1. Introduction to HTML

* Write htmlboiler on sublime and hit TAB -> Voila. For shorter form, use html:5 and hit tab
* Select any line or a bunch of lines on sublime and hit command + / -> Turns that into a comment -> Do again -> Un-comment
* Title goes as a tab name and helps google engine to find your page
* HTML element reference page on MDN has list of all the elements.
* h1,h2...h6, p are block level elements. Each of those takes up a new line.
* div and span both act as a container. Div is a block level element where as span is not. So, span helps us to style an element within a block.
* When you are referring to an 'a href' tag, make sure to mention http:// if you need to access the internet. Without 'http://' it will search for the link within your code folder which might be helpful if you want to reference another page.html.

2. Intermediate HTML

* The <input> tag creates interactive controls and the 'type' attribute determine the type of inputs.
* You can use placeholder with input tags to fill up any form temporarily.
* In a form, if I don't specify an action, it is going to send the data exactly where I am now and if I don't specify a method, its going to be a GET request.
* 'Labels' are useful for accessibility reasons. Screen readers or other accessibility software will spell out whatever there is inside a Label tag for a visually impaired person.
* Add 'required' where an input field is required. <- Presence validation
* If a button is the last thing in a form, it will by default 'submit' the form.
* textarea with rows and columns gives us a wide text area to type.

3. Introduction to CSS

* If you set a property on a parent, that affects the child as well. -> Property of Inheritence.
* Specificity is the means by which browsers decide which CSS property values are the most relevant to an element and, therefore, will be applied. Specificity is based on the matching rules which are composed of different sorts of [CSS selectors](https://developer.mozilla.org/en/CSS/CSS_Reference#Selectors).
* Value of Specificity: Type Selectors (li, a) < Class, Attribute, Pseudo-class selector (.example) < specific IDs (#example).

4. Intermediate CSS

* Font-family, font-size, font-weight, line-height, text-align, text-decoration
* The size of an em value is dynamic. When defining the font-size property, an em is equal to the size of the font that applies to the parent of the element in question. If you haven't set the font size anywhere on the page, then it is the browser default, which is often 16px. So, by default 1em = 16px, and 2em = 32px. If you set a font-size of 20px on the body element, then 1em = 20px and 2em = 40px. Note that the value 2 is essentially a multiplier of the current em size.
* font-weight: normal
* font-weight can also be a numerical value ranging from 100 (lightest) to 800 (strongest)
* line-height is exactly like line spacing in word processor.
* text-align helps to align the texts left, center or right.
* text-decoration: underline, overline, line-through.
* 4 parts of a box in box model -> Content, Padding, Border, Margin
* margin:auto sets things at center for us

5. Bootstrap

* Components -> Form -> form-control, form-group, form-inline
* Grid system (12 units) is the most important part of bootstrap.
* 4 different sizes available for a bootstrap grid: xs (phone), sm (tablet), med (laptop), lg (desktop)
* Look up text-shadow for shadowing effects.
* If you want your bootstrap styled website to be responsive on mobile then be sure to add the following meta tag to your <head> element, above the <title>  tag:  
    
  <meta name="viewport" content="width=device-width, initial-scale=1">

6. Javascript DOM

* console.dir (document) shows the entire document object of that particular page.
* The document comes with a bunch of methods for selecting elements.
  + document.getElementById()
    - takes a string argument and returns the **one element** with a **matching ID**
  + document.getElementByClassName()
    - takes a string argument and returns a **list of elements** with a **matching Class name.**
  + document.getElementByTagName()
    - takes a string argument and returns a list of all elements of a given tag name like <li> or <h1>
  + document.querySelector()
    - **Returns the first element** that matches a given **CSS-style selector**.
    - Helpful when you just need to retrieve the first element (like ‘body’ because there is going to be only one ‘body’ afterall.)
  + document.querySelectorAll()
    - **Returns a list of elements** that matches a given **CSS-style selector**.
* Manipulate an element’s style: **style** (tag.style.color, tag.style.border etc),
* **classList** is a read-only list that contains the classes for a given element. **It is not an array (so we can not do .push() or .pop()).**

var tag = document.querySlector (“h1”)

//Add a Class

tag.classList.add(“another-class”)

// Remove a class

tag.classlist.remove(“another-class”)

//Toggle a class

tag.classList.toggle(“another-class”)

* **“textContent”** returns a string of all the text contained in a given element. We can update a text using textContect. But doing this is not advisable as it might totally destroy the styling of the text. textContent can not read html. It reads whatever you give as just plain text.
* **“innerHTML”** is similar to “textContent”, except it returns a string of all the HTML contained in a given element. -> More advisable.
* Use “**getAttribute**” and “**setAttribute**” to read and write attributes like ‘src’ and ‘href’.

7. jQuery

* Methods discussed:
  + css()
  + val()
  + text() (eq to textContent)
  + attr()
  + html() (eq to innerHTML)
  + addClass()
  + removeClass()
  + toggleClass()
  + .click()
  + .keypress()
  + .on()
* Event Bubbling is default behavior. Use stopPropagation() to prevent it.

8. Node/Npm

|  |  |
| --- | --- |
| Install npm | npm install |
| Setting up environment | * Create a new folder * cd into it * run ‘npm init’ to create a package.json file |
| Login / Create an account | * npm adduser * To check, npm whoami |
| Start a project | * npm init scope=<username> * will create a package.json file |
| Install a module | npm install <modulename>  Note: Run `npm install @linclark/pkg --save` to install the module, and also update your package.json file at the same time. |
| Listing Dependencies | npm ls |
| Npm Test | Create a test file and add it to your package.json file under the ‘scripts’ section.  Npm always puts an ‘always failing’ test there by default. |
| Publish | npm publish |
| View what you just published | npm view <package name> |
| Check/edit/update version | npm version <version>  Illustration of versioning: 1.2.3  (Major version. Update for breaking API changes) .(Minor version. Update for API additions) .(Patch version. Update for every change) |
| Dist-tag (package distribution tag) | npm dist-tag add pkg@1.0.1 beta  npm dist-tag rm <pkg> <tag> |
| Check for outdated dependencies | npm outdated |
| Update | npm update |
| Remove | Npm rm --save |

9. Library vs Framework

Both Library and Framework are external codes that we can include in our own application but a library is something that you in control of. Its upto you how many methods you want to use from a particular library. But the framework, on the other hand, we have to give up a little bit of control. In most of the cases, some of the decisions are already made for us in the framework and we have to abide by them. The basic skeleton will already be given to us and we just have to fill up some gaps. Frameworks is not for creativity. Instead, it will take care of all the basic necessary works for us so that we can focus on the creative part.

10. API

* API’s are interfaces for code/computers to talk to each other.
* Generally communicate via HTTP.
* APIs don’t respond with HTML. HTML contains information about the structure of the page. But **Api responds with data, not structure**. They use simpler data formats like XML and JSON.
* Json (Javascript Object Notation) looks like JS objects. However, **everything is a string**.

11. Databases

* What is a database?
  + A collection of information/data
  + Has an interface
* SQL (Relational) vs NoSQL (non-relational)
* Mongod/Mongo
* Mongoose is an elegant mongodb object modeling for node.js. It allows us to access database from inside of a js/express file; basically, a js layer on the top of mongodb.

12. REST(Representation of State Transfer)ful Routes

* A mapping between HTTP routes and CRUD (Create, Remove, Update, Delete)
* Organizing routing in large applications and APIs so that it's easy to find and maintain in the future" is hard. Routing is extremely important. It defines the URL structure that someone uses to interact with your web application.
* Express apps utilize routers that are essentially containers for a set of middleware. We can put this middleware holder only on a certain route, which allows us to keep our logic in separate files and bring them together **on our terms**!
* List of 7 RESTful Routes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | URL | HTTP Verb | Description |
| INDEX | /dogs | GET | Display a list of dogs |
| NEW | /dogs/new | GET | Add new dog to DB, then redirect somewhere |
| CREATE | /dogs | POST | Add a new dog to db and then redirects to somewhere. |
| SHOW | /dogs/:id | GET | Shows info about one dog |
| EDIT | /dogs/:id/edit | GET | Show edit form for one dog |
| UPDATE | /dogs/:id | PUT | Update a particular dog, then redirect somewhere. |
| DESTROY | /dogs/:id | DELETE | Delete a particular dog, then redirect somewhere |

13. Authentication

* Tools Used: PassportJS, passport-local(usrname and pswrd authentication for passport), passport-local-mongose(a mongoose login for the same).
* Concept of sessions -> A way to make HTTP NOT stateless.
* Middleware: Runs between start of a route and its callback.