1. **General Tips**

* Pages: Static vs Dynamic (google news is dynamic coz it keeps updating).
* Responsive site means responsive to change in size of browser.
* % makes more responsive as compared to pixels.
* HTML, CSS, JavaScript of any site is accessible.
* <http://codepen.io/>
* Avoid <br> tags.
* Look for errors by Inspect -> Console. (Also shows line number)
* For setting background image, refer to Purrfect Match in Bootstrap. We have to set html’s height as 100% so that body takes up the entire screen. Also, background-size: cover and background-position: center.
* Dealing with different image sizes: [https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/5102522#questions/5506894](https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/5102522%23questions/5506894)
* Padding and margin when used as a percentage, is always based on the width of the parent element.
* When you specify the height, width, font-size as a percentage, that's a percentage with respect to the element's parent (container).
* 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">
* Pattern Project: In bootstrap the classes col-lg-someNumber for example, has some specific padding set to it by default.
  + .container defines default left/right padding of 15px.
  + .row defines default left/right margin of -15px (negative).
  + .col defines default left/right padding of 15px.
* Margin Collapse (Museum of Candy Project):
  + Margin collapsing works only on block level elements.
    - An element's bottom margin and it’s next elements top margins are collapsed. Lefts and right don’t.
    - <https://medium.com/@joseph0crick/margin-collapse-in-css-what-why-and-how-328c10e37ca0>
  + Generic fonts: <https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861340#questions/12455572>
* You can check browser support across versions for different frontend features, like for CSS those properties (that are included in the classes that you might be wondering about) on a website like this one: [www.caniuse.com](http://www.caniuse.com) . It also lists necessary prefixes.
* Use modernizr which makes our page more cross-browser compatible. <https://www.hongkiat.com/blog/modernizr/>
* [diffchecker.com](file:///C:\Users\asus\Desktop\Amazon\The%20Web%20Developer%20Bootcamp%20by%20Colt%20Steele\Material\diffchecker.com) to check differences in two codes.
* If something I want to do with JavaScript can be done with CSS instead, use CSS. CSS parses and renders faster.
* Todo list project after jQuery-
  + CSS Styling works for future elements too!
    - The CSS is loaded by the browser when the page loads, but it's constantly/dynamically applied to the elements that are present on the page at a specific time (which includes dynamically added elements added later on).
  + Correct folder structure.
  + CSS should be organized
    - In the order of the page, top to bottom.
    - Specificity, least to highest. (most popular)
  + We can align icons (here trash) using text-align as the icons are unicode.
* For event listeners (JS/jQuery), better to define the function inside the parameters that defining separately and then passing function name.
* <https://uigradients.com/> for beautiful CSS color gradients.

1. **VS Code Shortcuts**

* Lorem20 will give 20 lorem ipsum words.
* div.container.text-center will make a div with classes container and text-center.
* Alt+Click for multiple cursors.
* Ctrl+d for copy pasting the current line or the selected lines.
* li\*5 to get 5 lis.
* div>header>ul>li\*5 will give us the whole bunch of code with this nesting very quickly.
  + Header, nav, section tags are more meaningful than div but have same functionality.

1. **HTML**

* MDN HTML elements reference to find all the elements that can be used like body, h1, p..
* Title is what we see on google search and when opened, on the tab name.
* Headings, paragraphs are block-level elements. Not inline (unlike bold). So each heading (h1, h2,..) or paragraph take a separate line.
* Em > italic (emphasis is meaningful)
* Strong > bold (strong is meaningful)
* Divs and Spans are generic containers (For Styling)
  + Divs groups things together (for example, group images, texts and have a box around them)
  + But Divs are block level unlike Spans. So span groups inline.
* Attributes – Additional information for HTML elements. MDN HTML Attributes reference for details.
* Anchor tag is inline. Also, when mentioning a site, mention explicitly (including protocol (HTTP)). For example, <http://ww.google.com> instead of [www.google.com](http://www.google.com). Explicit is required because otherwise it would consider it a relative path. Like another html file in the same folder with have a common preceding path.
* Imgur to store images online which can be used in HTML file.
* Tables, use <thead>..</thead> and <tbody>..</tbody> to separate headings and rows of a table so that search engine understands. It doesn’t change anything visually. <tr>..</tr> denotes a row and <th>..</th> represents table heading. Table data <td>..</td> is for columns in a row.
* Form tag is like a container in which we put all the different types of inputs like radio button, drop-downs, check boxes, text boxes.
  + Method means what HTTP method (get/post) and action means where the form sends the data to.
  + Get method is by-default and if action isn’t specified then it refreshes the current \*.html file.
  + Get method (Google) doesn’t store in database, it just searches and gets data back.
  + Post method is like signup page of Facebook which gets stored in Facebook’s database.
  + Type attribute of Input element decides the type of input (MDN input).
  + Labels are important for visually impaired people. It tells what each part of the form corresponds to. Input can be nested inside label or can be separate but have same value of for in label and id in input.
  + Validation- required (blank not allowed) and data type (correct format).
    - Minlength and maxlength attributes to use for the input validation has less browser support than the pattern attribute.
    - Input type = email ensures that email if entered is in correct format.
    - <https://validator.w3.org/> (Markup validation service)
  + For radio input, we can use the same value for the name attribute in the input element so that we select only one of the radio buttons as a choice.
    - Value of value attribute is reflected in the URL on submitting. As name’s value=value’s value (radio) or the input text in the textbox.
* Textarea element is different from input type text. It lets us choose size of textbox.
* Dropdown, selected disabled will act like a placeholder.

1. **CSS**

* It’s better to separate HTML from CSS. Hence inline styling is a bad idea.
* If there are conflicting styles for the same element (suppose h1), then the one used later will be considered.
* Colors are
  + 6 digits hexadecimal (RGB). Use [www.colorpicker.com](http://www.colorpicker.com)
  + Or represented as rgb(rval, gval, bval) where each ranges from 0 to 255.
  + Rgba(rval, gval, bval, alpha) where alpha (0.0-1.0) for transparency.
* Background can be a color or image (set background-repeat to no if you don’t want the image to be tiled, and set background-size to cover if you want to stretch the image the entire screen).
* Border: width, style, color.
  + border: 5px solid purple;
* CSS Selectors: Element, #id and .class.
  + Same id values can be used only once a page.
  + Class values can be same in a page.
  + Descendant selectors- Select anchor tags with class “bolded” inside <li>..</li> only.

li a.bolded{

    color: red;

}

* + If we want to select both h1 and h3, we can use a comma in between.
  + Adjacent Selector: Allows us to select siblings. Here, all uls that come after h4 but in the same level, not inside.

h4 + ul{

    border: 4px solid red;

}

* Attribute Selector: All inputs of type text suppose. Or all anchors with link to google.com.
  + - * input[type=”text”]{
        + background: blue;
      * }
* Select nth of type selects suppose every 3rd li. Or select every even number.
* Inheritance and Specificity
  + Inheritance: If a body is set color pink, all elements within it will be pink.
  + Specificity: If a body is set pink, but paragraph within it is set blue, the paragraph wins as it’s more specific. Example, ***Id>class> li>ul>body***
  + Calculator*:* [*https://specificity.keegan.st/*](https://specificity.keegan.st/)
  + Type Selectors < Class, Attribute, Pseudo-Class Selector < ID Selector.
* Fonts availability In the different systems: <https://www.cssfontstack.com/>
  + Font-family, font-size, font-weight, line-height, text-align, text-decoration
  + Put font-family inside “” if the name consists of number.
  + Font-size either in px, em, rem.
    - Size of em value is for relative hence dynamic.
    - Em set 2.0 means twice the size of parent element.
    - If we set body font-size as 40 px, then we can relatively specify the size of h1, p, span (nested inside) and so on by using em.
    - Rem is not decided by parent element unlike em. They are decided by root element on the page.
    - If font-size of the root element is 16px then 1 rem = 16px for all elements. If font-size is not explicitly defined in root element then 1rem will be equal to the default font-size provided by the browser (usually 16px).
  + Font-weight can also increase page load.
  + Google fonts: <https://fonts.google.com/?sort=date>
    - When working with CSS link elements, the loading order doesn't matter. So if you place the link which loads the online Google Font below the CSS which applies it, it will still work (that's how the browser works with CSS in those scenarios).
* Box Model!!: Padding, Margin, Border, and More!
  + Every element has a rectangular box around it.
  + Spacing between elements achieved by margin.
  + Padding: Space between border and element inside it. Can choose between top, left, bottom and right like border.
  + Width: 50% means 50% of parent element. Like p inside body.
  + Blue=content, green=padding, orange=margin
  + Margin: top right bottom left. Or can put two values only which means top-bottom and right-left. Auto value is also acceptable.
* Class=”vertical middle” means two classes separated by space.
* ***FLOAT***
  + Since each div is block level, we can use ***float*** left which would consider no gravity means divs will be pushed up and then left.
  + Float also removes the whitespaces between images for suppose. These whitespaces are not through padding or margin. They are default.
* There should be only one h1 in the page. It should be the biggest thing. Otherwise use h2 or its descendants.
* This command sets width as 700px when possible otherwise 80%.

max-width: 700px;

width: 80%;

* Different hr styles: <https://css-tricks.com/examples/hrs/>

**5.1. Bootstrap-3**

* <https://getbootstrap.com/docs/3.3/> for version 3.3.7.
* Helps make good looking *responsive* websites very quickly.
* Navbars (converts to hamburger), Grid System, forms, buttons, jumbotrons,…
* Bootstrap.min.css will load faster than bootstrap.css as it is compressed but both are functionally same.
* Bootstrap.css is more readable than bootstrap.min.css.
* Bootstrap CDN: Link to hosted version of same bootstrap file we downloaded.
* We can always override the bootstrap styles. So, add custom styles link after bootstrap link.
* Container class gives nice padding and margin in bootstrap. We can have separate elements with class container, like inside navbar and outside jumbotron.
* Don’t but navbar inside container. Only its contents.
* Bootstrap’s JavaScript requires jQuery. Hence jQuery link before bootstrap’s JavaScript CDN.
* Grid System is number one reason why people use Bootstrap!!!
  + Every container in bootstrap can be divided into 12 different columns. Then we can decide how many of 12 units each element can take up.
  + As we resize browser, the units each element takes adjusts.
  + We can nest grids inside grids to further divide it into 12 units.
  + 4 layouts (sizes) available: xs (mobile), sm (tablet), md, lg (Details: <https://getbootstrap.com/docs/3.3/css/#grid> ).
  + If you want each element take 1 unit when large size, and 2 units when medium size:

<div class="col-lg-1 col-md-2">Responsive</div>

* If we set 2 for sm, it will automatically set 2 for md, lg as well. Similarly, for md, it will set up same value for lg. But we can always explicitly set values for larger sizes.
* Nesting syntax:

<div class="col-md-3 col-sm-6 col-xsm-12 pink">

<div class="row">

<div class="col-lg-6 orange">First Half</div>

<div class="col-lg-6 orange">Other Half</div>

</div>

</div>

* **NOTES varying size images**:
  + If you don't mind the images being of varying heights, but want to get rid of the white space bug then see [here](https://www.udemy.com/the-web-developer-bootcamp/learn/v4/questions/1989586).
  + If you're looking for Pinterest-like presentation of thumbnails of varying heights and/or widths, you'll need to use a third-party plugin such as [Masonry](https://masonry.desandro.com/), [Isotope](https://isotope.metafizzy.co/), or [Salvattore](https://salvattore.js.org/).
  + If you want to crop all of the images to the same size then see an example of this [here](https://codepen.io/nax3t/pen/MJwpdb).
* [www.unsplash.com](www.unsplash.com%20) provides a bunch of high-resolution pictures.
* Thumbnail class in bootstrap when our image is too large for a column so it would scale it down. Also, it gives a nice border to our image.
* When navbar is fixed to top, we will have to have the padding for body. Basically, the goal is to push the contents of the body down so it doesn't fall behind the fixed navbar, and we use padding to achieve that.
* Put bootstrap.css inside the css folder and css folder adjacent to font folder and html file so that fonts load successfully on html page.
* Fonts: Bootstrap Glyphicon or <https://fontawesome.com/icons?d=gallery&m=free>

**5.2 Bootstrap-4**

* Summary under migration tab in the documentation will detail all the changes.
* Bootstrap 4 supports flexbox! So no more floats!
* Global font-size changed to 16px (used for rem).
* 5 screen sizes available. Extra large is the new one.
* Stable release means don’t expect any breaking changes so you won’t have to rewrite much code. Alpha release means very buggy and liable to change. Beta means a little more polished.
* Bootstrap 4 Typography: Change font-size of root in html. As Bootstrap 4 is based on rem unit of measurement.
* Bootstrap Utilities- Spacing (margin and padding), borders and a lot more! $spacer is 1 rem by default. Can be changed.
* Responsive breakpoints!
  + {property}{sides}-{breakpoint}-{size} for sm, md, lg, xl.
  + {property}{sides}-{size} for xs.
* p-md-5 means apply padding 5 for size medium and **up**. Similar for other sizes.
* Here, padding left 5 at sm size will make padding left 5 for md, lg, xl also. Padding top 5 at md will make padding top 5 for lg and xl also. And so on.. Hence at xl size, we have padding of 5 on all sides.

<button class="btn btn-success p-0 pl-sm-5 pt-md-5 pr-lg-5 pb-xl-5">Button!</button>

* Navbars: We can change when the navbar should collapse and we can can background colors easily using style.
* Display Utility!! We can set an element as block-level, inline, hide and so on.
  + .d-{value} for xs.
  + .d-{breakpoint}-{value} for sm, md, lg, xl.
* **Flexbox!!**
  + Flex direction or main axis: Left to right. (.justify-content-start is default)
  + Cross axis: Top to bottom. (.align-items-stretch is default so they take stretch to the height of longest)
  + These are also responsive. Example, .justify-content-lg-end.
  + We can make left to right as right to left for main axis. As similarly change for cross axis.
  + We can make cross axis as main axis by .flex-column.
  + We can also move each item inside a flex individually by overwriting the default or class common to all items. For example .align-self-start.
  + With nav class, we have the flexbox enabled.
  + **Grid** is also built with flexbox. If not specified the elements will take equal space in 12 or remaining units.
* Pattern Project-
* **Section element** instead of div and **lead class** inside paragraph to make the text stand out and give extra space.
* Use anchor tag with btn class instead of button when not a form as more suitable.
* Background image centered horizontally and vertically, cover and no-repeat. The forward slash is required syntax for separating background position values and background size values. This is to eliminate misinterpretation of the values specified. If you're going to specify both, then background-position has to come first, then the forward slash, then the background-size.

    background: url("imgs/header.jpeg") center center / cover no-repeat;

* btn-outline-danger also changes color of text and font inside it to danger color.
* Museum of Candy Project:
  + Aria-label to make the hamburger meaningful. Nothing changes functionally.
  + In Bootstrap **images are made responsive** with .img-fluid which makes max-width: 100%, height: auto so that the image scales with the parent element.
  + Media query to selectively apply styles based on different screen size.
  + Change order of contents inside the flexbox based on screen size using Bootstrap!

1. **JavaScript**

**6.1 Introduction**

* <https://medium.com/@_jmoller/javascript-conventions-and-best-practices-9e980751c70f>
* Our code should be DRY (Don’t Repeat Yourself) not WET.
* JavaScript Developer Console in Chrome to write JS in our browser.
* The <script> tag can be placed in the <head> section of your HTML, in the <body> section, or after the </body> close tag, depending on when you want the JavaScript to load.
* < in the console means it’s returned from a function. For example, try console.log(4) or call any other function. < undefined in the output means nothing is returned.
* 5 primitive data types-
  + Numbers (whole, fraction, negative numbers)
  + Strings (“hello”, “43”)
  + Booleans (true, false)
  + Null (This is a value)
    - Means explicitly nothing. Like suppose a player dies, make it null.
  + Undefined (This is a value)
    - Declaring a variable but not assigning any value to it.
* Strings
  + In strings, double and single quotes are both acceptable as long as they are matched.
  + “He’s” is valid and ‘He’s’ is not. Similarly, ‘“HEY!”’ is valid and ““HEY!”” is not.
  + String operations-
    - Concatenation (+)
    - Escape special characters with \.
      * “Singin \”Do with diddy\”” is valid due to backslash.
      * “\\” outputs “\”.
      * Length using string.length.
      * Access a character using string[index]. (Zero-based indexing.)
* Variables
  + **camelCase**
  + var variableName = valueToBeStored;
    - var name = “Rusty”;
    - var isAdorable = True;
    - var age = 2;
  + Change variable value or variable type (change variable type coz of dynamic typing) just by reassigning
    - var num = 10;
    - num = 12
    - num = “Now a string!”
  + 3 ways of making a variable: const, var, let.
  + **Let > const > var (preference,** <https://medium.com/javascript-scene/javascript-es6-var-let-or-const-ba58b8dcde75>**)**
  + Var
    - Scoped to variable’s enclosing function or the global scope if not inside a function.
    - Can be reassigned whenever.
    - Initializing with value is optional.
    - Can be redeclared at any time.
    - Global variables are added to window.
  + Const
    - Block scoped.
    - It stands for constant and hence can’t be reassigned.
      * It’s not immutable. But the variable reference can’t change.
        + const people = [“Saanika”, “Gazal”]
        + people.push(“Gargi”) ✅ (Correct as reference isn’t updated)
        + people = 12 ❌(Can’t reassign)
    - Must be initialized with value.
    - Cannot be redeclared in same scope.
    - Does not create property on global window object.
  + Let
    - Block scoped.
    - Can be reassigned.
    - Initializing with value is optional.
    - Cannot be redeclared in same scope.
    - Does not create property on global window object.
* Built-in methods
  + alert(“Hello there!!”); or alert(192\*2);
  + var age = prompt(“What is your age?”); (Stores input as string)
    - typeof age is “string”.
    - Number(age) to convert it to int.
  + console.log(“Hello from console!”); (It’s only for our own purpose)
  + “Hello World”.indexOf(“World”);
  + “Hello”.slice(1);
  + clear()
* Alert, prompt functions doesn’t let HTML be displayed unless closed. Hence we can wrap the JS code in setTimeout() to let HTML load first. But anyway, alert and prompt are never really used and definitely not on page load.
* Object is a class in JS, so here: const dogs = {name: "Rusty", breed: "Mutt} , dogs is an instance of the Object class.
  + const dogs = {...} is an object literal.
  + The full syntax is: const dogs = new Object({...})

**6.2 Control Flow**

* Comparison Operator
  + >, >=, <, <=, ==, !=, ===, !==
  + Double == performs type coercion
    - == (5 == “55” is true)
    - !=
  + ===
  + !=== (5 !== ‘5’ as different type though same value)
  + Always use triple = as it’s more specific.
  + Examples
  + null == undefined is true.
  + null === undefined is false.
  + true == “1” is true.
  + 0 == false is true.
  + NaN == NaN is false (weird ha?! :P).
* Logical Operators
  + && (AND)
  + || (OR)
  + ! (NOT)
  + Every value in JS is truthy or falsy like NaN, 0, null, -1. Check it like !!NaN.
    - Falsy values
      * False
      * 0
      * “” (empty string)
      * Null
      * Undefined
      * NaN
    - Everything else is truthy. Example, “haha!”, “false” as these are not empty strings.
* Conditionals: if, else if, else.

**6.3 Functions**

* If we don’t pass a parameter to function, JS will consider it as undefined instead of throwing an error message.
* Return keyword stops the execution of a function.
* Two syntaxes for declaring a function
  + Function declaration
  + Function expression (if we reassign the variable, the function will get lost).
* We can declare functions anywhere coz JS hoisting will hoist it at top. But declaring with function expression won’t be that flexible.
* Functions written in the JS file linked to html file are accessible from the console.
* Scope means context (variables and properties) in which is the code is executed.
  + When we define something outside a function for example, global (parent scope), we would still have access to it inside a function (child scope).
  + If we reassign the global variable inside a function, the global variable’s value is changed as well.
  + If we redeclare the global variable inside a function, the global variable doesn’t change.
  + Every function has its own scope and they are not share between functions unless nested.
* Higher Order Functions: Calls or returns a function.
  + setInterval(anotherFuncName, intervalInms); (Stop by using clearInterval(num))
  + Can also directly declare the function in parameters instead of passing function name.
    - We can’t call this function from outside.
  1. **Arrays**
* <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>
* Zero-based indexing.
* We can reassign values at an index in array.
* If we assign value to an index 100 suppose (like appending), but last index in array was 6, then index 7 to 99 will be stored as undefined, and index 100 as that value assigned.
* We get undefined not an error if we access an index which wasn’t assigned.
* Initializing an empty array
  + var friends = []; // empty array
  + var friends = new Array(); // uncommon
  + var random\_collection = [49, true, null, “Saanika”]; // can hold any type of data
* Arrays have length property.
* Built-in methods
  + Add and remove from end of array
    - push (returns size of array)
    - pop (returns last element)
  + Add and remove from front of array
    - unshift (returns size of arrays)
    - shift (returns first element)
  + indexOf (return first instance of an item in case of multiple)
  + slice (to copy parts of an array or the entire array)
  + splice (to delete a a part of array)
* Iterating an array
  + forEach() // much easier and very common.
    - arr.forEach(funcName);
      * Calls a callback function (function passed into another function as an argument).
      * Callback function is expected to have 3 arguments in the following order
        + Array element.
        + Index of that array element.
        + Array that .forEach was called on.
    - Can also directly declare the function in parameters instead of passing function name.
      * We can’t call this function from outside.
    - Arguments passed in the function will be value of an array item.
      * Won’t throw error if function isn’t expecting arguments.
  + use for or while loop.
* Building our own forEach
  + If you want to create a function and execute it immediately after declaration just once, you can use anonymous function. Like (alert()) is an anonymous function that calls alert();. Another example-

(function (){

    console.log('Immediately invoked function execution');

})();

* + To immediately call a function, you need to either save it to a variable first or create an IIFE by wrapping it inside () which is an anonymous function that can be called immediately. (Examples: [https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861536#questions/12544910](https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861536%23questions/12544910))
  + This code below will define and invoke the function. We would have to wrap the function around () otherwise would throw error.

(function hello(){

  console.log("hiiiii");

})()

* + Function name is equal to function definition for example hello is equal to

function hello(){

  console.log("hiiiii");

}

**6.5 Objects**

* Store data in key-value pair.
* Retrieve data using
  + Bracket notation like arrays
  + Dot notation // I prefer this
    - We can’t use dot notation if property starts with a number or has a space in it.
    - In below example, str isn’t evaluated to “name”. Hence we can’t look-up an object property variable using dot notation (but we can using bracket notation).

var str = "name";

someObject.str;

* Update data after retrieval
* Methods of initializing objects
  + Make an empty object and then add to it using dot or bracket notation.
  + All at once (object literal notation).
  + Use new Object() which would return an empty object. // very rare
* Adding methods to objects helps in avoiding namespace collision.
  1. **DOM Manipulation**
* Application
  + Games
  + Scrolling effects
  + Dropdown menus
  + Form validations
  + Interactivity
  + Animations
  + Every awesome site ever!!
* Document Object Model (DOM) is the interface between JavaScript and HTML+CSS.
* Basically, Object Model means JavaScript objects representing/modelling HTML elements.
* Document object is the root node (top-level object) where everything resides.
  + console.dir(document); prints the entire document object (Browser hides the object representation from us if we only type document in the console.).
* We can access body object by document.body and so on for other children.
* The browser converts every HTML element into JavaScript objects (each containing properties like color, background color, fonts and so on) that we can manipulate.
* Process
  1. Select an element (returns the object)
     + document.getElementById();
     + document.getElementsByClassName(); // Returns HTMLCollection
     + document.getElementsbyTagName(); // Returns HTMLCollection
     + Query Selector // Newer method. Can do everything the first 3 does
       - document.querySelector(); // Better than ways that return HTMLCollection as for a single object also, they return HTMLCollection
       - document.querySelectorAll(); // Returns HTMLCollection
  2. Manipulate it.
* Methods like getElementsByClassName() or getElementsByTagName() return an HTMLCollection object which is an array-like list of objects representing HTML elements.
* In HTMLCollection object-
  1. We can access by index.
  2. We can use length property.
  3. We can’t use forEach().
* JS object properties and methods
  1. Through style property we can access all the styling properties we could define through CSS (Put the property values inside “” unlike in CSS).
  2. We can see the list of classes of an object suppose p by p.classList (It isn’t an array. We have methods like add, remove and toggle here.).
  3. Text content
     + Using *textContent*, we get the text contained in that element. (Doesn’t include tags nested inside that element. Like <p><strong>Hi!</strong> I’m Saanika</p> would only return Hi! I’m Saanika).
       - We can’t put html tags like <h1>…</h1> when overwriting as it would be treated as string and not HTML.
     + *innerHTML* will also return tags nested inside the element also with the text contained.
       - Putting html tags like <h1>…</h1> when overwriting will actually render h1 as innerHTML treats it as HTML and not text unlike textContent.
     + Changing the content by accessing it through the above properties, we would not be able to preserve HTML elements already applied on that text, like <strong>…</strong>.
  4. Use getAtribute() and setAttribute() to read and write attributes like src of image or href of a .
* **Separations of Concerns**
  + HTML responsible for structure.
  + CSS responsible for presentation.
  + JS responsible for behaviour.
  + We want no/minimum overlap between their responsibilities. Hence, we have a
    - A separate file for styling so that HTML and CSS are separate.
    - Using objectVariable.style.\* to manipulate a bunch of styles on the same element is not a DRY code. It’s better we define a CSS class and then using JS, we can add that class to the selected element in one line. This way we can turn on and off those styles instead of manipulating styles using JS.
  1. **Advanced DOM Manipulation**
* <https://developer.mozilla.org/en-US/docs/Web/Events>: addEventListener() // Run a function when a particular event happens for example click.
* We can add multiple event listeners to same object.
* We have to make sure that our HTML loads before JS because JS works on elements of HTML. (Like an event listener on button is loaded when the JS is loaded and hence it expects the button to be already loaded. Later when the button is clicked, the callback function runs.).
  + Move the script tag to the end of the body.
  + Doesn’t work in case of functions like alert(), prompt(), confirm().
* Color Game Project
  + Using CSS float on squares removed them from the normal flow of the page, and that's why they appear like they are not contained in the body element when you inspect it. (See here: [https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861536#questions/12566430](https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861536%23questions/12566430))
  + When using font-size: inherit; it will look up the HTML tree looking for an element with a declared font-size. If there isn't any, as in this case, it will resort to use the body’s default font-size, which is 16px (1rem).
  + Styling transitions might not work in all browsers. So, add support for them.

1. **jQuery**

* <https://jquery.com/>
* <https://api.jquery.com/>
* JS library.
* Why use it?
  + Helps manipulate DOM.
  + Event handling, animation, Ajax (make HTTP requests) can be done easily through APIs.
  + Makes code shorter and clearer.
  + Easy to use.
  + Provides cross-browser support.
* But…
  + It is a heavy tool.
  + It doesn’t do anything you can’t do using JS.
  + Performance (If you are not experienced, you might write inefficient code)
  + <http://youmightnotneedjquery.com/>.
* We can download and link the file or use CDN version (hosted copy).
* jQuery linked inside head and JS linked at the end of the body because *jQuery should load before our JS script as it’s a dependency*.
* Compressed (no whitespaces and names of properties and variables are shortened) will be faster to load.
* CDNs will be a little slow.
* Better to put the jQuery files in a folder (say lib) so that you don’t clutter up.
* After linking, type jQuery in console. If you get no error, means file is correctly linked.
* Select elements with $() ($ sign is a function).
* Getter
* .css(“property”) retrieves that style property of selected element.
* .attr(“attribute”) retrieves that attribute of selected element.
* Setter
  + Use .css(property, value) to style the elements. (Better avoid this as this is the work of CSS not JS!)
    - If we want to alter multiple properties, make an object with all the property-value pairs and pass the object in .css().
    - $(“li”).css(“color”, “blue”); would make all the lis blue. No for loop required unlike in JS.
  + Use .attr(“attribute”, value) to change attribute value of selected element.
* Events (these are sufficient!)
  + click()
  + keypress()
    - <https://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes>
  + on() // Most used jQuery method!! It’s like .addEventListener() of Vanilla JS.
    - *Why use on(“click”) over click()?*
      * click() only adds listeners for existing elements and on() will add listeners for all potential future elements (Delegated event handlers have the advantage that they can process events from descendant elements that are added to the document at a later time). See the use case in Todo List Project!
* Effects
  + .fadeOut();
    - It only hides or sets display to none for the selected element. Doesn’t remove or delete them. To remove we can use .remove().
    - If we set 1000 ms as the time to fade-out, the code written after it would not wait 1000 ms (as it’s time taken to fade-out, not an interval). If we want that code to run after 1000 ms, we can wrap that code inside the fadeOut function.
* In JS we can’t have property names with hyphens. Then we would have to convert property names with hyphens to camelCase.
  + But! There’s a hack! Check this thread for examples: <https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/10152062#questions/12006534>
    - In JS
      * l.style["font-family"] = "Arial";
      * l.style["fontFamily"] = "Arial";
    - In jQuery
      * $("div").css("background-color", "purple");
      * $("div").css("backgroundColor", "purple");
    - So, both ways possible when inside “”. But in pure JS, hyphen is strictly prohibited, for example,
      * h1.style.textDecoration = "line-through";
* Event bubbling (Anything clicked on li would also trigger ul and it would trigger container of ul, then body, then html! Basically it’s bubbling up!!).
  + Solution: .stopPropogation(); in jQuery.
* Project Todo List:
  + .parent() to select the parent object. This can be helpful in removing a todo when clicked on a cross sign i.e. child of todo.
  + Interesting! *JS supports method chaining!!!* The below code is valid because each of them returns an object. And functions are triggered from left to right. To confirm, type in console and see what is returned! Again, .remove() won’t wait for the fade-out to finish! So, use a callback function instead!

$(this).parent().fadeOut().remove();

* + Using .on() instead of .click(), will make sure that new todos also have the listeners added automatically.
    - Use delegated event handlers. Attach the event handler to ul. The event would bubble up from li to ul.
* Patatap Project
  + let and const won’t work in paper.js.
  + Paper.js scripts run in their own scope, hence their variables and so can’t be accessed from browser console.
  + Used HTML canvas is used for drawing circle.
  + Canvas’s height and width change when the visible area of the canvas changes (Coz we changed the browser size).
  + Cross Origin Errors in Console: To get rid of that, use live server in VS Code or run a local HTTP server. Reason: Chrome allows cross-origin requests (run sound files) only if the HTML file is run as a server on the appropriate protocol. [https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861574#questions/12597796](https://www.udemy.com/course/the-web-developer-bootcamp/learn/lecture/3861574%23questions/12597796)