Sprint 1

HTML/CSS

source : Chris's code manual that he shared with the team

What is CSS?

CSS (Cascading Style Sheets) are used to manage web page layout. Styles are normally stored in style sheets and define how to display HTML elements. They were added to HTML 4.0 to separate the content of HTML documents from the document's presentation layout. External Style Sheets are stored in CSS files so that you do not need to rewrite them in your new HTML file. Multiple style definitions will cascade into one, based on their order.

Syntax:
selector { property: value } The selector is normally the HTML element/tag you wish to define. The property is the attribute you wish to change, and each property can take a value. The property and value are separated by a colon, and surrounded by curly braces.

Comments

A CSS comment begins with /* and ends with */

Group and Class

You can group selectors, separate each one with a comma. With the class selector you can define different styles for the same type of HTML element. You can omit the tag name in the selector to define a style that will be used by all HTML elements that have a certain class

Style based on Attributes

The style rule below will match all input elements that have a type attribute with a value of "text":

- input[type="text"]{ background-color: blue } You can also define styles for HTML elements with the id selector, which is defined as a #. The rule below applies to all tags with the id value "life":
- #life { text-align: center; color: red; }

Adding style sheets to HTML

External style sheets

- Ideal when the style is applied to many pages. Each page must link to the style sheet.
- Use a link tag in the head of the HTML file

- An external style sheet file should not contain any HTML tags and should be saved with a .css extension

Internal style sheet

- Used when a single document has a unique style.
- Defined in the head section using the HTML style tag

Inline styles

- Loses many advantages of style sheets by mixing content with presentation
- Use this sparingly, such as when a style is to be applied to a single occurrence of an element
- Use the style attribute in the relevant tag

Full Stack Development

Source: Geeks for Geeks

Full stack development: It refers to the development of both **front end**(client side) and **back end**(server side) portions of web application.

Full stack web Developers: Full stack web developers have the ability to design complete web applications and websites. They work on the frontend, backend, database and debugging of web applications or websites.

Popular front end languages: HTML, CSS, Javascript

Popular back end languages: C++, Java, Javascript, PHP, Python, Node.js

Big emphasis on Javascript

A Database is the collection of interrelated data which helps in efficient retrieval, insertion and deletion of data from a database and organizes the data in the form of tables, views, schemas, reports etc.

React

Source: Reactjs.org
A javascript library for building user interfaces.

- Declarative: React will efficiently update and render just the right components when your data changes. Makes your code simple and easy to debug
- Component Based: Build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

- The render method returns a *description* of what you want to see on the screen. React takes the description and displays the result. In particular, render returns a React element, which is a lightweight description of what to render.
- JSX comes with the full power of JavaScript. You can put *any* JavaScript expressions within braces inside JSX. Each React element is a JavaScript object that you can store in a variable or pass around in your program.

Express.js

Source: Expressjs.com

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- ***Routing*** refers to how an application's endpoints (URIs) respond to client requests.

Source: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

- Node: Node (or more formally *Node.js*) is an open-source, cross-platform runtime environment that allows developers to create all kinds of server-side tools and applications in JavaScript. The runtime is intended for use outside of a browser context (i.e. running directly on a computer or server OS). As such, the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs including HTTP and file system libraries.
- Express:Express is the most popular *Node* web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

● Write handlers for requests with different HTTP verbs at different URL paths (routes).
● Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
● Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
● Add additional request processing "middleware" at any point within the request handling pipeline.

CPC final guide

- Managed Services: RDS, DynamoDB, EKS, ECS
- Connectivity options:
  1) Direct connect: on-prem server → cloud $ (not over public in.)
  2) VPN: on-prem server $\xrightarrow{\text{encrypted}}$ cloud (over public internet)
  3) Public Internet - not encrypted
  4) VPG: goes w/ VPN to connect to VPC
  5) Internet Gateway: Public Subnet → public internet (inbound)
  6) NAT Gateway: private subnets → public internet (outbound)
- Storage:
  1) S3 Glacier: low cost data archival
  2) Snowball: move data to cloud
  3) Standard
  4) Standard IA
- Databases:
  → EC2 installed DB vs. RDS: on EC2 you manage patches + updates, + read-replicas
- IAM:
  → Groups: never for resources, only IAM users → can have policies
  → Roles: have a collection of policies
    ↳ typical: resource uses role for permissions
    ↳ can assume a role as a user "admin role"
- AWS Support:
  - AWS Abuse: ddos, phishing, malware
  - AWS Support Cases: connect you to help
  - Premium Support: guidence from SA's
  - TAM's: Strategize for customer → enterprise only
  - AWS Trusted Advisor: evaluate to reccomend best practices
- Pricing
  - on-demand: cannot be interrupted
  - reserved-Instances: commit to a yr. to maximize discounts
  - Spot instances: 90% savings, interruptable

- More billing:
    - Cost Explorer: visualize, understand GUI for $ data
    - AWS C+U report: granular data about $, load into services
    - Amazon QuickSight: customer level data statistics
    - billing support case
    - Concierge: billing / acct queries → enterprise only
    - AWS Budgets: set custom spending plans

- IaaS: building blocks for cloud IT
    - EC2
    - VPC

- PaaS: managed, removes planning + managing infrastructure
    - RDS
    - S3
    - EBS
    - Lambda
    - Dynamo DB
- SaaS: completed product
    - elastic beanstalk
    - cloudwatch

- Databases:
    → RDS: relational SQL
    → Dynamo DB → nonrelational noSQL   key + attr
    → Redshift → data warehouse SQL
    → Auora → enterprise, relational SQL

- Security group: tied to an instance
- Network ACL's: tied to a subnet

JavaScript Quiz Study Guide

Hoisting: JavaScript only hoists declarations, not initializations.

ex./

```
x = 5;
var x;
```

=

```
var x;
x = 5;
```

this automatically gets brought to the top

- let + const are hoisted to the top of the block but not initialized.

ex./

```
var x = 5;
var y = 7;
elem.innerHTML = x + "  " + y
```

≠

```
var x = 5;
elem.innerHTML = x + "  " + y
var y = 7
```

- to avoid bugs, always declare all variables @ the top of every scope.

- variables can be declared after they have been used
- variables can be used before it is declared

const

```
const arr = [1, 2]
//good
  arr[2] = 0
//bad
  arr = []
```

if done w/ let... you'll get a compile error

hoisted

let

```
let a = 3
//good ... logs 3
const func = () => {
    console.log(a)
}
// bad ... undefined
const func = () => {
   console.log (a)
}  var a = 12
```

closures - "self invoking" functions
↳ way for a function to have "private" vars
- In JavaScript, variables can belong to the local or global scope.
- global varidbles can be made local (private) w/ closures.

ex./

```
const  myName  = 'Sam'

function  PrintName() {
    console.log (myName}
}

myName  = 'kate'

printName()
```

→ will output kate

- way for functions to use variables outside of functions + not passed to it

- Inner function has access to anything that the outer function has access to.

→ gives you access to an outer function's scope from an inner function.

• var statements create global variables
• blocks are only treated as scopes if variables are defined with let + const

JavaScript

- Javascript is a scripting language
  - ↳ cannot compile into byte code
  - ↳ cannot run on proussor w/o interpreter
  - ↳ must be running in a scripting environment
- weakly typed
  - → primitive types : int, string, float, boolean, null
    - passed/assigned by value ↳ + undefined!
  - → non primitive types: functions, objects, arrays
    - passed by reference + stored by pointers
    - ↳ object = key-value store {}
- -Strings have no size limit
- -numbers are always stored as floating pt
- -very large numbers are rarely accurate

let vs. var

- historically "var" declared variables
  - ↳ var is bad
- Let + const also create variables
  - ↳ always use const until you get to a place where you need to update that variable
- ↳ Let: block-scoped : allows you to declare variables that are limited to the scope of a block statement (loop/if)

- Javascript is single threaded
  - → no sleep or wait
  - → while 1 thing is running, nothing else is getting done in that tab.
  - → use SetTimeout (functo wait, #seconds)
- "callbacks" are how most things are done in java.
- → Timeout is run @ end