**CSS Box Model**:

Everything in CSS is treated as a Box!

*width*: decides width of the content

*height*: height of the content

**Border Properties**:

*border-width:* controls the thickness of the border

*border-color:* controls the color of the border

*border-style:* controls the the line style: dashed, dotted, solid etc

**Padding**: Space between **content** and **border**

**Margin**: Margin is the space between the **boxed** **elements**

**Display Property:**

It changes in-line elements to block and vice-versa. Most of the elements by default take either in-line or block. But we can change it using *display*

***inline-block***

**CSS Units**:

CSS Units are of two types: *Relative* and *Absolute*

***em’s and rem’s(root em’s)***: difference??? (TBD)

ems change according/with respect to their parent elements.

**Other CSS Elements**:

**Opacity & Alpha Channel**

Alpha Channel ranges from 0 (completely transparent) to 1 (not at all transparent)

Opacity also works on see-through-ness of the content, BUT even the text content changes as per Opacity given. Wherein Alpha channel works purely on background but not on texts.

**Position (?!?)**

* *Static*: nothing changes
* *Relative*
* *Absolute*
* *Fixed*

**Transitions**

Transitions help in changing something visual on the go based on the given parameters.

Ex: When you hover on a box, it should change its shape and color

We can customize the following:

* Property Name
* Duration
* Timing Function
* Delay

**Transform**

It basically decides the movement of the content within the page

**Background**: It decides everything about the content background be it colored background or an image background by taking image from local or web. We can also crop/adjust the background image

**Google Fonts**:

You can use google fonts for free from *fonts.google.com*

**Note**: When adding images, DO NOT add them line by line. Instead, add them like Web Page for Photos.html.

**CSS Flexbox**

CSS Flexbox is all about distributing space to the containers of the data. When we minimize the webpage, the data gets shrunk and it no longer takes their original space.

For ex, if a container has 300px, it will shrink to less than 300px when it is minimized and cannot hold the same 300px size.

**Flexbox**:

* **Main Axis**: From left to right
* **Cross Axis**: from top to bottom

There are as many as 10+ properties for Flexbox which defines the containers

Justify-Content works on Main Axis

Align-Items work on Cross Axis

**Responsive Design**

It is important that the websites are created by keeping all sorts of screen sizes in mind, as now a days, people use mobile screens more than an actual computer screen

So, it’s important that the design should work for all screen types.

**Media Queries:**

Media Queries allow us to modify our styles depending on particular parameters like screen width or device type

**Bootstrap** (library for CSS­):

Bootstrap is the world’s famous CSS framework

**Framework** is nothing but a readymade important chunk of code, that can be incorporated wherever we need it in our project code.

Bootstrap -> *Components* and *Grid System*

CDN helps us to access code remotely so that we need not to download and include in the code.

**NOTE**: To use Grid System, you need to have ***at least*** one container class in the code.

**JAVASCRIPT**

Primitive Types (the basic building blocks):

Javascript has ONLY ONE number type

* **Number**: Positive numbers, negative numbers, whole numbers (integers), decimal numbers
* String
* Boolean
* Null
* Undefined
* Symbol
* BigInt

Read, Evaluate, Print, Loop (R E P L)

JS Console is the interactive section

**PEMDAS** (parenthesis, exponential, multiplication, division, addition and substraction) is the way how NUMBERS work in Javascript

**NaN**: Not a Number, it still falls under the typeof() a Number

Any mathematical operation involving NaN results in NaN again!

**Variable** is a container that stores value and “let *VariableName* = Value;” is the syntax for a Variable in Javascript

let Score = 50;

**Score = Score +10;**

OR

**Score += 10;**

Score = 60;

Score++ increments the value by 1 whereas Score-- decrements the value by 1

**Const** works just like the variable, but you **cannot** change the value stored in it.

“Const Score = 50;”

Score stored here will always be 50 and cannot be changed

**Var** is the only way back then to declare variables **before** let and const

**Booleans (true or false in lower case)**

You **can change** the types of the variables ex: Number to Boolean or vice versa

**Variable Naming and Conventions**

* You cannot have a variable name with a number at the beginning: “1name”
* You cannot give spaces in between for a variable name
* You can use CamelCase for new words in the variable name **but** first word remains small case: “camelCaseExample”, “isGameOver”

**String**: String is a textual information stored. And should be wrapped up inside “quotes” can be double or single quote but not a combination of both in a single string!

Strings are indexed and **Indexes start from 0**. animal = “Pig” has indexes of animal[0], animal[1] and animal[2].

Strings can be concatenated. “Pig” + “Dog” = “PigDog”

**String Methods**

There are various String Methods which are typically actions to be performed to the string stored. isLowerCase, trim etc are some.

let name = “CHAITU”

let newname = name.isLowerCase().trim()

Adding ‘()’ defines it as a method and it needs to be added in order to execute the action to be performed on string.

String Methods also accepts **arguments inside the ()** which makes them String Methods with Arguments.

**Null** defines an empty space. It is nothingness.

LoggedInUser = null;

*Once user logs in*

LoggedInUser = Chaitu Subarashii;

Note: **==** compares the value but not the type wherein **===** is very strict and compares both values and its types.

Ex:

1 == ‘1’ => true

1 === ‘1’ => false

**console.log():** Prints arguments to the console

**Ctrl/Cmd + /** comments out the code

**[Arrays] in JavaScript**

*[Arrays] allow to group the data together, more like ordered collection of values like songs in a playlist, list of comments on IG Post etc*

**Indexing** is a useful function of Array wherein we can call the stored data based on the Index of that element/value inside of an Array

**Array Methods**:

**Push**: Add to the end

**Pop**: Remove from end

**Shift**: Remove from start

**Unshift**: Add to Start

**Slice**: Slice as the name suggests, slices/cuts down the Array based on *Start* and *End* and declaring End is optional. By default it will go from Start to the End of the Array.

Array takes **different** memory space each time you compare two contents in Arrays.

That way [‘chaitu’] === [‘chaitu’] results to be **FALSE**

But if you make **Array1** === [‘chaitu’] and **Array2** === [‘chaitu’].

Then that way Array1 === Array2 results to be **TRUE**

**Object Literals**

Objects are collection of properties, and 2nd data structure in JS after Arrays

It uses **Key-Value pair**, instead of Indexing, we access information using Key

**Property** = Key + Value

Keys turn into a **String** when storing information

**Arrays** and **Objects** allow us to store large piece of web information

**Loops**:

Loops allow us to perform repeated tasks. Ex: *Print “Hello” 100 times*

**for (**

**[initialExpression];**

**[condition];**

**[incrementExpression]**

**)**

**Functions**

Functions are reusable chunk of code that can be called at any point in the code

*Define the Function* -> *Run/Call the Function*

**Arguments** are inputs to the functions!

We can also write functions that accept inputs, called **Arguments**

function greet(firstName){

console.log(`first name is: ${firstName}`)

}

**Return:**

Return returns the value that is executed but also holds the value to save and it also stops the code written after it.

**Scope**

It is variable visibility, a location where a variable is defined dictates where we have access to that variable

**IMP**: We cannot call a variable outside of a function as it is **ALWAYS** limited within the function, also known as *Function Scope*

Same applies to a Block such as Conditionals or Loops, called as *Block Scope*

However, a child function inside a parent function can access the variables from its parent and also from its grandparent (topmost function and lower most function), called as *Lexical Scope*

**Document Object Model (DOM)**

Font-size in CSS changes to fontSize when coming to JS. Attributes will be renamed to camelCase

**Stack** is last thing in and first thing out; imagine it like keeping clothes in a box wherein the last placed dress is the one we can remove it at **first**.

**try** {

If this is an error;

} **catch(e)** {

print this, avoid the error message

}

**AJAX:** Asynchronous Javascript and XML

**API**: Application Programming Interface, which connects or helps to communicate between softwares or interfaces

**XML**: Extensible Markup Language

**JSON**: Java Script Object Notation

**AJAJ**: Asynchronous Javascript and JSON

Web APIs respond with data in JSON, we can then parse that data into JS object

XML Http Requests (XHR) does not support **Promises**, which makes the code junky

But Fetch is the best alternate method which will improve code readability and supports Promises.

**Object Prototypes**:

All the objects will have predefined built-in functions, which can be seen in their prototypes.

But we can also manually create a new method using function, which will be added to their prototypes.

**Why using Terminal is better than GUI**:

* Faster processing of given commands
* Access to entire system
* Access/View hidden files

**Node JS**

Node is the implementation of the JavaScript that runs outside of the browser (as a ***native*** application)

REPL (Read, Evaluate, Print, Loop)

NPM (Node Package Manager)

We can install all the required packages into Node using NPM

Packages can either be installed *locally* or *globally*

**Express** is a framework for web development creating servers using Node JS

Both **Libraries** and **Frameworks** use an imported piece of code.

But ***Libraries*** mostly assist in smaller functionalities in code, and we can decide or control the usage in terms of where to use and how to use

Wherein ***Frameworks*** come with lot of functionalities, and we cannot decide things on our own but to follow as per designed Framework

Using Express, we can create a response, but request object is used by Express itself, which means we need not to define request

***Routing*** is the process of matching the incoming request with an answer as a response

In real world, we cannot write code for a response to each request, so we use ***patterns***

If a request is with google.com/search => we give a response

If a request is with google.com/search/q/query => we give another response

**Nodemon** is a packagein Node JS that restarts the server automatically whenever it detects any chanes in the code

Dynamic HTML is achieved by ***templating***

Templating allows us to define a preset **Pattern** for a webpage, that we can dynamically modify

**EJS (Embedded JavaScript)** allows us to include JS functionality inside an HTML file, by making a .ejs file

**REST** (Representational State Transfer) is a guide for how a client and server should communicate

**CRUD Functionality (**Create, Read, Update and Delete**)**

**Databases** are used to store data efficiently and helps us to retrieve or update the data and offers good security and almost are scalable.

Databases are primarily classified into *SQL Databases* and *NOSQL Databases*.

SQL Databases are relational databases which will use SQL (Structured Query Language) and data is stored in a relational manner and more of a table form.

NOSQL Databases do not use SQL and data is stored in a more diversified manner like key value pairs or document format.

**Object is a collection of properties, where each property has a *key* and a *value***

Its more like an unordered collection of named values.

**SERVERS**:

Servers are centralized machines that are looped with multiple computers to perform high efficiency tasks.

Servers will have a different set of processors, different set of RAM and works on different Server-Specific OS.

Servers are designed to handle multiple requests from client side and memory is very well organised.

Main Types of Servers include:

**Web Server**: which provides the data of the website to the frontend when accessing that website

**Mail Server**: which provides send and receive functionality of the mails from an email domain

**Database Server**: which storages huge amount of data in the backend and can be accessible from frontend when requested

**For more:** <https://www.youtube.com/watch?v=UjCDWCeHCzY>

**CLOUD STORAGE AND CLOUD COMPUTING**

Cloud Storage or Cloud Computing practically refers to huge data centers that are designed to maintain and provide various services.

Imagine, you need to store about 200 TB data into a storage like a Hard Disk. In this case you need to invest in buying a Hard Drive of 200 TB and you need to maintain it if there’s any issue.

Instead, you can simply upload the data of 200 TB into a cloud (typically to a Data Center), so that the Cloud will store and maintain the data for you and you pay for this service.

All the Cloud Storage Platforms work on the same principle wherein they store your data in their Data Centers and you would need to pay for it to use.

Netflix uses AWS to store and maintain its data base.

Same goes with **Cloud Computing,** wherein instead of investing in the software infrastructure like RAM, GPU etc needed for you, you would remotely use a well build computer from a Data Center.

If you need to perform extremely high-end tasks, you need not to invest in buying the configuration like high RAM and all. Instead, you can use Cloud Computing Services like Amazon AWS or Microsoft Azure.

Well, what’s the use of opting Cloud Storage or Cloud Computing?

It saves lot of money and efforts. You **need not** to invest on buying anything on your own, instead can use other’s (Data Center) high end computer or Storage.

Your data will be maintained by Data Centers and you’ll be provided with high-end machines to complete your tasks.

For more: <https://www.youtube.com/watch?v=_a6us8kaq0g>

WATERFALL MODEL FOR SOFTWARE DEVELOPMENT:

In Waterfall model, the end product or the software would take significant amount of time, as each and every functionality needs to be completed and each team would work individually with minimum or no sync. If a software takes 2 complete years to develop, it is **impossible** to implement new changes after 2 years. In short, we cannot add any new features!

To overcome all these drawbacks, organisations started implementing the new model called AGILE.

**Agile Methodology**: Iterative and Incremental Approach

Agile Methodology is focussed on Customer Satisfaction and each piece of software is being delivered to the client or end user at regular intervals (*continuous delivery*).

**Product Backlog** is a collection of requirements, collected by a *Product Owner* fromthe client/customer.

Product Owner then decides the **Priority** of the *User Stories* that are created in the Product Backlog based on requirements.

A **Spring** usually lasts anywhere between 1-4 weeks, with 1 week being minimum and 4 weeks being maximum.

**Spring Planning Meeting** will be conducted by Product Owner, Scrum Master, Dev Team and Testing Team. They will discuss on how many stories that we can commit to deliver by the end of Sprint.

Stories that are chosen to deliver by the end of the Sprint will be moved to **Spring Backlog**

**Product Backlog:** A collection of all the features/requirements to be delivered

**Spring Backlog:** Chosen features that are to be delivered in the current Sprint

Once Sprint is started, Scrum Master will conduct stand up calls every 24 hrs to check the status of the ongoing Sprint

When Sprint is nearing to an end, a **Sprint Review** is conducted by Product Owner, to select/reject the worked upon user stories from that Sprint.

Once the Sprint is completed: a **Sprint Retrospective Meeting** is being conducted to look back and analyse the limitations faced in the last sprint and to deliver improved work for the upcoming Sprints

For more: <https://www.youtube.com/watch?v=uZP7pNkNylA>

Web Servers sends back the information as a *response* in 2 formats:

**HTTP**: Hyper Text Transfer Protocol (not secured, hence easily traceable)

**HTTPS**: Hyper Text Transfer Protocol Secured (completely secured)

**API** (Application Programming Interface)

Used to help two interfaces to communicate with each other, without the need of providing any real data

ex: Third Party flight booking apps communicating with Airlines

Receiver API verifies the Sender API by **API Key**,howeverAPI Key is not mandatory and can also be ignored which is a rare case.

API is the mediator who helps us to communicate with the actual system

Vanilla JS is the by default JS

Angular Framework is used by Google, PayPal, MS Office, Netflix, Gmail wherein React Framework is used by Facebook.

**Angular**: TypeScript is mandatory for Angular; No need to additionally install any add on like Routers;

Angular JS came first and then followed by Angular 2, which was rewritten with lot of improvements since then.

**React**: Regular JS (Vanilla JS) is enough; Minimal Framework and components need to be installed manually;

**GIT**:

GIT is a version control system that monitors version history. It allows multiple users to change/work on a file and updates its version for each change.

As it tracks all the changes that ever happened to a document, it helps you to undo the previous change. It does by making **a new change**, which is exactly opposite to the change that we want to undo.

It also answers the questions like: who made changes or why changes were made.

**Version Control System**:

* Keeps track on project history
* Allows you to revert files to its previous state
* A VCS also provide a collaboration environment
* Can easily recover

**Local Version Control**:

Local Version Control has only one DataBase that keeps all the changes in a project.

**Central Version Control**:

* It helps to collaborate with other developers
* It works on a single server

**Distributed Version Control**:

* They fully clone the repository
* You can collaborate with many different groups in different ways
* Bring a local repository up to date with a remote repository

**About GIT**:

* Initial version came out in April 2005
* Exceptional Branching system for non-linear development
* Incredibly fast
* Very efficient with large projects
* GIT is open source
* Easy and safe to save and share files
* GIT need not to connect to a central server. It performs all the operations locally on the user’s machine.

**Synchronous vs Asynchronous JavaScript**

JavaScript is always synchronous and *single-threaded*.

**JavaScript Engine** in the browsers execute the JS code.

**Synchronous Code**:

console.log(“before”);

console.log(“JavaScript is Awesome!!!”);

console.log(“after”);

Output: In the above code, “*before*” is printed **first** followed by “*JavaScript is Awesome!!!*” and “*after*”.

A thread is assigned to one task and starts working on it. Once the task is completed, it is available for the next task. In this model, it **cannot** leave executing a task in the middle and take up another task.

If we have couple of tasks or lines of code just like above example, current system provides just a single thread, then tasks are assigned to the thread **one by one**.

**Asynchronous Code**:

console.log(“before”);

setTimeout(function){

console.log(“JavaScript is awesome!!!)}, 2000);

console.log(“after”);

Output: Expected output would be in the same order, ”*before”*, “*JavaScript is awesome*” and “*after*”. But, “before” and “after” are executed first, then “*JavaScript is awesome!!!*” is executed following the given **setTimeout**.

In contrary to Synchronous model, in Asynchronous model a thread once starts executing a task can take a pause and store the task1 and move to another task and completes it i.e., it will jump to another task which can be performed while it waits for task1 to complete.

However, JavaScript is still a single threaded language. But how come Asynchronous Code is possible???

Because the code that we write will be executed in **Call Stack**, which is single threaded code. However, the code given in **Event Loop** and **Internal APIs** execute the Asynchronous Code.

**Callback Function**: A function which will ***just be passed***, but later called by JavaScript engine. For ex: we do not call or print a setTimeout function, but it will print the code post given condition of timeout.

**Event Loop** manages the asynchronous code and it pushes the code to the Call Stack.

Examples for Asynchronous Code: **setTimeout/setInterval**, **AJAX**, **Event Handlers** etc.

Video: <https://www.youtube.com/watch?v=Sv50j2sEnWY&t=256s>

**ANGULAR**

Angular is a JavaScript frontend Framework which is mainly used in making single page applications. It is very reliable and maintained by Google.

**Single Page Applications**: The applications, which need not to be reloaded for the changes to reflect. If you’re using gmail, gmail doesn’t reload on every click you make, but still reflect/display the requested changes.

Angular JS was first introduced, followed by Angular 2. Since then Angular released multiple updated versions with minor changes which usually doesn’t impact much to the work.

Applications that are build using Angular are easy to **test** compared to Vanilla JavaScript or JQuery.

Angular gives applications a clean or easy structure to understand and implement. It includes lot of reusable code. Applications made on Angular are more testable.

**Hot Module Replacement/Reloading (HMR)**

Whenever the source file is modified, ***webpack*** automatically refreshes the browser.

Angular CLI uses a tool called ***webpack*** which is a built automation tool. Webpack gets all our scripts and stylesheets, combines and puts them in a bundle and then minify the bundle, which is for optimization.

Whenever there is a change made in the code, Webpack recompiles and refreshes the bundles.

**Angular Libraries**: are distributed as separated packages via NPM (Node Package Manager)

* @angular/core: we use in every application by default
* @angular/complier:
* @angular/http: for calling http services
* @angular/router: for adding navigation to applications

**TypeScript**:

TypeScript is a superset of JavaScript. Any JavaScript code is also valid TypeScript code. TypeScript catches errors at ***Compile time*** instead of run time. When we compile our TypeScript code, we can catch these errors and fix them before it gets deployed into the application.

Browsers only understand JavaScript. So, TypeScript gets compiled or trans-piled into JavaScript.

**Variable Declarations in TypeScript**

ES5 (ECMAScript 5): Supported by all the browsers

ES6 (Introduced in 2015)

ES2016

ES2017

***Difference between Var and Let?***

Lambda function in C# is nothing but Arrow Function (=>) in JS

**Building Blocks of Angular Apps**:

**Component**: encapsulate the *data*, *HTML template* and *logic* for a view, which is an area of the screen that the user sees.

* Create a component
* Register in it a module
* Add element in an HTML Markup

**Modules**: Combination of related components

Components can be reused and can be maintained in smaller blocks

**Data Types in JS**

**Primitive Data Type**: Memory allocation in Stack, also called base data type

* ***Strings*** (collection of characters): “Chaitu”, ‘Subarashii’
* ***Numbers***: 3,4,5,6,7
* ***Boolean***: true false
* ***Null***: intentional empty value, used to check in later point of time whether it is truly a null, but displays as Object in *typeof*
* ***Undefined***:
* ***Symbol*** (recently introduced in ES6):

**Reference Data Type**: Memory allocation in Heap (Dynamic Memory), also called Objects (derived)

* ***Arrays***: [1,2,3,”Chaitu”,true,false]
* ***Object Literals***: {keys: values}
* ***Functions***
* ***Dates***

**Difference between *let*, *var*, and *const***: helps in creating variables and communicates to interpreter to create a variable

***Const*** and ***let*** improves code readability

***Const*** is *immutable*, however you can change its contents by using array functions such as push, **But** not reassignment!

ex:

*const* arr1 = [12,45,6,7,8]

arr1.push(45);

console.log(arr1);

but NOT this:

*const* arr1 = [12,45,6,7,8];

arr1 = [34,67,8,3];

**Var** is used in old JS

***let*** is of block level scope (more of local variable) whereas ***Var*** is of global scope (global variable)

`backticks` is great when creating variables with different types:

**`chai’I”ij,l`**

**Rules for creating JavaScript Variables**:

* Cannot start with numbers
* Can start with letter, numbers, \_ or $
* Are case sensitive

**Most common programming case types**:

* ***camelCase***
* ***kebab-case***
* ***snake\_case***
* ***PascalCase***

**Conditionals in JS:**

* **Simple *if***(Single Condition)
* ***if* – *else***(2 conditions; yes/no)
* ***else – if***(multiple conditions)
* **Switch** (multiple conditions)

‘*if*’ will check every statement whereas ‘*else if*’ will check ***only*** if previous statement is false.

**Operators inside Conditionals**:

**&&** - both the conditions should be *true*

**||** - any one condition must be *true*

**==** - equals

**===** - **strictly** equals

**Ternary Operator**:

condition ? expr*IfTrue* : expr*IfFalse*

const age = 26;

const beverage = age >= 21 ? "Beer" : "Juice";

console.log(beverage); // "Beer"

const age = 45;

console.log(age==45? “Age is 45”: “Age is not 45”);

**Switch**:

const age = 28;

switch (age) {

    case 18:

        console.log("your age is 18");

        break;

    case 28:

        console.log("your age is 28");

        break;

    case 38:

        console.log("your age is 38");

        break;

    default:

        console.log("invalid age");

        break;

}

It works more like else if and used for multiple conditions.

**AJAX**: (Asynchronous JavaScript and XML)

* It is a technique used to send and receive data asynchronously
* JSON is the latest and most popular data format

**XmlHttpRequest (XHR) Object**:

* API in form of an object
* Provided by browser
* Works with HTTP
* Also supports XML, JSON, HTML and Plain Text

AJAX is capable of reloading the webpage dynamically by reloading only a part of webpage that needs to be updated (Partial Reloading).

If we are writing any web operations that takes more time to process, we should not write code in synchronous model, it will block the UI otherwise, which is also known as UI Blocking.