.html">2020 Sales</a>`

HTML Links

- URLs:
  - Full:
    `<a href="https://www.demo.com/sales/2020.html">2020 Sales</a>`
  - Absolute:
    `<a href="/stock/current.html"/>`
    Same as: `<a href="https://www.demo.com/stock/current.html">`
  - Relative:
    `<a href="2020/Oct.html"/>`
    Same as: `<a href="https://www.demo.com/stock/2020/Oct.html">`
- Other:
  - Anchors:
    - Jump to a specific location in the current document
      `<a href="#ref1">Click me!</a>`
      `<p id="ref1">Some paragraph...</p`

URLs in HTML

- Loading a new page:
  `<a href="..."/>` Or type the URL into your browser
- Load an image:
  `<img src="..."/>`
- Load a stylesheet:
  `<link rel="stylesheet" type="text/css" href="..."/>`
- Embed a page:
  `<iframe src="http://www.ucsc.edu">`

URL Encoding

- Punctuation and white space in URLs is problematic

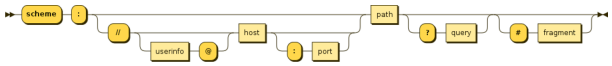
  `https://www.demo.com/customer?name=M&S Stores`

- Characters other than upper- and lower-case letters, digits, and any of `-_.` or `~` in a URL must be represented as `%xx` where `xx` is the hex value of the character

  `https://www.demo.com/customer?name=M%26S%20Stores`

URL vs. URI

- Universal Resource Identifier




```
        userinfo       host      port
        ┌──┴───┐ ┌──────┴──────┐ ┌─┴─┐
https://john.doe@www.example.com:123/forum/questions/?tag=networking&order=newest#top
└─┬─┘   └─────────────┬─────────────┘└───────┬──────┘└────────────┬────────────┘ └┬┘
scheme          authority                path                 query           fragment

ldap://[2001:db8::7]/c=GB?objectClass?one
└─┬─┘ └──────┬──────┘└─┬┘ └──────┬──────┘
scheme   authority   path      query

mailto:John.Doe@example.com
└─┬──┘ └─────┬─────┘
scheme     path
```

- URLs can be thought of as URIs with an HTTP(S) scheme

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

Assignment 2

- Simple HTML and CSS exercise
  - ***Single*** HTML file
  - Three different style sheets
  - Browser renders quite differently styled versions of ***the same content***
- Three requirements
  - Basic - 2 points
    - Display some vertical text at the top center of the page
  - Advanced - 1 points
    - Split the text - letters top left, numbers bottom right
    - Implement hover effects
  - Stretch - 1 point
    - Use some fancy effects we did not cover in class
      - Do you own research on how they work
- 1-point deduction for poorly formatted code

Upcoming Lectures

- Monday: JavaScript I
- Tuesday: JavaScript II
- Friday: Quiz 1 & Assignment 3
- Monday: JavaScript III

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**

Web Applications

CSE183

Fall 2020

JavaScript I

Baskin Engineering UC SANTA CRUZ

Notices

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**

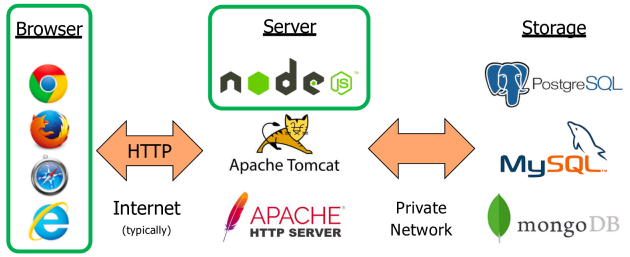
- **Quiz 1** during lecture **Friday October 16**

Today's Lecture

- JavaScript Overview
- Dynamic Types
- Scoping & Hoisting
- Basic Types
- Classes, Objects, Properties
- Arrays, Dates, Regular Expressions
- Exceptions
- Including JavaScript in HTML

- Questions

Full Stack Web Applications



JavaScript - Overview

- High-level
  - Heavily abstracted from hardware details
- Interpreted
  - Not compiled, executed by a platform-dependent run-time environment
- Dynamic
  - Undertakes compiler-like operations at runtime
- Untyped / Dynamically Typed
  - Any variable can hold any type of data
- Prototype-based
  - Object-oriented behaviors are re-used (inherited) from existing objects (prototypes)
- Has first-class functions
  - Functions are objects and can be manipulated as such

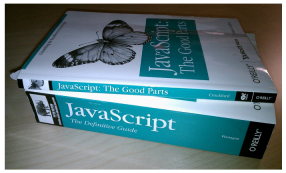
JavaScript - Overview Cont.

- Programming Models
  - Object-oriented
    - Encapsulation, abstraction, inheritance, polymorphism,
  - Imperative
    - Instructions executed sequentially; code is easy to understand
  - Functional
    - Declarative composition of value returning functions creating a call tree rather than manipulating a global state
- Not particularly like Java
  - Both heavily influenced by C, but took different paths
- ECMAScript
  - JavaScript Standard supported by most browsers
  - New version every year
    - Current is the 11th edition, "EMCAScript 2020"

https://www.ecma-international.org/ecma-262/11.0/index.html

JavaScript - Overview Cont.

- Early versions earned a bad reputation
  - Invented at Netscape in 1995
  - Appeared to have been designed in a rush
  - Grew rapidly with little consideration for cross-browser compatibility
  - ECMAScript initiative calmed things down
- The upside
  - It's feature rich ☺
- The downside
  - It's huge and complicated ☹
  - Code quality checkers are essential
    - Eg. `jshint`, `jslint`
- The real upside
  - We can use the same language in the browser and the server ☺ ☺ ☺



Heavily Based on C

```
int dat = 3;
int sum = 0;
int i = 0;

int bar(int p) {
  return p * 0.4;
}

void foo() {
  dat = dat * 6 - 2 + (dat / 10);
  while (dat >= 0) {
    sum += dat * dat * 1.2; // line comment
    dat--;
  }
  for (i = 0; i < 4; i++) {
    /* block comment */
  }
  if (dat < 2) {
    dat = bar(dat);
  } else {
    dat = dat * 0.04;
  }
}

int main(int argc,  char *argv[]) {
  foo();
  printf("dat: %d\nsum: %d\n", dat, sum);
}
```

```
var dat = 3;
var sum = 0;
var i = 0;

function bar(p) {
  return p * 0.4;
}

function foo() {
  dat = dat * 6 - 2 + (dat / 10);
  while (dat >= 0) {
    sum += dat * dat * 1.2; // line comment
    dat--;
  }
  for (i = 0; i < 4; i++) {
    /* block comment */
  }
  if (dat < 2) {
    dat = bar(dat);
  } else {
    dat = dat * 0.04;
  }
}

foo();
console.log(`dat: ${dat}\nsum: ${sum}`);
```

```
$ gcc -o SameAsC SameAsC.c
$./SameAsC
dat: 0
sum: 1789
```

```
$ node SameAsC.js
dat: -0.27999999999999975
sum: 1894.9560000000004
```

Why the difference?

Dynamic Typing

```
var i;          // typeof i == "undefined"
i = 128;        // typeof i == "number"
i = "hello";    // typeof i == "string"
i = true;       // typeof i == "boolean"
```

- Variables have the type of their most recent assignment
- Primitive types:
  `undefined`, `number`, `string`, `boolean` , `bigint`
- Structural types:
  `function`, `object`

Scoping & Hoisting

- Global and function scopes
- All `var` statements hoisted to function start

```
var gVar;

function() {
  // lVar2 is hoisted here
  var lVar;
  if (gVar > 0) {
    var lVar2 = 2;
  }
  // lVar2 is valid here - it has function scope
}

function foo() {
  // x is hoisted here
  x = 2
  var x;
  ...
}
```


Scoping & Non-Hoisting

- `var` has non-explicit scopes

```
function() {
  // str is hoisted here
  console.log('Str is:', str);
  ...
  for(let i = 0; i < 2; i++) {
    var str = "whatever";
    ...
  }
```

- `let` and `const` have explicit scopes

```
function() {
  console.log('Str is:', str);    // syntax error!
  ...
  for(let i = 0; i < 2; i++) {
    let str = "whatever";
    ...
  }
```


`Number` type

- Stored as 64-bit floating point number - like `double` in C
  `Number.MAX_SAFE_VALUE = 2^53 -1`
- Do NOT check for equality
  `0.1 + 0.2 != 0.3` (it's 0.30000000000000004)
- Peculiarities:
  - "not a number" and "infinity" are numbers
    `1/0 == Number.POISITIVE_INFINITY;`
    `Math.sqrt(-1) == Number.NaN;`
  - Bitwise operators (`~`, `&`, `|`, `^`, `>>`, `<<`, `>>>`) are 32-bit

`String` type

- Variable length (no equivalent of C `char` type)

```
var str = 'Hello World';
str.length == 11;
```

- Concatenation operator: `+`

```
str += '!';
str == 'Hello World!'
```

- Utility methods:

```
indexOf(), charAt(),
match(), search(), replace(), slice()
toUpperCAse(), toLowerCase()
```

etc. etc.

`Boolean` type

- `true` or `false`
- Plus `truthy` and `falsy` for conversions to Boolean
  - `falsy`

    `false`, `0`, `""`, `null`, `undefined`, `NaN`
  - `truthy`

    Implicitly `!falsey`

    All objects (including functions)

    non-empty strings

    non-zero numbers

`undefined` & `null`

- `undefined` : does not have a value assigned

```
var i;           // no initial value
i = undefined;   // explicit removal of value ( if any )
// i == undefined;
```

- `null` : A “special” value representing whatever you like
- Both are `falsy`, but *not* equal

```
null == undefined
null !== undefined
```

- Self Study:

  == & != abstract comparison operators

  === & !== strict comparison operators

`function` type

```
function foo(x) {
  return x > 1 ? 1 : x * foo(x-1);
}

// typeof foo == 'function';
// foo.name == 'foo';
```

- Same as:

```
var foo = function bar(x) {
  return x > 1 ? 1 : x * foo(x-1);
}
```

- Definitions are hoisted
- Implicitly support variable arguments
- All return a value

First class `function`

```
var vfunc = function (x) {
  console.log('Called with number', x);
  return x + 1;
};

function func(f) { // passed as a param
  console.log('Called with', f.toString());
  var ret = f(10);
  console.log('Return value', ret);
  return ret;
}

func(vfunc);
```

```
Called with function (x) {
  console.log('Called with number', x);
  return x+1;
}
Called with number 10
Return value 11
```


`Object` Type

- Unordered collection of name-value pairs

```
var foo = {};
var bar = {name: "Alice", age: 23, state: "California"};
```

- Name can be any string:

```
var x = { "": "empty", "---": "dashes"}
```

- Referenced like a structure

```
bar.name
// foo.nonExistent == undefined
```

- Or a hash table with string keys:

```
bar["name"]
```

- Global scope is an object in browser (i.e. `window[prop]`)

Dynamic Types - Properties

- Add:
  - Assign the property

```
var foo = { };
foo.name = "Foo's Name";
```

- Remove:

```
var foo = { name: "Foo's Name" };
delete foo.name;
```

- Enumerate:

```
var user = { name: "Alice", age: 23 };
console.log(Object.keys(user));
```

```
[ 'name', 'age' ]
```


Arrays

```
var arr = [1,2,3];
arr.length == 3
```

- Special objects: `typeof arr == 'object'`
- Indexed by non-negative integers: `arr[0] == 1`
- Can be sparse and polymorphic:
  `arr[5]='FooBar';` (`arr` is now `[1,2,3,,,'FooBar']`)
- Have many methods:
  - `push`, `pop`, `shift`, `unshift`, `sort`, `reverse`, `splice`, ...
- Can store object properties (e.g. `arr.name = 'Foo'`)
  - But some properties have special uses: (e.g. `arr.length = 0;`)
    - Self study:
      - Try to delete the `length` property from an array

`Date` Objects

```
var date = new Date();
```

- Special objects: `typeof date == 'object'`
  - The number of milliseconds since midnight January 1, 1970 UTC
  - Timezone needed to convert
  - Not good for fixed dates (e.g. birthdays)
- Many methods for getting and setting:

```
date.valueOf() == 1602463885467
date.toISOString() == '2020-10-12T00:51:25.467Z'
date.toLocaleString() == '10/11/2020, 5:51:25 PM'
```


Regular Expressions

- `var re = /ab+c/;` or `var re2 = new RegExp("ab+c");`
- Defines a pattern that can be searched for in a string
  - String: `search()`, `match()`, `replace()`, and `split()`
  - RegExp: `exec()` and `test()`
- `test`

	Returns
`/SS/.test(str);`	True if `str` has the substring `SS`
`/ss/i.test(str);`	Same but case insensitive
`/[Cc]se [0-9]/.test(str);`	True if str either “Cse N” or “cse N”

- `search`

`'XXX abbbbbbc'.search(/ab+c/);`	4 (position of ‘a’)
`'XXX ac'.search(/ab+c/);`	-1, no match
`'12e34'.search(/[^\d]/);`	2

Regular Expressions

```
var str = "This has 'quoted' words like 'this'";
var re = /'[^']*'/g;
var match = null;

while ((match = re.exec(str)) !== null)
  console.log(match);
```

```
[
  "'quoted'",
  index: 9,
  input: "This has 'quoted' words like 'this'",
  groups: undefined
]
[
  "'this'",
  index: 29,
  input: "This has 'quoted' words like 'this'",
  groups: undefined
]
```


Exceptions

```
try {
  undefinedFunction();
} catch (err) {
  console.error("Error:", err.name, ":", err.message);
} finally {
  console.log("Finally...");
}

function throwsString() {
  throw "Whoah Nelly!";
}

try {
  throwsString();
} catch (err) {
  console.error("Error", err.toString());
}

function throwsError() {
  throw new Error("Whoah Nelly!");   // better than throwing a string
}
```

JavaScript & HTML

- Include a separate file: (recommended approach)

```
<script type="text/javascript" src="foo.js"></script>
```

- Embed in the HTML:

```
<script type="text/javascript">
  console.log("Hello World!");
</script>
```


Upcoming Lectures

- Wednesday: JavaScript II
- Friday: Quiz 1 & Assignment 3
- Monday: JavaScript III

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**

Web Applications

CSE183

Fall 2020

JavaScript II

Baskin Engineering
UC SANTA CRUZ

Notices

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**

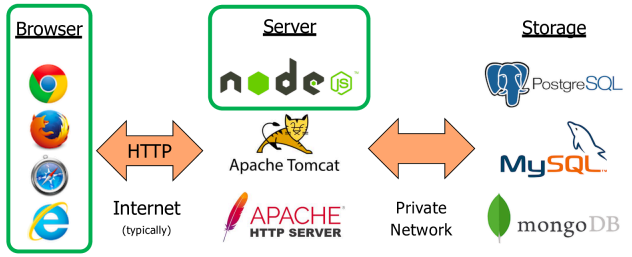
- **Quiz 1** during lecture **Friday October 16**

Today's Lecture

- JavaScript Recap
- Object Oriented Programming
- Functional Programming
- Closures
- Idioms and Quirks

- Quiz Instructions
- Questions

Full Stack Web Applications



JavaScript - Overview

- High-level
  - Heavily abstracted from hardware details
- Interpreted
  - Not compiled, executed by a platform-dependent run-time environment
- Dynamic
  - Undertakes compiler-like operations at runtime
- Untyped / Dynamically Typed
  - Any variable can hold any type of data
- Prototype-based
  - Object-oriented behaviors are re-used (inherited) from existing objects (prototypes)
- Has first-class functions
  - Functions are objects and can be manipulated as such

JavaScript - Programming

- Models
  - Object-oriented
    - Encapsulation, abstraction, inheritance, polymorphism
  - Imperative
    - Instructions executed sequentially; code is easy to understand
  - Functional
    - Declarative composition of value returning functions creating a call tree rather than manipulating a global state
- Evolution
  - Originally convention (pattern) based
  - ECMAScript keeps adding language features
    - E.g. the `class` concept

https://www.ecma-international.org/ecma-262/11.0/index.html

Object Orientation - Methods

- First class functions ⇒ an object property can be a function
  - A “method” in object oriented speak

```
var obj = {count: 0};
obj.increment = function (amount) {
 this.count += amount;
 return this.count;
}
```

- Method invocation:
  - Function is called and literal `this` is bound to the object

```
obj.increment(2); // returns 2
obj.increment(-1); // returns 1
```


Object Orientation - `this`

- In methods `this` is bound to the object

```
var obj = { prop1: 'property 1'};
obj.addProp2 = function() {
 this.prop2 = "property 2";
}
console.log(Object.keys(obj));
obj.addProp2();
console.log(Object.keys(obj));
console.log(obj);
```

```
[ 'prop1', 'addProp2' ]
[ 'prop1', 'addProp2', 'prop2' ]
{ prop1: 'property 1', addProp2: [Function], prop2: 'property 2' }
```

- In non-method functions:
  - `this` will be the global object
  - Or if `"use strict";` `this` will be undefined
    - However, `"use strict";` should *not* be used globally ¯_(ツ)_/¯

Object Orientation - Functions are Objects

- Functions can have properties

```
function increment(value) {
  if (increment.invocations == undefined) {
    increment.invocations = 0;
  }
  increment.invocations++;
  return value + 1;
}
increment(4);
increment(-1028);
// increment.invocations == 2
```

- Analogous to static/class properties in “purer” object-oriented languages like Java

Object Orientation - Functions are Objects

- Functions can have methods

```
function func(arg) { console.log(this,arg); }
```

- `toString()` returns function as source string

  `func.toString()` returns `'function func(arg) { console.log(this,arg); }'`

- `call()` calls function specifying `this` and arguments

  `func.call({t: 1}, 2)` prints `'{ t: 1 } 2'`

- `bind()` creates new function with `this` and arguments bound

```
let newFunc = func.bind({z: 2}, 3);
newFunc(); prints '{ z: 2 } 3'
```


Object Orientation - Classes

- Functions are classes

```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
  this.area = function() { return this.width * this.height; }
}
var r = new Rectangle(6, 7);
// r.area(); returns Jackie Robinson's 42
```

- Not a good way to add methods - Why?

Object Orientation - Inheritance

- JavaScript has a prototype object for each object instance
  - Prototype objects can have their own prototype objects forming a prototype chain
- When *reading* the value on an object property, JavaScript will search up the prototype chain until the property is found
  - The full set of properties of an object are its own properties plus all the properties found up the prototype chain
    - This is known as prototype-based inheritance
    - Single inheritance only, cannot inherit properties from two ‘parents’
- Property *updates* work differently:
  - JavaScript creates the property in the object instance if not found up the prototype chain

Using Prototypes

```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
}
Rectangle.prototype.area = function() {
  return this.width * this.height;
}
var r = new Rectangle(6, 7);
// r.area(); still returns Jackie Robinson's 42
```

- As JavaScript has a dynamic type system, changing the prototype causes all instances to change
  - Which is typically what we want
  - Much better way of adding a method ☺

Inheritance

```
function Shape(name) {
  this.name = name;
}
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
}
Rectangle.prototype = new Shape('Rectangle');
Rectangle.prototype.area = function() {
  return this.width * this.height;
}
var r = new Rectangle(6, 7);
let s = new Shape('Generic Shape');

// r.Name == 'Rectangle'
s.area(); // syntax error
```


ECMAScript Syntax

```
class Shape {                           // Class definition
  constructor(name) {
    this.name = name;
  }
}
class Rectangle extends Shape {         // Class definition and Inheritance
  constructor(height, width) {
    super('Rectangle');
    this.height = height;
    this.width = width;
  }
  area() {                              // Instance method definition
    return this.width * this.height;
  }
  static foo() {                        // Static method definition
  }
}
var r = new Rectangle(10,20);
console.log(r);
```

```
Rectangle { name: 'Rectangle', height: 10, width: 20 }
```


Functional Programming

```
let before = [1,2,3,4,5,6];
let after = [];
```

- Imperative

```
for (var i = 0; i < before.length; i++) {
  after[i] = before[i]*i;
}
```

- Functional

```
after = before.map(function (value, index) {
  return value*index;
});
```

- ECMAScript "Arrow" Functional

```
after = before.map((value, index) => value*index);
```

- In all cases

```
console.log(after);
```

```
[ 0, 2, 6, 12, 20, 30 ]
```

Functional Programming

- You can program almost entirely in an imperative style, but..
- Functional programming is required in asynchronous code
  - Browser:

```
function callback() {
  console.log("timeout");
}
setTimeout(callback, 3*1000);
```

  - Server:

```
function callback(err, data) {
  console.log(String(data));
}
fs.readFile('/etc/passwd', callback);
```


Closures

- Given:

```
var glob = 1;
function local(arg) {
  var loc = 0;
  function embedded() {return ++loc + arg + glob;}
  return embedded;
}
```

- What's the difference between:

```
let func = local(2);
console.log(func());
console.log(func());
```

```
4
5
```

and

```
console.log(local(2)());
console.log(local(2)());
```

```
4
4
```

- The closure of `func` includes `glob`, `loc`, and `arg`
- The other version has two distinct instances of `local`

Closures for Privacy

```
let obj = (function() {
 let count = 1;
 let text = "test";
 let setCount = function(value) { count = value; }
 let compute = function() { return count + ' ' + text; }
 return {compute: compute, setCount: setCount};
})();
// typeof obj ==  'object'
// Object.keys(obj) returns [ 'compute', 'setCount' ]
obj.setCount(128);
```

- What does `obj.compute()` return?
  `'128 test'`
- What is the value of `obj.count`?
  `undefined`

`count` and `text` are enclosed as private attributes of `obj`

Nested functions and `self`

```
function readFileFunc() {
  fs.readFile(this.fileName, function (err, data) {
    if (!err) {
      console.log(this.fileName, 'length', data.length);
    }
  });
}
({fileName: "/etc/passwd", readFile: readFileFunc}).readFile();
```

```
undefined length 6946
```

```
function readFileSelf() {
  let self = this;
  fs.readFile(this.fileName, function (err, data) {
    if (!err) {
      console.log(self.fileName, 'length', data.length);
    }
  });
}
({fileName: "/etc/passwd", readFile: readFileSelf}).readFile();
```

```
/etc/passwd length 6946
```

```
function readFileArrow() {
  fs.readFile(this.fileName, (err, data) => {
    if (!err) {
      console.log(this.fileName, 'length', data.length);
    }
  });
}
({fileName: "/etc/passwd", readFile: readFileArrow}).readFile();
```

```
/etc/passwd length 6946
```

JavaScript Object Notation (JSON)

```
let obj = { s: 'str', n: 1, a: [1,'two',,4], o: { n: 1}};
console.log(obj);
```

```
{ s: 'str', n: 1, a: [ 1, 'two', <1 empty item>, 4 ], o: { n: 1 } }
```

```
let json = JSON.stringify(obj);
console.log(json);
```

```
{"s":"str","n":1,"a":[1,"two",null,4],"o":{"n":1}}
```

```
let nobj = JSON.parse(json);
console.log(nobj);
```

```
{ s: 'str', n: 1, a: [ 1, 'two', null, 4 ], o: { n: 1 } }
```


A Useful JavaScript Idiom

- Assign default values

```
class Rectangle {
  constructor(height, width) {
    this.height = height || 0;
    this.width = width || 0;
  }
  area() {
    return this.width * this.height;
  }
}
```


CSE183 Quizzes

- Zoom Proctoring
  - Have your camera on pointing at you
  - Turn microphone off
- Canvas Quiz
  - Randomized question order
  - Must be answered in order presented
  - Some multiple choice
  - Some text-entry
- 25 Minutes
  - Starts at 09:25
  - DRC Accommodations have time multipliers
  - Submits automatically
- Practice quiz available later today

Upcoming Lectures

- Friday: Quiz 1 & Assignment 3
- Monday: JavaScript III & Document Object Model

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**