**Week 1 Notes**

Reading: JavaScript and the Browser

* Web technology is decentralized
  + Loose collaboration to improve, but minimal internal consistency

INTERNET

* A computer network that allows computers to shoot bits at each other
* Network protocol: style of communication over a network
* Hypertext Transfer Protocol (HTTP): protocol for retrieving named resources
  + Specifies that the side making the request should start with a line like this:

GET /index.html HTTP/1.1

* + Treats the network as a streamlike device that has the bits arrive at the correct destination in the correct order (very hard to do!)

PROTOCOLS

* Most protocols are built on top of other protocols
* Transmission Control Protocol (TCP) addresses the HTTP problem
  + One computer waits (i.e. *listens*) for other computers to talk to it
  + To listen for different kinds of comms at the same time on one machine – each listener has a number i.e. port associated, most protocols have a default port
  + Another computer can establish connection by connecting to target machine w/ port #
  + Acts as a two-way pipe for bits to flow. Machines on both ends can put data into it
* SERVER: Listening Computer CLIENT: Connecting Computer

THE WEB

* World Wide Web (WWW) is a set of protocols and formats that let us visit web pages in a browser
  + To become part of web, connect machine to Internet and have it listen on port 80 w/ HTTP protocol
* Each document on the Web is named via Uniform Resource Locator (URL)
* Machines connected to internet get IP address
* Can register domain name for specific address or set
* Typing URL into browser → browser tries to retrieve & display doc @ URL. Browser looks for what the URL refers to, then w/ HTTP protocol, it makes connection to server @ address ands asks for the resource. If it works, the server sends the doc back

HTML (Hypertext Markup Language) and JavaScript

* HTML: Doc format used for web pages
* JavaScript: Make things happen (alerts, buttons, background changes, etc.)
* <script> tag in HTML can be given an stc attribute to fetch a script file
* Browsers severely limit the things a JavaScript program can do
  + This is an isolation trick called *sandboxing*
    - Has to allow programs to be useful while still restricting them from doing anything dangerous

**Web Architecture Overview**

Two parts: client and server

Client-Server Relationship

*Client*:

For class purpose, client = web browser. Things that happen on client side are things that happen within the browser w/o contacting a server

*Server*:

Simplified – the server is what the client requests data from. In general it is some computer elsewhere, but can be a server app accessed locally. Can use own computer file system as server (can’t change content on page, but can serve static files)

Server side means it has access to all the data on the server (e.g. weather site has recent weather data)

**Web Languages**

There are a bunch used, but generally:

*Document structure*: HTML, is not responsible for what the page looks like

*Page Look & Feel*: CSS, modifies the layout and styling of a page

*Client Side Interactivity*: JavaScript, resembles C language, has all control structures expected in robust language

*Server Side Languages*: Many, many for this (PHP, C#, JavaScript, Ruby, Go, Python, etc.). In this class, JavaScript & Node.js to program on server. Others – Django w/ Python, or PHP (which does not require framework)

**HTTP and TCP/IP**

HTTP: HyperText Transfer Protocol

TCP/IP: Transmission Control Protocol/Internet Protocol

HTTP Request and Response

Request gets sent from browser to server, composed of two parts – the headers and the body. Body is often blank, but if a form, for example is submitted to the server, the info is included in that body

Important parts of the header:

First Line – specifies the request type (often GET or POST), resource requested, version of HTTP used

Host – who the client sends the request to

Accept – Type of file client is OK getting back from server

Cookie – info stored on client that is sent to server w/ every request, can be used to keep track which client is which when there are several clients

Referer – Page the client came from, can be used for analytics but can be fooled

Response

Important parts:

First Line – HTTP status code (generally <400 is OK, >400 is bad)

Location – where you end up after server does its work.

Content Type – lets browser know what kind of data was sent back (so it may parse HTML, disp plain text, render image, etc.)

Content Length – how much data was sent back in response body. Can be helpful with debugging

Response body is usually just the content of a file

Life Cycle of Web Page

User Clicks Link – Click button, fires event that browser responds to

Browser Sends Request –

Browser gets information together and sends to server in *request*. Includes header w/ request method, identifier of thing requested, format the client wants data, info about the browser and page state, etc

Server Responds to Request –

After route from server to server, request ends at destination. Server looks for resource somewhere, might return thing directly, or modify then return, or build page on-the-fly based on

State of the server or contents of request

Client Receives Response –

Client gets response, parses it. Might render directly or change based on logic included in file it

got back. File might tell browser it needs more files, which causes browser to request more

Client Renders the Page –

Render happens as requests finish. Speed of connection, size of requests may change how fast

things appear. Once finished, server stops influencing the page, but the browser may make

changes (e.g. drop down menus)

**Intro to Git and Version Control**

Basics of Git

Master

* Main branch of repository, should represent software in most advanced/complete state. Most recent commit = most complete version of software that is correct and functional

Branch

* Separate version of software project. Often from main, can branch from other branches. When branch, takes snapshot of main. Changes only exists in branch until merged back to main

Commit

* Snapshot of workspace. Changes not saved to repository until added and committed. Every time you do a commit, you add a snapshot you can branch off or roll back to if needed.

Merge

* Process of taking branch and applying changes to its parent, gets messy only if the changes were made on same line of code in branch merging into and branch you are merging

Clone

* Makes a copy of the repository. Generally all branches are cloned as well.

Push

* After cloning, you might want to get changes back to place you cloned from, to do so you push the repo. When you push, you only push a single branch to a single branch. Often current branch to origin’s master branch

Pull-Request

* GitHub is a tool separate from Git, like a wrapper adding additional features. Main purpose to help people work on and share repos from remote locales. At an office, often one repo on network everyone can clone and push work. When not on same network, more difficult. GitHub has pull-request, which, if you make a change that improves the software, can make pull-request, which tells the owner of the repo that I want to make changes. Once authorized, works like a merge.