**Introduction to Programming for the Cloud**

Static Elements – Content stored on the server

Dynamic Elements – Content generated each it’s requested by the client

Cloud apps are built to work seamlessly with a cloud based backend infrastructure.

Cloud apps use cloud based data storage and data processing, making them very scalable and resilient

Environment for building cloud apps:

1. Front end
   1. Client-side, user interactions
   2. HTML, CSS, Javascript, etc
2. Backend
   1. Server side, before data is sent to client
   2. Handles logic and functionality, including authentication
   3. Relational or NoSQL databases

**Learning Frontend Development**

New front end dev language – Syntacticallly Awesome Style Sheets (SASS)

-Extension of CSS that is compatible with all versions of CSS

Learner Style Sheets (LESS)

-Enhances CSS, backwards compatible with CSS

Javascript Frameworks:

1. Angular – opensource maintained by google
   1. Allows websites to render html pages quickly and efficiently
   2. Has built-in tools for routing and form validation
2. React.JS – developed and maintained by facebook
   1. Javascript library that builds and renders components for a web page
   2. Not complete suite of tools
      1. Ex: routing is not a part of it
   3. Only helps build and drop components into a page
3. Vue.js – community maintained
   1. Flexible, scalable
   2. Main focus is view layer
   3. Adaptable – can be a library or it can be the framework

**Introducing Application development tools**

1. Version control – git/github
2. Libraries – collections of reusable code
   1. JQuery
   2. Email validator
   3. Apache commons proper
3. Frameworks – provide a standard way to build and deploy applications
   1. Act as a skeleton you extend by adding own code
   2. New frameworks can’t be adding in at a later date
   3. Dictate the architecture of the app
   4. Less control than libraries, but provide standardization
   5. Examples
      1. Angular.JS
      2. Vue.js
      3. Django

**Application development tools**

1. CI/CD
   1. Enables devs to deliver frequent changes reliably
   2. Implemented through a build-automation server
   3. CI automatically builds and tests code
   4. CD deploys the changes
2. Build tools - Compiler
   1. Transform source code into binaries for installation
   2. Automate tasks like:
      1. Downloading dependencies
      2. Compiling code
      3. Packaging binaries
      4. Running tests
      5. Deployment to production systems
   3. Examples:
      1. Webpack
      2. Babel
3. Packages
   1. Packages make apps easy to install
   2. Packages contain:
      1. App files
      2. Instructions for installation
      3. Metadata
4. Package managers
   1. Make working with packages easier, maintaining/installing/etc
   2. Coordinate with file archivers to extract package archives
   3. Verify checksums and digital certs to ensure integrity and authenticity of packages
   4. Locate, download, install, update existing software from a repo
   5. Manage dependencies to ensure a package is installed with all packages it requires
   6. Package managers by platform:
      1. Linux
         1. Debian
         2. Red hat
      2. Windows
         1. Chocolately
      3. Android
         1. Package manager
      4. macOS
         1. Homebrew
         2. MacPorts
   7. Libraries and utility code are managed with cloud app package managers
      1. Node.js/javascript
         1. Npm
      2. Java
         1. Gradle
         2. Maven
      3. Ruby
         1. RubyGems
      4. Python
         1. Pip
         2. Conda

**Importance of Backend development**

1. Backend dev – creates and manages resources needed to respond to client requests
2. Frontend dev – creates websites and cloud apps
3. Work together closely throughout the life of the website or cloud app
4. What does backend do?
   1. Backend processes browsing data
      1. Login info
      2. Product searches
      3. Payment info
   2. Backend devs
      1. Write and maintain code to process this data
5. Backend dev skills
   1. Web app dev language
   2. Database query language
   3. User account management
   4. Authentication and authorization
   5. Secure data handling
   6. Secure storage
6. APIs, Routing, and Endpoints
   1. APIs, routes, and endpoints process requests from the front end data
      1. API – works with data (JSON, XML)
      2. Route – path to a website
      3. Endpoint – API or path
   2. Backend devs create routes to direct requests to correct service
      1. Ex: 404 error, 200 success
   3. APIs provide a way for cloud apps to access resources from the backend
7. Backend languages and frameworks
   1. Javascript
      1. Node.js (Framework)
      2. Express (Framework)
   2. Python
      1. Django (Framework)
      2. Flask (Framework)
8. Working with databases
   1. Structured Query Language (SQL)
   2. Object-relational mapping (ORM)

**HTML5 and CSS Overview**

Objectives of HTML5

1. Single language which can be written in HTML syntax or XML syntax
2. Interoperability with earlier html implementations
3. Improves markup for documents
4. Includes markup and APIs for idioms, such as web apps

**HTML Features**

1. Categorizes web pages into different sections
   1. Tools for data management, drawing, video, and audio
2. Cross-browser applications for the web and portable devices
3. Greater flexibility
   1. Exciting and interactive websites
4. Help to create a more engaging user experience
   1. HTML5 pages provide an experience similar to desktop apps
5. Allow for enhanced, multiple-platform development
   1. Combining abilities with APIs

Document Object Model (DOM) Tree

DOM Tree is an In-Memory representation of a document

1. Contain nodes
   1. Doc type node
   2. Headers
   3. Paragraphs
   4. Etc.

XML example – Must have an xml tag

Using HTML or XHTML

1. Using XSLT? (Extensible Style Sheet Language Transformations)
2. Both use the same tags/semantics
3. XHTML tags all need to in lowercase (Case doesn’t matter in HTML)
4. XHTML must be well-formed
   1. End tags for all elements
   2. All attributes must have a value
5. XHTML will stop processing on a malformed element

**HTML Management and Support**

Key themes of HTML5

1. Syntax is compatible with HTML4 and XHTML1
2. Separates conformance requirements for user agents and authors
3. HTML5 includes APIs that help in creating web apps
   1. Video and audio elements
   2. An API that supports offline web apps
   3. An API for drag and drop

HTML5 for Web Apps

1. Modern browsers support full rang of HTML5 features
2. APIs for enhanced user experience
3. Improved page load times
4. Search index indexing
   1. Meta tags can provide keywords for search engines

HTML5 Elements

1. Structural elements
   1. Section
   2. Article
   3. Header
   4. Footer
   5. Figure
   6. Figcaption
2. Graphics and embedded content
   1. Canvas
   2. Audio
   3. Video
   4. Track
3. Input elements
   1. Input type attributes
      1. Tel
      2. Email
      3. Datetime
      4. Number
      5. Range
      6. Color
4. Web storage
   1. localStorage
   2. sessionStorage
5. Web workers
   1. Web workers provide a way to run processing-intensive tasks without blocking the user interactions to the current page

**HTML Scripting**

Scripting can be disabled – recommend using scripting, but don’t rely on it

Scripting is enabled in a browser context when:

1. Browser supports scripting
2. Use has not disabled scripting
3. Browser context does not have the sandboxed browsing content flag set
   1. The purpose of HTML5 sandboxes is to manage iframe ‘mash-ups’

Can implicitly give a third party vendor ability to advertise on your site if the sandbox attribute is NOT set on you embedded objects – use attribute on any tag that contains an embedded object

Each HTML and XML document loaded into a browser page becomes a Document object.

DOM Properties – DOM tree accessors are HTML Document APIs provide access to all HTML elements on a page (Prefix by the word “document”):

1. Document.Head – returns head element
2. Document.title – sets or returns the title of the doc
3. Document.images – returns HTMLCollection of the img elements in the document
4. Document.lastModified – Returns the date of the last mod to the doc
5. Document.scripts – returns and HTMLCollection of the script elements in the doc

Common HTML DOM tree methods:

1. getElementById(‘id’) – accesses the first element with the specified id
2. getElementsByTagName(‘tag’) – returns nodelist of all elements with the specified html tag
3. open() – opens an output stream to collect the output from document.write
4. write() – writes javascript code to the document
5. close() – closes output stream previously opened with document.open

**HTML5 Browser Support**

Support tables – show features supported by different browsers (caniuse.com)

Can use javascript to check support

1. Document.createElement() – dynamically create the element
   1. Creates the element regardless of whether the browser supports it or not
2. Check the object you created for a known property or method
   1. If browser supports it, it is displayed
3. For input elements, set the type value you want to test for
   1. If browser supports it, it is displayed
   2. Otherwise, browser returns input type=”text” and displays a regular text field by default

**CSS Styling HTML**

Creates uniform look throughout the webpages

Base style: Body

Guidelines:

1. Use RGB for colors
2. Use pixels (em), or percentage for size

Fluid layout - Height and width is flexible (percentages and ems)

Fixed layout – specify height and width (use pixels)

Universal Selector is \* - matches elements of any type

**HTML5 Elements**

1. Article – content from an external source
2. Aside – content aside from the page content
3. Audio – used to embed sound content
4. Canvas – used to draw graphics
5. Datalist – provides a list of predefined options for input controls
6. Details – used to show or hide contents
7. Embed – embeds an external application or interactive content into page
8. Figcaption – caption for the figure tag
9. Figure – specifies self-contained content
10. Footer – footer of document or section
11. Header – specifies group of introductory or navigational elements
12. Keygen – specifies a key-pair generator field used in forms
13. Mark – highlighted text
14. Meter – used for measurements with min and max values
15. Nav – document navigation
16. Output – result of calculation
17. Progress – state of work in progress
18. Section – defines sections in document or article
19. Source – used to specify multiple media resources for media elements
20. Summary – defines a visible heading for the details tag
21. Time – specify date or time in a document
22. Video – movie clip or video stream
23. Div – lowest form of element (very generic)
24. Fieldset – found within the form tag, used to group related elements in an HTML form inside a box
    1. Attributes – disabled, form, name
25. Legend – used with fieldset as a first child to define the caption for the grouped fields

**Attributes for the Input Tag**

Input type attributes <input >:

1. Tel – expects telephone number, does not enforce numeric only
2. Email – Regular text input field
3. Date – date and time control (depends on the browser)
4. Datetime – date control (depends on the browser)
5. Number – Takes numeric value, can specify min and max values
6. Range – Number selector, displays a slider. Additional JS code needed to display the values of the slider
7. Color – color chooser (depends on the browser)
8. Search
9. URL – validate urls
10. List – use datalist feature, useful for autocomplete functionality
11. Placeholder – provide hints of input text format

Validation fallback

1. Prebuilt script libraries – javascript and JQuery libs
2. Assume browsers will support more HTML5 features over time, leave all final validation server side
3. Include custom javascript validation – client side validation that is attached to the form submit event handler

**Javascript**

Javascript interpreters embedded in web browsers

Asynchronous Javascript and XML (Ajax)

-Works with HTML, CSS, XML, and JSON

Primitives:

1. Number (decimal, octal, or hexadecimal)
   1. 16, 020 (octal), 0x10 (hex)
   2. Floating point, double precision
2. String
   1. Double or single quotes
3. Boolean
4. Null
5. Undefined
   1. Data type not assigned, or variable does not exist

All others are “Objects”

Wrapper objects – same name as primitive type, but start with an uppercase letter

1. Typeof
2. Valueof()

Arrays – zero indexed

numArray = new array (0,1,2,3); - constructor

colors = [“yellow”, “green”] – array literal

Date objects – var today = new Date()

Javascript automatically applies the toString() method if you attempt to display the date on a page or alert box

Error objects – created when an exception is thrown

1. Message: description of the message
2. Name: identifies type of error
   1. TypeError
   2. RangeError
   3. URIError
   4. SyntaxError
   5. EvalError
   6. ReferenceError
3. Throw new Error(“This is a custom error message”)

**Variables and Control Statements**

Var keyword to declare variables

Do not need to declare type of var, javascript is loosely typed

Variable name rules:

1. Must start with letter, underscore, or a dollar sign
2. Case-sensitive
3. Variable scope
   1. Declared without the var keyword have a global scope
   2. Not initialized have a value of undefined

Control statements:

1. If, else if, else
2. Indentation not required (like python)
3. No block scope, variables defined inside a condition, can be used outside of that condition
4. Switch (expr), case LABEL1, break, default
5. For (var I = 0; I < maxVal; i++)
6. While (condition)

**Functions and prototypes**

Keyword function

Function add (n, m) { Return n + m } //can be numbers, strings

Uses “this” keyword in functions

Prototypes:

1. “new” keyword
2. Scripts can add custom properties and methods to the prototype
3. Scripts can override prototype properties and functions
   1. These changes affect the current working instances without affecting the static object prototype

Function car (make, model, year) {

This.make = make;

This.model = model;

This.year = year; }

Car.prototype.getName = function () {

Return this.make + “ “ + this.model + “ “ + this.year;}

-When a car object is instantiated, it will include the getName functionality

--All existing instances will also inherit the getName method

Self-executing functions:

1. Start running immediately after they have been declared
2. Functions and variables are isolated from the rest of the script
3. Can be anonymous (unnamed)
4. Can look like this:

(function () {

//Statements

})();

1. Often used to initialize data or to declare DOM elements

**Client-side javascript: with HTML**