24/01/2019
Thursday

Full stack :

MEAN : MongoDB, Express JS, Angular JS, Node JS
MERN : MongoDB, Express JS, React JS, Node JS.

Protocol : Set of rules.

FTP : File Transfer Protocol. file Server
SFTP : Secured file transfer Protocol file Server
SMTP : Simple mail transfer Protocol mail Server
URL : Uniform resource locator.
http : Hyper text transfer Protocol. web Server

* The default port for http is 80

- GET
- POST
- Put
- Delete

} 4 imp http methods

We have to trace,

API → Use JSON [Java Script Object Notation]
(or)
XML

→ GET : Get method is used to get/retrieve the data only limited data without secure in the form of JSON type.

→ POST : Post method is used to creating new data. i.e; huge amount of data with secure. we can create (or) send & sent it huge amount of data.

PUT : Put method is used to update/modify the existing data.

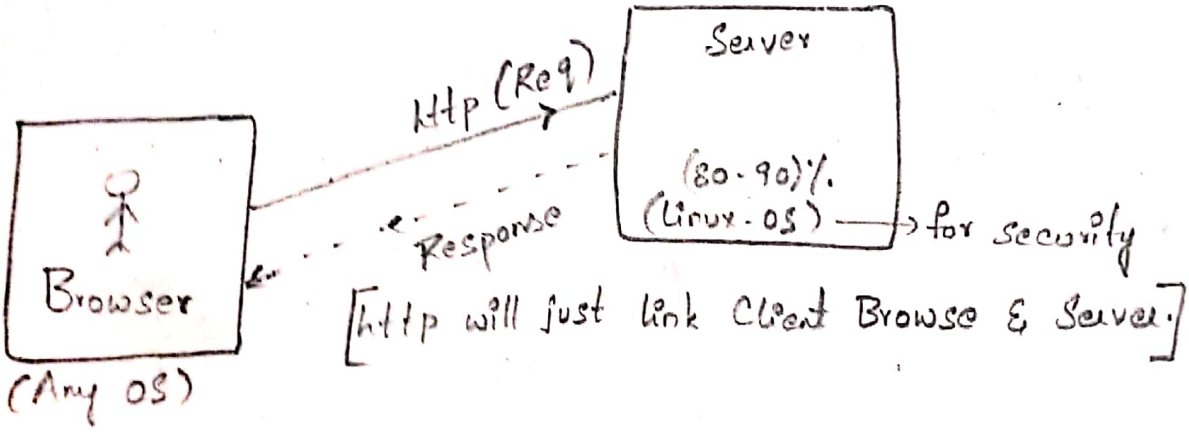
Delete : Delete method is used to Delete the data.

FTP: File Transfer protocol
Default port is 21 — file Server.

Https: hypex text transfer protocol with secure.
→ Default port is 443 — Web Server.

SFTP: Secured file transfer protocol.
→ Default Port is 22. — file Server.

SMTP: Simple mail transfer protocol
→ Default port is 25. — Mail Server.



On Server side:
- Java
- C#
- Python
- PHP
- .Net
- JS
- ~~Node JS~~

Frame works in Server side.
- Express JS

DB
- SQL
- !
- MongoDB

Front End
- ~~Html~~
- ~~Css~~
- ~~Java script~~
  - Angular JS
  - React JS

Frameworks in front End

~~JEE~~ — ~~SQL~~
Node JS — Mongo DB

To learn API's, we use JSON,
So, we will also learn JSON.

VS code [visual studio code] → developed by Microsoft.

html : 5 → current version

<DOC TYPE> tag will tell type of document.

Root
- <html lang="en"> (tagname) → html Element represent the root of an html document.
- </html> → Post Element.

html Element ⇒ Starting tag + Ending tag + Content.

html tag ⇒ wrap with 2 angular braces with tagname. ie; html tag

1] <a> ⇒ If the 'a' Element acts as an href attributes, then it represents hyper link. A hyper text ~~about~~ anchor labelled by its content.
⇓
used to link one page to another.

2] target → attribute [target = "_blank"]
⇓ in
to open external page

To open in internal page, we don't use target.

3] <img src = " " alt = " "> ⇒ represents image.
source alternate txt.

If error in the image path provided in the src, we use alternate text.

For free images, the link is https://Pixabay.an/en.

From the above link, copy the image into the folder

 : represents list of Elements

→ Order list

→ Elements are in proper order.

 : represents list of Elements.

→ Unorder list

By default numbers

1 test 1
2 test 2
3 test 3
4 test 4

- test 1
- test 2
- test 3
- test 4

↳ To change default orderlist.

To write paragraphs.

 : li Element represents a list item. If parent element is ar

"type" attribute → to change 1, 2, 3, 4 ---- to our choise based

"start" attribute → to start with order list.

Syntax:

```
<ol type = "  " start = " ">
    <li> test 1 </li>
    <li> test 2 </li>
</ol>
```

Block level Elements : If Elements (or) contents are in line by line then Element are Block level, Since they occupy entire width

Ex: h1 to h6

<P>
<h>

 ..10 ..10

(will occupy the Entire width.)

Inline Elements: If Elements (or) contents appear at browser are one after the other they are Inline Elements. Since they occupy only Element width only.

Eg: <sup>
<sub>

<img>.

Empty Elements: No need to close the tag. Since no content.

Eg: <img>

<input>.

<form> </form> For collecting user information (data).

form Element represents a collection of form associated Elements some of which can represent editable values that can be submitted to server for processing.

type of method of https. (Post method)

Eg: <form action = "#" method = "Post" > [No validate (without validating for connecting to server.

<Label for = "username"> Username </label>

← to connect Label & input "for" & "id" attributes are used

<input type = "text" id = "username"

To capture user information if post → at https we get user data.

name = "username". placeholder = "Enter username"

if get → No user information at https.

maxlength = "10", minlength = "3" required />.

[for the input for username with all the specifications].

<input type = "submit" value = "submit" />.

<input type = "radio" name = "gender" id = " " > female.

By giving name attribute, we make it conditional.

<input type = "radio name = "gender" id = " " >male

<input type = "checkbox" name = "language" id = " "> Java

<input type = "checkbox" name = "language" id = " "> .Net

<input type = "checkbox" name = "language" id = " "> Python

<select multiple > (To select multiple options)

To avoid it from selecting as an option.

<option value = "java" disabled > java </option>

<option value = "Python" > Python </option>

<option value = "ruby" Selected > ruby </option>

<option " " </option> To make it selected by default.

</select>

<input type = "submit" value = "submit" >.

Note :

<select>
</select> ← In order to Select → <datalist>
</datalist>

- We can select multiple options.
- It will not search. We have to dropdown the list.

- We cannot select multiple options.
- It will give type with search No need of dropdown. Rather we will search.

Eg: <datalist id = "language" >

<option value = "java" > </option>

<option value = "ruby" > </option>

<datalist>

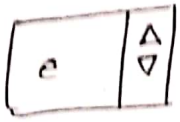
<input type = "hidden" > (for hiding purpose.)

Will not visible for Endusers.

<input type = "number" name = " " id = " " max = "10" min = "0"

step = "2" > → will accept from 0 to 9 numbers only.

with diff '2'.



Also, accepts 'e' → exponential.

<input type = "file" name = " " id = " " >

will accepts to upload file / choose file.

<input type = "range" name = " " id = " " >

(range)

for selecting the range

<input type = "date" name = " " id = " " >

↳ only dd/mm/yyyy.

<input type = "datetime-local" name = " " id = " " >

↳ with time & dd/mm/yyyy.

<audio controls .

```
<audio controls loop autoplay ="true" >
    < source src = "./audio link " type = " audio.mp3" />.
</audio>

< video controls loop autoplay >
    < source src ="/video link " type = " video.mp4" />
</video
```

25/01/2019
Friday

Semantic Element : Browser easily understands, there is no need for it to search and the document will be retrieved directly.

For Eg:

<Section> : Section is a element represents a generic section
</Section> : of document @ application.

<article> : The article element represents a complete (or) self
</article> : contained, composition in a doc, page, application or site

<header> : The content written inside the header will be at the
</header> : top of body. [like title].

<main> : It represents main content of body of a document
</main> : @ application.

Also, the paragraphs are represented in

<P> Lorem </P>
dummy.

<figure> : For image, we use figure wrapper. If we represent
</figure> : <img> inside figure, the browser easily understands

<figcaption> : To give title for the image.
</fig caption>

<footer> : The content written at the bottom of the Page is
</footer> : written inside footer.

The footer is written inside the <P> tags.

Eg: <P> Footer here </P>.

Code : Document

```
<section>
    <article>
        <header>
            <h1> Title here </h1>
        </header>
        <main>
            <p> Main body of the Document </p>
            <figure>
                <figcaption> Image title here </figcaption>
                <img src = "image link" alt = " ">
            </figure>
        </main>
        <footer>
            <P> Footer Content here </p>
        </footer>
    </article>
</section>
```

semantic elements

- <details> </details> : The details element represents the data by hiding it and showing it by click.

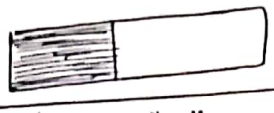
Eg:

```
<details>
    <summary> Click here to show summary </summary>
    <P> Content is written here and will be
        getting it by clicking. </p>
</details>
```

* We will get to see the data only once after by clicking This is due to the <details> tag.

<meter> : The meter element represents a scalar measurement within
</meter> a known range or a fractional value.

Eg: disk usage, the relevance of a query result or fraction of a voting population to have selected a Particular candidate.

fig: 

Syntax: `<meter value="3" max="10" min="1">out of 10</meter>`

<progress> : The progress element represents the completion process
</Progress> of a task. The progress is either indeterminate, indicating that progress is being made.

Syntax: `<progress value="90" max="100" min="1">out of 100</Progress>`

1.e <nav> : nav represents a section of page that links to other
</nav> Page. This is also a Semantic element.

Syntax:
```
<nav> //navigation purpose
    <ul>
        <li><a href="#">home</a></li>
        <li><a href="#">about
        <li>
        <li>
    </ul>
</nav>
```

5.e <aside> : aside element represents
</aside>

CSS
- Cascading Style Sheets.

Three ways you can reference style sheets in html Page.

1.] Internal style sheet - with the help of style tag.

2.] External style sheet - with the help of link tag.

index.html	index.css
html file	css file.

Syntax: `<link rel="stylesheet" href="index.css">`

→ So, we can link the css file into html Page by using <link> tag. i.e; above link.

→ This should always be in head Section only.

i.e;
```
<head>
    <link>
</head>.
```

<link> : The link elements allows authors to link their document. to other resources.

css : In order to create css, we have to create new folder with .css extension.

Note: Always External way is the best way to apply css.

Selectors:

There are mainly 3 types of Selectors.

i.] id Selectors

ii.] Class Selectors

iii.] Element Selectors.

Always, the browser priority will be as id → class & → Element Selectors.

Eg: 1.] Id Selector:-

html	css
type1	#name1 { color : red }
type 2	#name 2 { color : blue }.

1.] id Selector will not allow duplicate id names. So, this will always be the first priority for the browser. [becoz only one type is allowed.]

2.] "#" used to represent id in the css script

2] Class Selector:

html	css
type 1	.name 1 { color : red }
type 2	.name 2 { color : red }

1.] Class Selector will allow duplicate class names. This is given as second priority.

2.] "." used to represent Class in the css script.

3.] Element Selector:

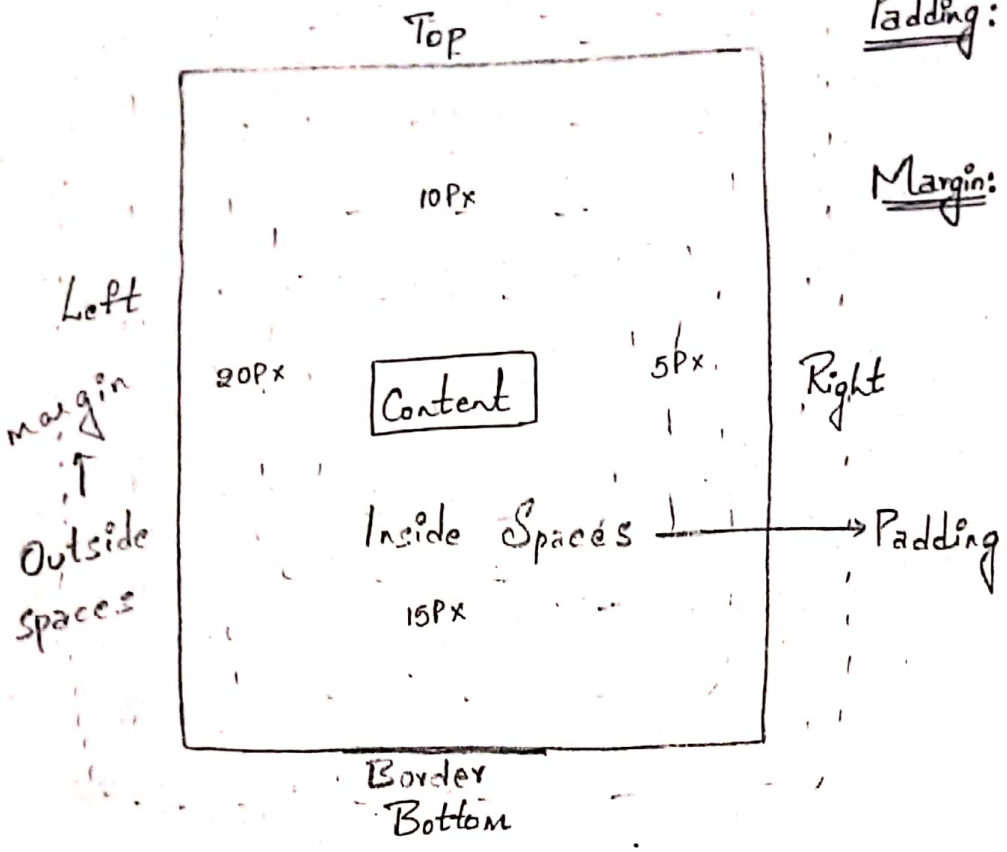
html	css
type 1	div { color : red ; }
type 2	

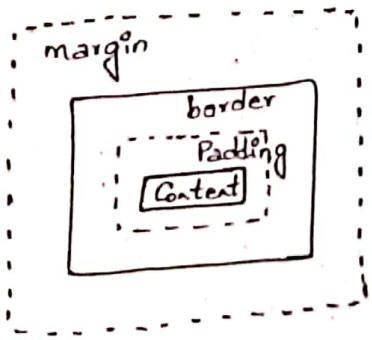
1.] For, Element Selector, we cannot get two different inputs. Only one input can be passed for two types. This is also Decided by browser.

Box Model in CSS:

Padding: Providing spaces Inside box.

Margin: Providing spaces Outside box.





Padding : 10Px (T) 5Px (L, R) 15Px (B)

Padding : 10Px (T, B) 5Px

Padding : 10Px (T) 5Px (R) 15Px (B) 20Px (L)

Margin : 100Px (T, B) Auto (L, R)

box width : 300px.

1.] After applying width, the margin alignment gets changed accordingly.

2.] To apply center alignment for the text, we make use of text-align : center.

3.] To make changes for image, we can use

img { display : inline-block ;

4]* User Agent Style sheet is the default browser for CSS.
we can override it with custom CSS.

5] For applying shadow feature, make use of box-shadow property with parameters and colour. [For applying Shadow]

Eg: box-shadow
: 0px 3px 85px 8px pink;

. By default, the shadow will appear outside. If we want it inside, make use of "inset keyword" in the syntax.

Eg: box-shadow
: inset 0px 3px 85px 8px pink;

6] text-transform : capatize → Chandu
Upper Case → CHANDU
Lower Case → chandu.

7] letter spacing : 0.4Px
To get the spaces b/w the letters in a word

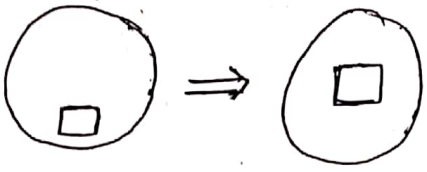
8] To get fonts : 1] Go to "https://fonts.google.com" website.
2] Search for the required type of font and then copy the link of font in popup.
3] Then paste it in the header after the title bar
4] After paste the font type in CSS.

9] font-weight : --Px;
To change the thickness of text (bold (or) normal etc..)

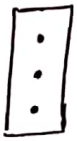
10.] font-size ~~color~~ : To change the text size [--Px;].

11.] color : To apply color to any field. [#ASCII code].

12.] line-height : To change the vertical position



13.] list-style : To remove radio buttons in Ul.


. bullets r removed.

Eg: Ul {
    list style : none;
    Padding : 0; // To remove the default browse
    margin : 0; // CSS behaviour.
}

14.] Span : This will not allow height, width, margin top, margin bottom properties.
If we want to apply we use display : block;

Eg: display : block; → It will occupy entire width.
display : inline-block; → Occupy only element width.

15.] img : hover To apply the animation effect to the image. [i.e; for rotation].

Syntax: img : hover {
    Padding : 40Px; // So, the img will rotate
    transform : rotate (360deg); 360° for 1.6 sec time
    transition : ease-in-out 1.6s all; Period.
}

li i {

font-size = 40px ; !important

Even if we increase the font size, sometimes it will not increase. So, we take help of a 3rd party in CSS and override them to "highlight certain parts" [!important is one among them.]

li : hover : To change the Animations, colors of icons. while moving the cursor. We will be declaring all the necessary Properties inside this Object { }.

Eg: li : hover {

Padding : —

transform : —

transition : —

color : —

margin : —

}

Clear = both whenever we use margin 0, auto, we should be having the width, in order to seperate, we use clear = both.

What is the current element that will allows to add multiple images at background? (CSS MCQ).

- background image.

Project 2 : https://html5up.net/uploads/demos/minport/

```
</head>
<body>
    <section id="headerblock">
        <nav>
            <ul>
                <li><a href="#">Top</a></li>
                <li>      "      >Work<   "
                <li>      "      >Portfolio<  "
                <li>      "      >Contact<  "
            </ul>
        </nav>
    </section> <!--headerblock-->
    <section id="sliderblock">
        <div class="leftblock">
            <figure>
                <img src="./.." alt=" ">
            </figure>
        </div> <!-- leftblock -->
        <div class="rightblock">
            <header>
                <h1> Hi. I'm Chandu. </h1>
            </header>
            <main>
                <P> The text what we wished is written
                    here [copy pasted] </P>
            </main>
            <footer>
                <button> button type text </button>
            </footer> <!--rightblock-->
</body> </section> <!--sliderblock-->
```

Style.css :

```
body {
    margin : 0;
    Padding : 0;
}
#headerblock {
    background :#000;
    height     : 50Px;
    line-height : 50Px;
}
ul {
margin : 0;
Padding : 0;
list-style : none;
}
ul li a {
  color : #fff; //default color of browser is blue.
text-decoration : none;
  Padding : 10Px;
}
ul li {
  display : inline-block;
}
nav {
  text-align : Center;
}
.leftblock {
    width : 40%
    float : left; // to align side by side.
}
.rightblock {
    width : 60%;
    float : left;
}
# sliderblock {
    Padding : 70Px 40Px 70Px 40Px;
}.
```

Opacity : It is used to make the content (or) anything to blur which we want.

26/01/2019
Saturday

Html5 tags :

1) Selection
2) article
3) aside
4) details
5) figure
6) fig-caption
7) footer
8) header
9) main
10) nav
11) summary
12) time
13) address
14) mark
15) meter
16) Progress
17) datalist

⇒ Above all are Sematic elements. Expect datalist.

Html5 attributes :

1) Color
2) date
3) datetime
4) datetimelocal
5) Email
6) month
7) number
8) tel
9) range
10) search
11) url time
12) weak
13) autocomplete
14) autofocus
15) min
16) max
17) maxlength
18) min length
19) Pattern
20) multiple
21) Placeholder
22) required
23) step
24) hidden
25) selected
26) disabled
27) not validate.

Bootstrap

Steps:

1.] Google → get bootstrap → https://getbootstrap.com.
Go to the above website page and the required version is selected.

Bootstrap: 1.] Bootstrap is an open source toolkit for developing HTML, CSS & JS application.

⇒ The latest version of bootstrap is "V4.2X".
⇒ for capgemini, we will be using on "V3.37".

2.] Select the version of bootstrap. We need V3.3.7.
Clicking on the version we want, will lead to another link https://getbootstrap.com/docs/3.3/.

3.] Click on download → Select Download Bootstrap.

4.] After downloading, we need copy paste in project folder. Extract the files. [Remove zip folder].

2] Bootstrap is the most popular HTML, CSS & JS framework for developing responsive, mobile first projects on the web.

3] Bootstrap works with predefined classes. We have to just use the respective class name for using it accordingly.

4] In order to make the code more simplier, we go for bootstrap.

5] Instead of customized CSS documents, we go for bootstrap and achieve the tasks accordingly.

To call bootstrap, we have to set link as

<link rel="stylesheet" href="./bootstrap-3.3.7-dist/css/bootstrap.css">.

- Bootstrap works with predefined classes.

info - light blue
success - green
danger - red
warning - orange
Primary - dark blue
default - No background (white)

} button colors.

Syntax: <button class="btn btn-info"></button>

→ Tables are used for tabular purpose.

<table class="table">
<table class="table table-dark">
<table class="table table-stripped">
<table class="table table-bordered">.

→ th = bold.
→ td = normal
→ tr = row.

Ex: <table class="table">
<tr>
<th></th>
" "
" "
</tr>
<tr><td></td>
" "
" "
</tr>.
</table>.

Fluid Container:

Turn any fixed width grid layout into a full width layout by changing your outermost.

i.e; Contain to container-fluid.

```
<div class = "container-fluid">
  <div class = "row">
      =
  </div>
</div>.
```

Grid-System: [Bootstrap follows Grid system]

Bootstrap includes a responsive, mobile first fluid grid system that approximately scales upto 12 columns as the device or viewport size increases. It includes predefined classes for easy layout options, as well as powerful mixins for generating more semantic layouts.

Bread crumbs:

These are used to create secondary menus.

Syntax:

```
<ol class = "bread crumb">
    <li><a href = "#"> Home </a></li>.
    <li><a href "#"> Library </a></li>.
    <li><a href-"#"> Data </a></li>
    <li><a href= "#">
    <li><a href = "active"> Data </a></li>.
</ol>.
```

says this is the last menu.

& active all the time (shows/highlights).

Typography : [Fonts]. [h1, h2, h3, h4, h5, h6 classes.]

Alignment classes : (for paragraphs). <h1 class = "h1"> hello! </h1>.

lead class:
↳ text-left

<P class = "lead text left">

text-center

text-right

text-justify

text-nowrap - break the content.

Transformation Classes:

text-lowercase

text-uppercase

text-capatilize.

global css's : bootstrap.css

bootstrap.min.css.

Development → under construction. We can't use [banking websites]

Production → We can use even the code is getting modified.

In production, min.css used. It removes spaces in style code to increase the performance.

⇒ To get the .css file from min.css file → Go to internet → copy the min.css file & convert into .css code.

<ul class = "list group">

<li class = "list-group-item">.

List group & list group item classes for to change default behaviour [user agent style sheet].

form group ⇒ To group all the content

form control ⇒ To take Content

Ex: text-radius -- etc.

Step 2 : To linkup the given program with Bootstrap.

⇓

Need to link the bootstrap to the html. in the head section.

<link src = "./bootstrap"> (Select)

⇓

Bootstrap depends on JQuery. So, download it.

⇓

Go to jquery.com

⇓

Download a Version of 3.3.1

⇓

Copy the link from the tab and that link is Pasted within the <script> tags in the head section of the html program.

<script src = "copied link"></script>.

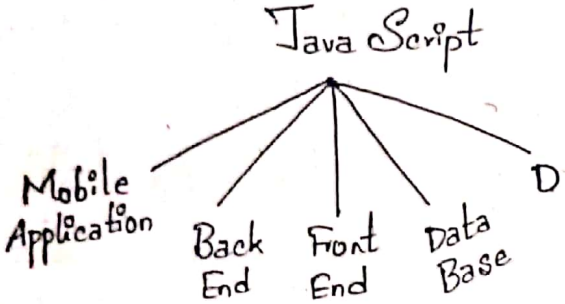
⇓

Now, get the bootstrap work for javascript. So, get the bootstrap.js inside the <script> tags in the head section.

<script src = "./bootstrap-3.3.7-dist/js/bootstrap.min.js"></script>.

JAVA SCRIPT

- Java Script is the most popular and world's most used scripted programming language.
- This is not Object Oriented.
- Java Script is → functional programming language.
  → Weakly typed language.
  → Unlight / loosely tighted language.
  → Prototype based language.
  → Dynamic programming language
  → Scripted language
  → Procedural language.



- We can use JS with Browse & without Browse.
- Html, CSS & JS we need to compile only in Browser.
- In Java, JRE/JDK/JVM are helpful in compiling & Execution.

Html/css/JS

↓

Browser.

↓

JS Engine

↓

V8 Engine.

⇒ Container is the bootstrap default class which give left side right side margin.

Syntax:

⇒ Bootstrap have 12 strips in it.

classes
- col-xs-12 → mobiles
- col-sm-4 → tablets
- col-md-4 → laptops
- col-lg-4 → TVs.

→ These classes are used for setting the size of the input boxes for every type of system like mobiles, tablets, laptops & T.V's.

→ "col-md-offset-4": This is the class which is used for keeping the text at the centre. (that may vary) (used for lapi's)

→ For mobiles, it is col-xs-offset-4
For TV's, it is col-lg-offset-4.
For Tablets it is col-SM-offset-4.

⇒ Nav:

This is used for Navigation, default classes are class = "navbar navbar default" (inverse-black). white colors.

For mobiles, the default classes are

i.e; navbar-toggle collapsed.

class = "navbar-toggle collapsed."

For menu, the default class is

class = "navbar-collapse collapse"

class = "nav navbar-nav navbar-right"

If the display context needs at right side, we want to use this.

We will be getting this code from

Eg: https://bootstrapdocs.com/v3.3.6/docs/examples/navba

```
<body>
<nav class="navbar navbar-default">
    <div class="container-fluid">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed"
          data-toggle="collapse" data-target="#navbar"
          aria-expanded="false" aria-controls="navbar">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="#">Project name</a>
      </div>
      <div id="navbar" class="navbar-collapse collapse">
        <ul class="nav navbar-nav navbar-right">
          <li class="active"><a href="#">Home</a><li>
          <li> a href="#" _____ >About<
          <li> a href="#" _____ >Contact<
        </ul>
      </div><!----/.nav-collapse--->
    </div><!---/.contain-field---->
</nav>
```

Pagination: To move to nextpage (or) directly to lastpage(or) so other random pages.

Ex: <nav ⇒ formenu

```
<ul class = "pagination">
    <li>
      <a href = "  " ></a>
    </li>
    <li><a href = "#">1</a></li>
          "          >2<    "
          "          >3<    "
```

Jumbotron: For Highlighting purpose. (things).

It is light weight, flexible component that can optionally extend the Entire viewport to showcase key content on your site.

```
<div class = "jumbotron">
    <h1> Hello! </h1>
```

Alerts: Any of the warnings, requirements, events ..etc. ~~are~~ which are to be showed by special pop up's are called Alerts.

Ex:

```
<div class = "container">
    <div class = "alert alert-success" role = "alert"> </div>
                  "alert alert-info"    → blue    "
                  "alert alert-warning" → Orange  "
                  "alert alert-danger"  → red.    "
```

End of Bootstrap for CSS...

JavaScript:

1Q: *** Carousel class: This is for the sliding purpose of Pictures. for this, we need code that is obtained from jQuery.

Link for jQuery code : https://code.jquery.com.

* Bootstrap Javascript depends on Jquery library.

carousel slide – for slider purpose.

model class: For generating pop ups.

Steps to get used for for jQuery in bootstrap:

Step 1: Go to the getbootstrap.com website.

⇓

Select the version V.3.37. [top right corner]

⇓

Now Go to JScript on the navigation bar.

⇓

i.e; carousel(here).

Select the class which we want from all the classes present there.

⇓

Copy the code present there ** for the class we have selected and paste it in the Editor (body)

/* ... Go to minified.js in google to convert min.css to → .css. ... */

MCQ's:

1.] What is 'V8 Engine'?

"V8" is Google's open source high performance JavaScript and WebAssembly Engine, written in C++. It is used in Chrome and in Node.JS, among others. It implements ECMA Script & Web Assembly.

It's job is to convert Normal code into Machine code.

2.] What is Spider Monkey?

"Spider Monkey" is the Mozilla's javascript engine written in C & C++. It is used to converts normal code into machine code.

3.] Browsers with engines?

→ Internet explorer uses "chakra" developed by Microsoft version of Internet explorer 9 (IE9) web browser.

→ Safari uses "webkit engine" to convert normal code to machine code.

4.] Node JS:

→ Node JS is developed by "Ryan Dahal" without using browser with the help of V8 by rapping V8 code with C++.

→ Node JS is a javascript runtime built on "chrome's V8 javascript Engine" or have a look at the Long Term Support (LTS) schedule.

→ Browser supports window Objects

→ Node.js supports Global Objects.

JavaScript:

Datatypes:

Primitive Data types	Non-Primitive Data types.
String	functions } object types
number	Objects } object types
Boolean	Arrays } object types
undefined	
null	

2. Functions:

i] Internal way of using JavaScript in html file.

Eg:
```
<body>
    <script>
    </script>
</body>
```

ii] External way

Eg:
```
<body>
    <script src="  ">
    </script>
</body>
```

iii] Inline way of using JS is by using Attributes.

Alert function: Alert method is used for popups and alert give only one option at popup.

alert ("Hello") [OK].

Confirm function:

confirm ("Hello this is confirm method.").

Confirm is also used for popup's with 2 options.

[OK] [Cancel]

Inbuilt functions in Javascript: Only client side functions. (Front end JavaScript)

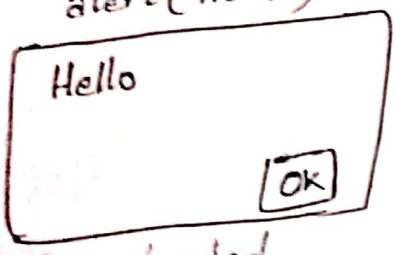
// for popups:-

Alert() - Popup with only one option. [Ok]

Confirm() - Popup with two options. [Ok] [Cancel]

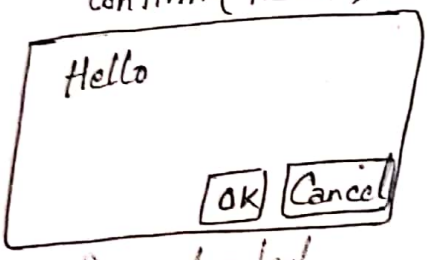
Prompt() - Before page loading user can Enter data in popup with 2 (two) options [Ok] [Cancel]

alert("Hello")



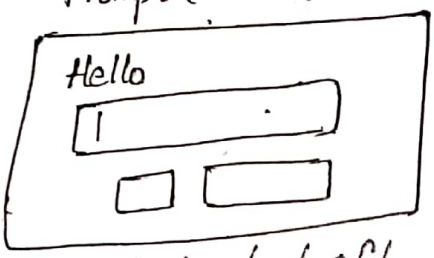
Page loaded

Confirm("Hello")



Page loaded

Prompt("Hello");



Page not loaded after loaded.

document.write: To write content at Browser. (O/P).

doooument.write ("Hii group");

→ window.alert()
→ window.confirm()
→ window.prompt()
→ window.document.write()

} These are all work for Browser, not for Node JS.

Without window also executed. Browser will know by default it is window functions.

Console.log ("Hello!") :- It is a Universal function. It will work for JS & Node JS.

(<script>)	(Cmd)
(front End)	(back end)
(Enduser/Client)	(developers)
(Browser client)	(Server).

o/p at Console.

Hello!

We get the o/p at Console.

Console.log (type of Var) :- This will tell type of data given by developers in pgm.

Console.

```
Var i = "Chandu" <Enter>
> undefined
 type of i
>"string
```

Undefined datatype : It is variable not initialized and not yet use.

Console

```
Var test;
>undefined
 Var test = "Chandu"
> undefined  // Since not yet used.
```

Null : Empty value. Instead of no. value, we initialize with "null."

Console.

```
Var test = null;
> undefined
type of test
>"object".
```

. Diff. b/w undefined & Null

undefined	Null
1] Declare variable, not initialized and not yet used.	1] Declare variable and initialize with null.
2] undefined	2] Object.

⇒ JavaScript is undertaken by Net.

ECMA company (maintainance)

European Computer Manufacturers Association Script.

JS 5 → ES5 (Versions)

JS 6 → ES6 (ES 2015) ⇒ Std. Version currently.

JS 7 → ES7 (ES 2016)

All browsers support ES5 & ES6.

Most of the browsers will not support ES7.

ES5

1] Var

2] Console

```
Var i=10;
undefined
Var i="Chandu"
undefined
```

Var can overrided

3]
```
if()
{
  Var i="test";
}
document.write(i);
```

No Error — Since, Var can use anywhere Data overflow can happen.

ES6

1] let
const

2] Console

```
Const i="Chandu"
undefined
const i=10;
Error
```

Const can't be overrided.
We can't use multiple times.

3]
```
for(let i=10; i<=10; i++)
{
  Console.log(i) // o/p: 10
}
Console.log(i) //Error
```

Var: Var keyword can be used anywhere & overrided. Ex: Var i = "Chandu"
Var i = 10;

Const: Const keyword will constant the value. We can't override value.

Let: let, is a single that the Var may be reassigned, such as counters in a loop. (or) a value swap in an algorithm.
It also signals that, the variable will be used inside block only.
Data will not overflow.

Ex:
```
Var i = "Chandu"
for(let i=0; i<=10; i++)
{
  console.log(i) // 0 1 2 ...
}
console.log(i) //Chandu
```

```
let i = "chandu"
for(let i=0; i<=10; i++)
{   X Can't override
  Console.log(i)
}
```

```
Var i = "Chandu";
for (Var i=0; i<=10; i++)
{
  Console.log(i) //0 ...
}
Console.log(i) //0, 1, 2 ... 10.
// First priority to local variable
```

⇒ **ES5**
for strings " "

ES6
for strings ` ` (Backtics)
` ` called Templet literals.

⇒ Templet literal are string literals allowing Embedded expressions
⇒ You can use multi line strings & string interpolation features with them
⇒ They are called "templet strings" in prior editions of ES2015.
*⇒ Without going to html, we can work in JS only using templet string

In ES6:
```
Var name = "Chandu"
Var str = `<section> Hello. this is String
            <h1> ${name} </h1>
            <P> ..... </P>
            <button> Submit </button>
          </section>`;
document.write(Str);
console.log(typeof Str);
```
Interpolation calling variable ${ }

28/01/2019
Monday

Non-Primitive:

Arrays: Can hold multiple values. It is a collection of Data.

Data of array is homogenous or heterogeneous data.

Eg: Var array = ["java", "nodeJS", 'angular', 'react', "javascript", 10];

```
console.log(array);
var test = array[0];
console.log(test);
```

O/p: array (6)
↳ Data given (→1)

⇒ Another way to create array with "new" keyword. using constructor way.

Eg: Var array1 = new array ["java", 'python', 'nodejs', 'true'];

```
Console.log(array1);
Var test = array1[0];
Console.log(test);
```

O/p: 4 ["java", "python", "nodejs", "true"]

0: java
1: python
2: nodejs
3: true.

⇒ The array objects lets you store multiple values in single variable.

⇒ An array is used to store a collection of data, but it is often more useful to think of an array as a collection of var's of the same type.

typeof array; → Object type

All non-primitive data types are of Object type.

1MCQ

Array.isArray(): To check whether given var is array or not

It returns boolean.

if array ⇒ true

if not an array ⇒ false.

To Convert String into Array:

```
Var Str = "Sakthi Chandu";
undefined
type of Str
"String"
Var arr = Array.from(Str)
undefined
arr
(6)["S", "h", "a", "n", "d", "u"]
type of arr
"Object"
Array.isArray(arr)
true.
```

10:MCQ

Spread Operator:

The Spread Syntax

The Spread Syntax is simply "three dots" ...

It allows an iterable to expand in places where 0+ arguments

are expected.

Combine Arrays:

There have always been a variety of ways to combine arrays, but the spread operator gives use a new method for combining arrays.

```
Var arr1 = ['two', 'three'];
Var arr2 = ['one', ...arr1, 'four', 'five'];
```

Eg:

```
Var array = ["java", "node JS", "angular", "react", "javascript", 10];
Var array1 = new Array("java", "python", "node JS", true);
Var str = "hello node JS";

//javaScript way
Var test = array.concat(array1);
console.log(test);

//esc way
var array3 = [...array, ...array1, ...str];
console.log(array3);
```

** Objects:

Objects consists of both keys and values. whereas arrays consists of only values.

```
Var Obj = {
    name : "vinay",
    age : 20,
    Company : "QSP"
}   // Don't end with ;/,/;
```

key-value Pair

This is "JavaScript Object Literals."

```
Var Obj1 = {
    "name" : "Chandu",
    "age" : 20,
    "Company" : "JSP"
}
```

Notation

JSON doesn't accept functions.

•JSON

This is "JavaScript Object Notation." [JSON]

to retrieve data: Console.log (Obj.name).

Eg:

```
Var obj = {
    name : "vinay",
    age : 20,
    Company : "QSP"
}
var test = obj.name;
console.log(test);
```

} literals way

//

```
var obj1 = new Object();
    obj1.name = "Chandu",
    obj1.age = "22",
    obj1.company = "JSP"; // End with ;
```

} literal with structural way.

To retrieve:

```
var test1 = obj1.name;
console.log(test1);
```

} (or)

```
console.log(obj1.name);
console.log(obj1.age).
```

→ By using these keys, we retrieve data.

* In JavaScript, we create Objects literal way.
* In JSON, if we work with Java, we follow Notation way.

Note: JSON will not allow functions.

Ex: "company": function() ✗

⇒ We retrieve data from JSON using ANGULAR JS.

Functions: It is just a block set of instructions need to execute.

→ Set of instructions is called function with pre-defined for reusability purpose.

→ Write once, use multiple times.

Syntax: function() {

}.

1.] Ananamous function: Eg: function() {
alert("hello");
}

→ The function which doesnot have proper name is called Ananamous function.

1.] Anonymous functions are function that are dynamically declared at runtime. They're called anonymous function because they are not given a name in the same way as normal functions.

2.] Anonymous functions are declared using the function operator instead of the function declaration.

3.] You can use the function operator to create a new function wherever it's valid to "put an expression."

Eg: function() {
    alert("hello");
    }

// In order to call this function, we need to convert it Anonymous function to Anonymous Variable (function expression) // declaration function.

2] Function Expression: [variable declaration function]:

A Function expression defines a function as a part of a larger expression syntax (typically a variable assigned). function defined via functions expression can be named or Anonymous.

Function expression must not start with "function".

Eg: [Syntax:] Vartest = function() { // function declaration.
    alert("hello");
    } called part
}

for retrieving: test(); → calling part

Inline JavaScript:

In order to give connection between index.html & function.js

onclick = "test()" [function variable declaration]

<button onclick = "test()"> submit </button>.

If we want to control the function i.e; unnecessarily not Executed, if we need that only then only execute. We go for Inline Calling of JavaScript.

In Inline Javascript, we use Attributes.

for Eg: The function has to execute when we click on button.

html: <button onclick = "test()"> Submit </button>.

3] Named functions:

The created function is named, which means that name Property of the function object holds its name.

```
function test()
{
  document.write("Hello"); // called part
};
test(); // calling part.
```

To restrict this function, use html

<button onclick = "test()"> Submit </button>.

4] Immediately Invoked Function Expressions: [IIFE]

It doesnot take very long working with Javascript, before you come across this pattern:

```
(function() {
  // Login here
})();
```

for called part ← for calling part

→ your first encounter is likely to be quite confusing.
But fortunately the concept itself is simple.

→ The pattern is called an immediately invoked function expression, or IIFE (Pronounced "iffy").

Eg: i] Invoke functions directly Executed.
ii] There will be No calling part.

MCQ: Properties:

Function Hoisting:

The function declaration function hello {...} create a variable hello that is hoisted to the top of the current scope.
hello variable holds the function object.

Eg:
```
hello(); //calling part
function hello(){
    alert("Chandu");
}
```
called post

Hoisting
```
test();
function test()
{
    document.write("Hello");
};
```
→ test();

→ Function hoisting is only for named functions.

→ hoisting is done by default & by V8 in Java script Engine.

Syntax for types of functions:

function()	var test = function()	function test()	(function(){
{	{	{	})();
}	}	}	

Closure : [Declaring a function within function.]

1.] A closure is an inner function that has access to the outer (enclosing) function's variables - scope chain.

2.] The closure has three scope chains.
i) it has access to its own scope (variables defined b/w its curly (flower) braces).
ii) it has access to outer functions variable.
iii) And it has access to the global variable.

3.] The inner function has access not only to the outer function's variables, but also to the outer function's parameters.

4.] Note that the inner function cannot call the outer's function's arguments object. however, even though it can call the outer function's parameters directly.

Eg:

```
function show() {  // outer function
    var outerblock = "hello this is outertext";
    document.write(outerblock + "<br>");
    function innerfunction() {  // inner function.
        var innerblock = "hello this is innertext";
        document.write(innerblock)
    }
    return innerfunction();  // inner function returning to outer
                             function to execute.
};
show();  // outer function calling.
```

Function with parameters :

Eg:

```
function addnumbers (a, b) {
    var c = a+b;
    document.write (c)          O/P : 7.
}
addnumbers (3, 4)
function test (a, b) {
  return a+b; // return is optional.
}
// document.write (test (10, 10));
```

— Functions for ES-5. —

Arrow Function : (From ES-6, we use fat arrow function)

1] This is called the lambda function.

2] This will be working only for Anonymous and function expression.

3] An arrow function expression is a syntactically compact alternative to a regular function expression.

4] Arrow functions (often referred to as "fat arrow functions or lambda functions)"

5] Their short syntax is further enhanced by their ability to return values implicitly in one line, single return functions.

6] Arrow functions can never be named and as such are always considered anonymous functions.

Eg:

```
var test = () => function () {
    alert ("test");
}
```

```
var test = () => {
    alert ("test");
}
```

The word "function" in the ES-5, is converted and it is changed to "flat arrow". And this is in usage from ES-6.

Eg:

```
var test = (a, b) => {
    return a+b;
}
document.write (test(40, 40));
```

```
var addnumbers = (a, b) => a+b;
document.write (addnumbers (10, 20));
```

} if single statement.

```
var test1 = element => element;
document.write (test1 ('Chandu'));
```

(i)

```
var test2 = (element) => {
    return element;
}
document.write (test2 ("kavya"));
```

(ii)

next Prg Eg:

} if more statements. i.e; two inputs.

Note:

Trim the above program into one line. Since, it has one Parameter, one line of program.

⇒ remove () & { & return.

Eg:

```
var test2 = Element => Element.
document.write (test2("kavya"));
```

end of functions

Note: In order to validate the inputs given, we need to use "if condition" in the JS code.

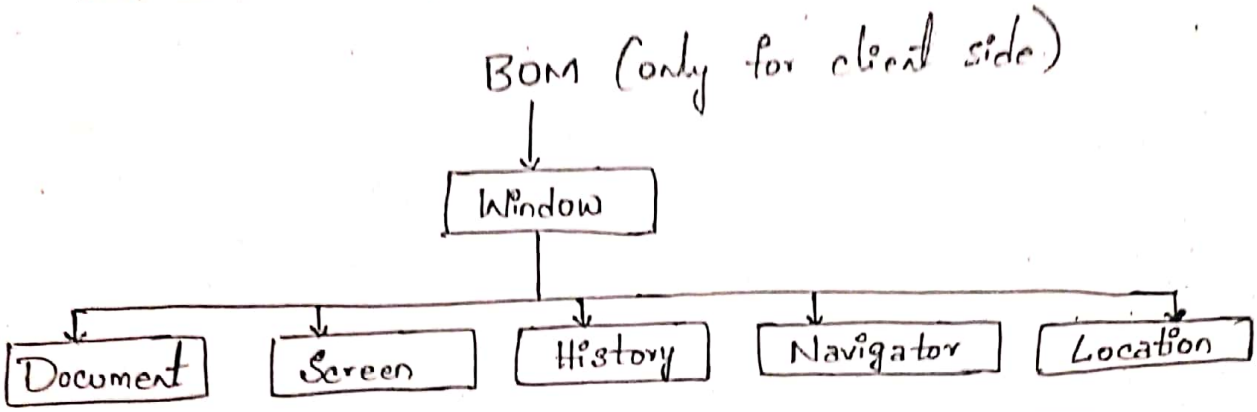
Eg:
```
if (isNan (firstNumber)) {
    alert ("Please add first number");
    return;
}
if (isNan (SecondNumber)) {
    alert ("Please add second number");
    return;
}
```

"isNan" is a predefined keyword to verify the input as the number. If we give input as string, we will be getting the result as "Nan." [Not a number].

BOM [BROWSER OBJECT MODEL]

1.] Bom is Browser Object Model, is a hierarchy of browser objects that are used to manipulate methods and Properties associated with the Web browser itself.

2.] Objects that make up the BOM include the window object, navigator object, screen object, history, location object and the document object.



Screen - Frames.

2.] Screen Object :

The Screen Object contains information about the visitor's screen.

NOTE: There is no public standard that applies to the screen object, but all major browsers support it.

Screen Object Properties :

1.] Width returns the total width of the screen.

2.] availHeight returns the height of the screen (excluding the windows Taskbar).

3.] availWidth returns the width of the screen (excluding the windows Taskbar).

4.] colorDepth returns the bit depth of the color palette for displaying images.

5.] For displaying images height returns the total height of screen

6] PixelDepth Returns the color resolution (in bits per pixel) of the screen.

window.Screen.property

Eg: Window.Screen.width
1366
Window.Screen.height
768
Window.screen.availwidth
1366
Window.screen.availHeight
728
Window.screen.PixelDepth
24
Window.Screen.colorDepth.
24.

3] History Object:

1.] The History Object maintains the Url's visited by the users (within a browser window)

2.] The History Object is a part of the window object & is accessed through the window.history property.

Note: There is no public standard that applies to the history Object but all major browser supports it.

i) Length returns the number of URL's in the history list.
ii) back() loads the previous URL in the history list.
iii) forward() loads the next URL in the history list.
iv) go() loads the specific URL from the history list.

Eg: Window.history.length <Enter>
Window.history.back(); <Enter>
Window.history.forward();
Window.history.go();

without UI's we use code for back/forward/go.

4] Navigator :

The Navigator Object contains in formation about the brows

Navigator Object Properties :

1] appcodeName Returns the code name of the browsers.

2] appName Returns the name of the browser.

3] appVersion Returns the version information of the browser.

4] Cookie Enabled Determines whether cookies are enabled in the browser.

5] **geolocation Returns a geolocation Object that can be used to locate the user's position.

6] Language Returns the language of the browser.

7] online Determines whether the browser is online.

8] ~~Platform~~ Product Returns the engine name of the browser.

9] Platform Returns for which platform the browser is compiled.

10] User agent Returns the user-agent header sent by the browser to the server.

11] Connection Returns network connection information. i.e; Speed of network.

Eg :- window . navigator. appCodeName
Mozilla
Window . navigator. appName.
Netscape.
Window . navigator. appVersion.

Window . navigator. connection.

Window . navigator. devicememory

Window . navigator. platform.

Wednesday

Web Storages:

The Web storage is used to store the data in the web or browser. This is done by 3 ways.

i.] local Storage
ii.] Session Storage } Html5, upto 5MB data can be stored. [browser storages]
iii.] Cookie Storage. → html 4 → Client & Server storage.

1.] local Storage:

1.] local Storage has no expiry. We can store the data Permanently. It is a type of web storage (5MB). [Browser storage]

2.] The local Storage Object stores the data with no expiry data. The data will not be detected, when the browser is closed and will be available the next day, week or year.

Example:

In Html:

```
<body>
    <div class = "container">
        <div id = "result"> </div>
    </div>
    <script src = "./storage.js"></script>

</body>
</html>
```

Storage.js:

```
// store
localStorage.setItem("lastname", "Chandu");
```

// Retrieve

document.getElementById("result").innerHTML = localStorage.getItem("last name");

In order to go to the Web Storages,

Right Click → Inspect → >>> (right side top corner) → application → localStorage → link.

ii.] Session Storage :

1.] Session Storage will store for the particular session. It is temporary. (5MB). It is the Browser Storage.

2.] The Session storage object is equal to the local storage object, except that it stores the data for only one session.

3.] The data is deleted when the user closes the specific browser tab.

Eg: "html code" is same as the local Storage prgm.

Storage.js :

// Store

sessionstorage.setItem("Lastname", "Chandu");

// Retrieve

document.getElementById("result").innerHTML = sessionstorage.getItem("Lastname");

iii.] Cookie Storage :

1.] Cookies stores only 5kB of Data. It can transform data to the server. It can be stored either in client side or in server side.

* What are Cookies ?

Cookies are data, stored in small text file, on your computer.

→ When a web server has sent a web page to a browser, the connection is shut down and the server forgets everything about the user.

→ Cookies were invented to solve the problem "how to remember information about the user": when a user visits a webpage, his name can be stored in a cookie. Nexttime, the user visits the page, the cookie 'remembers' his name.

2] Cookies are saved in name-value pairs like :

document.cookie = "username = Chandu"; expires = Thu, 18 Dec 2013 12:00:00 UTC";

Note: To clear local storage memory,

localStorage.clear();

O/p: data removed in localStorage.

* geolocation :

This is helpful in getting the user's location by "latitudes and longitudes."

Eg:

```
var x= document.getElementById('result');
function getLocation()
{
  if(window.navigator.geolocation)
```

```
{
  window.navigator.geolocation.getCurrentPosition(showPosition);
}
else
{
  x.innerHTML = `getlocation is not supported`;
}
};
function showPosition(position)
{
  x.innerHTML = `latitude: ${position.coords.latitude} <br>
                 longitude: ${position.coords.longitude}`
}.
```

inbuilt

Html code :

```
<body>
    <div class="container">
        <div id="result"></div>
            <button onclick="getLocation()">Submit</button>.
        </div>
        <script src="./storage.js"></script>
</body>.
```

o/p: Inspect → submit → o/p

submit

longitude : 78.3946993

Latitude : 17.495815099999998.

5] Location Object :

The Location object contains information about the current URL.

The location object is a part of the window object and is accessed through the window.location.property.

hash : sets or return the anchor part (#) of a URL.

Host : sets or returns the hostname and port number of a URL.

hostname : sets or returns the host name of a URL.

href : sets or returns the entire URL.

Origin : Returns the protocol, hostname and portname of a URL.

Pathname : sets or returns the ~~port number~~ Path name of a URL.

Port : sets or returns the port number of a URL.

Protocol : sets or returns the protocol of a URL.

Search : sets or returns the query string part of a URL.

Eg:

```
document.write('${window.location.host}'); <br>
document.write('${window.location.hostname}'); <br>
    "    "   ('${window.location.href}'); <br>
    "    "   ('${window.location.Pathname}'); <br>
    "    "   ('${window.location.search}'); <br>
    "    "   ('${window.location.protocal}'); <br>
    "    "   ('${window..location.port} <br>');
```

30/01/201.
wednesday

DOM - (10)

- Document Object Model.

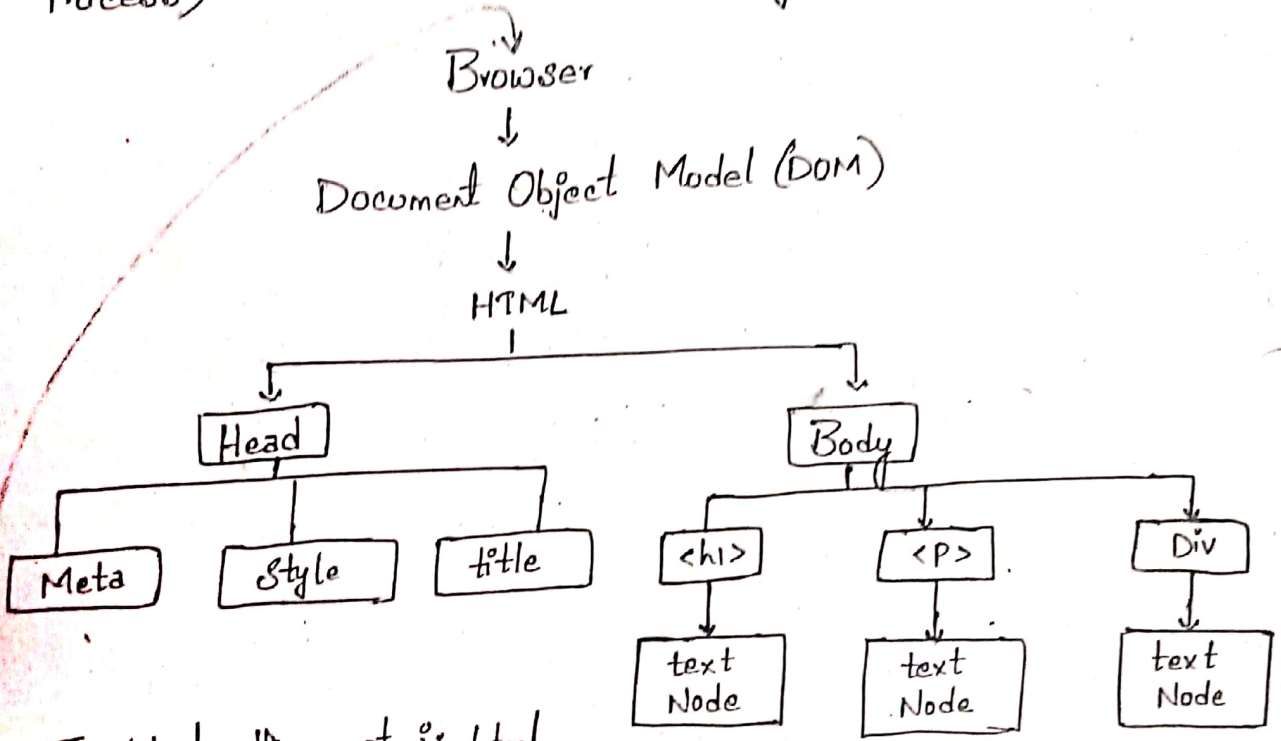
1.] When web page is loaded, the browser creates a document Object Model (DOM) of the page.

2.] The W3C Document Object Model is a platform & language neutral Interface that allows programs & scripts to dynamically access and update the content, structure and style of document.

3] HTML converts to Objects. It is only for Browsers.

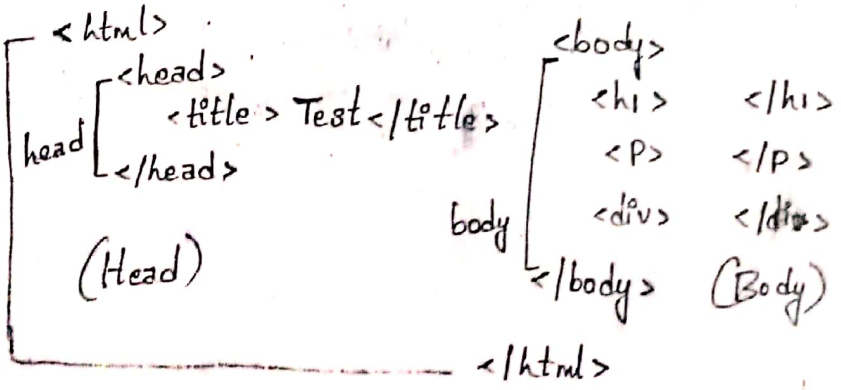
4] HTML code DOM model is constructed as a tree Objects.

5] When HTML code enters into chrome/Browser (at compilation Process) then browser automatically creates DOM.



⇒ For html, the root is html.

⇒ For DOM, the root is document.



The HTML DOM model is constructed as a tree of Objects:

1. JavaScript can change all the HTML elements in the page.
2. JavaScript can change all the HTML Attributes in the page.
3. JavaScript can change all the CSS styles in the page.
4. JavaScript can remove existing HTML elements and attributes.
5. JavaScript can add new HTML elements and attributes.
6. JavaScript can react to all existing HTML events in the page.
7. JavaScript can create new HTML events in the page.

To check how many forms, links, action, document Using DOM:

Eg: <html body:

```
<body>
    <div class = "container">
        <a href = " ">test</a>
        <a href = " ">test</a>
        <form action = " " method = "get">
            <input type = "text">
        </form>
        <form action = "index.java" method = "get" id = "second">
            <input type = "text">
        </form>
    <div id = "result"></div>
    <button>Submit</button>
</div>
<script src = "./document.js"></script>
</body>
```

document.js :

```
var test = document; (total document displayed.)
test = document.all; (will give total elements present in html code.)
test = document.head; (will give head) Elements in head
test = document.scripts; (will get all the scripts) Elements in scripts
test = document.links; (gives how many links are present) in links
test = document.forms [1].action; [URL of doc. with file name / name given for action.
test = document.forms [1].method; [gives the mtd used in form.]
                    index no.'s
console.log(test);
test = document.charset;
test = document.contentType;
test = document.body;
test = document.bgcolor="red";
test = document.bgColor = "hello";
console.log(test);
```

1. JS body: If we are having "id" without class:

html body:

```
<body>
    <div class = "container">
        <ul id = "ul">
            <li id = "list1"></li>
            <li id = "list2"></li>
            <li id = "list3"></li>
            <li id = "list4"></li>
        </ul>
    </div>
    <script src = "document.js"></script>
</body>
</html>
```

Js body:

```
var test = document.body.className = "Chandu";
test = document.body.classList.add("kavya");
document.body.style.background = "red";
console.log(test);
Var x = document.getElementById("ul");
    x.classList.add("test"); // Adding & giving class.
    x.SetAttribute("name", "kavya"); // Adding Attribute
    x.style.color = "red";
    x.style.padding = "10px";
    x.style.background = "#111";
    x.innerHTML = "test";
    x.getAttribute("id"); // calling that Attribute.
    console.log(x);
```

will combine existing one with new one.

→ will give color to body.

[replace; overriding the existing name]

for class adding
- → className
- → classlist.add()

for Id adding / values adding
- → attributes
  - → setAttribute()
  - → getAttribute().

O/p: we get the o/p in elements we get o/p html code with class name & id.

2nd prog: If we don't have id, we can use tagname to add class, id, we have to convert and then we have add by classList.

.js code:

```
Var list = document.getElementByTagName("li");
Var x = Array.from(list);
```

```
x. forEach (function (i))
{
  i. classList.add ("li");
});
```
} for each method (or) for each loop.

x. forEach (i ⇒ i. classList. add ("li")); → for each method by flat arrow method.

for traditional no need to convert into array.

```
for (var i=0; i<= list. length; i++)
{
  list[i]. classb.add ("li");
```
} traditional for loop

.html code :

```
<div class = "container">
  <ul>
    <li> list 1 </li>
    <li> list 2 </li>
    <li> list 3 </li>
    <li> list 4 </li>
  </ul>
</div>
```

o/p : we will get o/p with class

3rd prog : if we are having "class" without id.

```
var list = document. getElementsByClassName ("list");
```

.js code :

```
list[0] . style. color = "red"
```
→ This gives only for particular line.

```
Var x = Array . from (list);
  x. forEach (function (i))
  {
    i. style. color = "red"
```
} This gives entire form.

· html :

```
<div class = "container">
<ul class = "list">
  <li> test1 </li>
  <li> test2 </li>
  <li> test3 </li>
  <li> test4 </li>
</ul>
```

o/p: html code with class names.]

By using CSS selectors: [Advance way to give className, Id name & tag Names without DOM]

getElementsById
setElementsById } if we have id name.

getElementsByClassName
setElementsByClassName } if we have classname.

getElementsByTagName
setElementsByTagName } if we don't have classnames & id's.

** Without using above methods, we have advanced way to get the data by using QuerySelectors.

```
Var ul = document.querySelector("#ul");
Var x = document.querySelectorAll("li:nth_child(3)");
x.forEach(function (e) {
e.style.color = "red";
});
console.log(x);
```

child number.

⇒ JQuery selectors will give "Nodelist". ∴ No need to convert.

* we use jQuery Selectors / CSS Selectors.

we have already 3 selectors in CSS i) Class
ii) id
iii) Tag Name.

without CSS selectors (Class, Id, TagName), we use queryselectors.

queySelector (" ") → '#' '.' 'TagName'

queySelectorAll (" ");

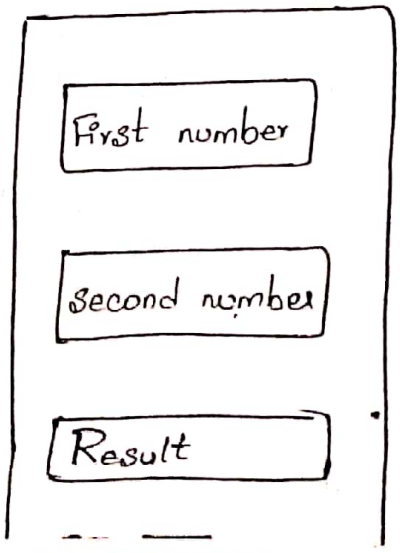
Ex: queySelector ('#ul');
querySelector (".ul");
queySelector ("ul");

* In query Selector, through id name, it gives entire list names. But in case of class name, it will give the name only to the first element of the list. it will not give the entire list.

* In queiySelectorAll, it will give "nodelist". Therefore, there is no need of converting html collection to array because it is nodelist.

* In query SelectorAll, it accepts all and it gives the entire list.

Project:



Program :

```
<div class = "container">
    <section id = "loginblock">
        <form action = " " method = "get">
            <div class = "form-control">
                <input type = "text" placeholder = "name">
            </div>
            <div class = "form-control">
                <input type = "text" placeholder = "email">
```

Program :

html code :

```
<div class = "container">
    <section id = "loginblock">
        <form action = " " method = "get">
            <div class = "form-group">
                <input type = "text" placeholder = "name">
            </div>
            <div class = "form-group">
                <input type = "text" placeholder = "email">
            </div>
            <div class = "form-group">
                <input type = "submit" "type = "submit">
            </div>
        </form>
    </section>
</div>
```

// html codes without class names.
class names are given by js code. //

TS code :

```
var form-group = document.querySelectorAll("#loginblock div");
form-group.forEach(function (e) {
    e.classList.add("form-group");
});
var form control.forEach(e => {
e.classlist.add("form-control");
});
var submit = document.querySelectorAll("input:nth-child(1)");
```

first (or) Last (or)

```
Submit[2].classlist.add("btn", "btn-success");
Submit[2].classlist.remove('form-control');
var loginblock = document.querySelector("#loginblock")
loginblock.classlist.add('col-md-4', 'col-md-offset-4');
```

Creating an Element :

TS code :

```
var ul = document.createElement("ul"); creating 'ul' Element.
ul.classList.add("list-group");
        add list-group class. (class="list-group" for ul
var li = document.createElement("li");
        it will be creating 'li' Elements. [<li> </li>]
li.innerText = "hello";
        add element i.e; <li> hello </li>
li.classList.add("list-group item");
        add "list-group item" for 'li' element as class.
```

ul.appendChild(li);

it will add child 'li' for ul ⇒

.

console.log(ul);

console.log(li);

//

Array Methods

Var array = ["Python", "Java", "Node JS", "Angular", "React JS"]
0 1 2 3 4

Different methods:

1] array.pop(): By default, Pop() method will delete the last index value in the array.

Eg: Var array = ["Chandu", "Lalli", "Sahi", "Siri"]
0 1 2 3

Var test = array.pop();

O/p: Chandu, Lalli, Sahi

2] array.shift(): This will delete the first index value in the array.

Eg: Var array = ["Chandu", "lalli", "Sahi", "Siri"]

Var test = array.shift();

O/p: Lalli, Sahi, Siri.

3] array.push(): Push() by default add the value at last.

Eg: Var test = array.push("manisha");

O/p: Chandu, lalli, Sahi, Siri, Manisha.

4] array.unshift(): It will add new value at first / begining of array.

Eg: Var test = array.unshift("rupa");

O/p: rupa, Chandu, lalli, Sahi, Siri. Manisha.

5. array.splice(): This method is used to delete and add the elements in b/w the array. It has 3 parameters.

Syntax:

variableName.splice(Position frm where to start, No. of elements to delete, which you want to add)

Ex: array.splice(2, 1, "abc") → add this in that place.

- 2 → starts frm 2nd position index
- 1 → one element to delete
- "abc" → add this in that place.

Var x = [a, b, c, d, e, f];
0 1 2 3 4 5

→ x.splice(2, 1, G); → o/p :- [a, b, G, d, e, f]. (at)

→ x.splice(3, 2); → o/p :- [a, b, c, f]

→ x.splice(2, 0); → o/p :- [a, b, c, d, e, f].
zero elements to delete.

6. sort(): This is used to sort the strings.

```
var test = array.sort();
```
→ To sort things.

* In order to sort the numbers, we have to use below method.

This is used to sort the numbers according to order.

```
array.sort(function(a, b) {
return a-b; (ascending order)
    (or)
return b-a; (descending order)
});
```

7] Reduce method : Reduce method will add all the Elements of array. to reduce into single value.

```
Var test = num.reduce (function (a, b) {

    Console.log(a);
    Console.log(b);
    return a+b;
});
```

8] fill method : It will fill array with static value.

```
Var test = array.fill("Chandu");
```

Ex: ["Java", "c", "nodejs", "c++", "Php"]

O/p : "Chandu", "Chandu", "Chandu", "Chandu", "Chandu".

9] index of(): This method is used to know the index of that element.

Ex: Var array ["java", "python", "angular", "node.js"];

```
var test = array.indexof("angular")
```

Element name.

10] join() : It is used to join the array elements with any special symbol.

```
var test = array.join("  ");
```

→ it is converted into string gives the o/p in string form

Ex: Var arr = [a, b, c]

```
var test = arr.join(" + ");
console.log(test);
```

31 Thursday

11] forEach(): It is an array method to iterate the values.

```
var array = ["java", "python", "nodejs"]
var test = array.forEach (function (item) {
    console.log(item);
});
var test1 = array.forEach (item => console.log (item));
    console.log(test1);
    console.log (test2);
```

01/02/2019 Friday

12] map(): The map() method creates a new array with the result of calling a function for every array element.

⇒ The map() calls the provided function once for each element in an array.

Note: 1). The map() does not change the original array.

2). The map() doesnot execute the function for the array elements without value.

```
var test = array.map (function (item) {
    return item;
});
```

Eg:

```
var array = ["java", "python", "nodejs", "angularjs"];
var num = [1, 40, 20, 50, 90, 19];
var test = array.map (function (item) {
    return item;
});
var test1 = num.map ((item => {
    return item *10;
});
```

```
document.write(test1 + "<br>");
document.write(test1);
```

O/P: 10, 400, 200, 500, 200, 190.

java, python, nodejs, angularjs.

13] filter(): A filter() method creates an array filled with all array Elements that pass a text provided as a function.

Note: 1) filter() is not execute the function for array elements without values.

2) filter() doesnot change original way.

3) To remove the duplicates in array.

Eg:
```
var array = ["java", "python", "nodejs", "ruby", "nodejs"];
var test = array.filter(function (value, index, array) {
    return array.indexOf(value) == index;
});
document.write(test1 + "<br>");
var test1 = array.filter(function (x) {
    return x > 20;
});
document.write(test1);
```
(x = age)

14] includes(): 1) The includes() determines whether a string contains the characters or not. of a specified string.

2) This method returns the boolean value. ie; true or false.

3) It checks whether the ~~string~~ array contains the element or not.

Ex: var num = [20, 25, 30, 28, 34, 58, 89];

var test = num.includes(40);

document.write(test);

O/p: false.

Note: . includes() is case sensitive.

var array = ["java", "python", "nodejs", "nby", "angular"];

var num = [1, 40, 90, 50, 20, 19, 100];

```
var test = num.includes(40);
var test1 = array.includes("Java");
```

document.write(test1 + "
");

document.write(test);

O/p: false (case sensitive).

03/02/2019 Sunday

true.

15] find(): This method returns the value of the first element in the array where predicates is true and undefined otherwise.

Eg: var array = [10 (10>20 F), 11 (11>20 F), 21 (21>20 T), 500, 40, 29, 50, 51, 1000];

```
var test = array.find(function(a) {
    return a > 20;
});
document.write(test);
```

O/p: 21. (So it prints 21.)

Note: 1] find() methods returns the value of first element in an array that pass a test (Provided as a function)

2.] The find() method executes the function once for each element present in the array.

3.] If it finds an array element where the function returns a true value, find() returns the value of that array element. (and does not check remaining otherwise it returns undefined.)

4.] * * find() does not execute the function for empty arrays.
* * find() does not change the original array.

16.] Reverse(): The Reverse method reverses an array in place. The first array element becomes the last, and the last array element becomes the first.

Eg: var array1 = ["nodejs", "python", "ruby", "angular"];

```
var test = array1.reverse();
```

document.write(test);

Prog:
```
var array1 = ["nodejs", "python", "ruby", "angular"];
var str = "hello nodejs";
<!... for (var i=0; i<array1.length; i++) {
    console.log(array1[i]);
    } ---> [traditional for loop].
```

But we can have another alternative to use this by using "for ... of" loop. This is used in ECMA Script 6.

**10:

17] **for-of loop**: The for...of statement creates a loop iterating over iterable objects, including: built-in string, Array, Array-like Objects. [Eg: (arguments or Node list)] Typed Array, Map are user defined iterables. It invokes a custom iteration hook with statements. to be executed for the value of each distinct property of Object.

Syntax:

```
for (let : of array1)
{
  document.write(i);
};
```

Eg:

```
var array1 = ["nodejs", "python", "ruby", "angular"];
var str    = "hello nodejs";

for (let i of array1) {
document.write(i);
};
for (let x of str) {
 document.write(x);
};
```

O/p: nodejs, python, ruby, angular
hello node js.

(i + "
") → nodejs
python
ruby
angular

(x + "
") → h e l l o n o d e j s

18] **for-in loop**: The for...in loop is used to loop through an object's properties. [This is not ES6, this is javascript]. As we have not discussed objects yet, you may not feel comfortable with this loop.

In each iteration, one property from object is assigned

to variable name and this loop. continues till all the properties of the object are exhausted.

Syntax: The Syntax of "for...in" loop is

```
for (variablename in object) {
    statement or block to execute;
};
```

Eg:

String Methods:

1.] length ( ) : discussed earlier

2.] indexOf( ): To get indexOf first occurance of a specified text.

3.] lastIndexOf( ): It returns indexOf the last occurance of specified text in string.

Eg: var str = "hello nodejs and we love nodejs";
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

```
Console.log(str.indexOf('n'));
Console.log(str.lastIndexOf('n'));
```

o/p: 6
25.

4.] Search( ): The search() method searches a string for a specified value and returns the position of the match.

Eg: var str = "hello nodejs and we love nodejs";
var test = str.search ("nodejs"); [First priority is given to the first occurance]
~~var test~~ Console.log(test);

o/p: 6.

Extracting - String Characters:

var str = "Hello_Angulas";
0 1 2 3 4

str.subString (S 4, E 9); o/p: o, A [Automatic swap is done in this. It accepts only +ve's numbers] (1, 0)

(0, 1); o/p: Hello

string methods

str.slice (S 0, E -1) (It accepts -ve nums, swap is not possible.)

str.substr (s c) (starts and count / (length) is taken)
(from)

array slice( ): for extracting values but not characters.

5.] SubString(): Returns the sub string at the specified location within a string object.

Syntax: string.subString (start number, end number): String.

str.subString (4, 0) (automatic swap is done)

(0, 4).

6.] Slice(): Returns the subString. It extracts a part of a string and returns the extracted part in a new string.

Syntax: Slice (start number, end number);

Slice (0, 4).

Note: Swapping is not possible in Slice(). [Slice(4,0) X]

7.] substr(): Gets a substring begining at the specified location and having the specified length.

Syntax: substr (from starting point, count);

↓

To print how many characters after starting point.

Note:

1.] The Slice() method extracts a part of a string and returns the extracted part in a new string.

→ The method takes 2 parameters: the start position, and the end position (end not included).

2.] The subString() method is similar to Slice().

The difference is that subString() cannot accept -ve indexes.

3.] The substr() method is similar to slice().
The difference is that the second parameter specifies the length of the extracted part.

Difference b/w Slice & subString.

SubString	Slice
· ~~Both are for iterating~~	
* Substring() automatically swap the values.	* Swapping cannot be possible.
* Does not accept -ve values	* Accept -ve values.

⇒ Both are for iterating the values (or) data from the String/array.

8.] repeat(): Returns a string value that is made from count copies appended together. If count is 0, the empty string is returned.

count - no. of copies to append.

→ Will repeat entire String.

Syntax: repeat(10); repeat the string 10 times.

9.] replace(): `replace('nodejs', 'angularjs');`

Replaces text in string using a regular Expression (or) search string.

O/p: Hello angularjs, we love nodejs. (this is not changed)

Replaces text in string using a regular Expression (or) search string.

Search value - A String to search for

replace value - A String container.

Eg: replace ('nodejs', 'angular');

```
test = replace ('nodejs', 'angular');
document.write (`<div class = "container">
                  <h1> ${test} </h1>
                </div>`);
```

o/p: hello angularjs.
we love nodejs.

(or)

To replace, all values with given value we use Regular Expression

str.replace (/nodejs/g 'angularjs');
(global)

o/p: hello angular js, we love angularjs.

10] match() : The match() is an inbuilt function in javascript which is used to search a string for a match against any regular expression and if the match will found then this will return the match as an array.

Parameters: Here the parameter is "regExp" i.e; regular Expression which will compare with the given string.

Eg: Var str = 'hello nodejs and we love nodejs';

```
var test1 = str.match (/nodejs/g);
console.log(test1);
```

o/p: ["nodejs", "nodejs"]. (array)

* If not matches, o/p will be ⇒ null.

11] startsWith(): Returns true if the sequence of elements of search string converted to a string is the same as corresponding Elements of this Object (converted to string) starting at position, Otherwise returns false.

Eg: startsWith (searchString ; String, position ? ; number)

startsWith ('h') ⇒ true
startsWith ('o') ⇒ false.

12] EndsWith(): It will check at End of String. (same as of startsWith.)

Eg: EndsWith ("js") ⇒ true
endsWith ('Q') ⇒ false.

13] Split(): Splits a string into subStrings using the specified separator & return them as array. To convert String into array.

Eg: Str.Split(" "); ⇒ ['h','e','l','l','o','A','n','g','u','l','a','r']
Str.Split ("<space>"); ⇒ ["Hello", "Angular"]
Str.split ("<space><space>");

We don't have reverse() for Strings for Arrays reverse() is there.

∴ Convert String into array & reverse it.

Var Str: "hello world";
Var test = Str.Split(" ").reverse()

To convert given array into String.

Var test = str.split(" ").reverse().join(" ");

14] Strike(): It will strike Entire String.

15] toUpperCase(): Converts all characters of String into Uppercase.

16] toLowerCase(): Converts all characters of String into Lowercase.

17) trim(): It will remove spaces at starting of the given String.

Ex: Var str = " I am Chandu "
document.write(str.trim());

o/p:

18] link(): It will help to link one page with another page.

19] anchor(): It will also helps to link the pages. It was just like <a>.

20] bold(): It will improves the font-weight.

21] charAt(value): It will returns the character at the given index value.

Ex: Str.charAt(2); ⇒ gives character At '2 index'

22] CharCodeAt("h"): It will returns ASCII value of given character.

23] fontcolor("red"): It will change the text-color into given color.

24] includes("h"): It will add the characters to the String.

25] italics(): It will change the font-style into italics style.

04/02/2019 04/02/2019
Monday Monday

Date Object:

Date Object: To create data Object.

Syntax: `Var date = new Date();`

Date Methods :-

```
date.getDay();
date.getDate();
date.getFullYear();
date.getMilliseconds();
date.getMinutes();
date.getHours();
date.getMonth();
date.getTime();
date.getSeconds();
```

* Creates a javascript date instance that represents a single momemt in time.
* Date Objects use Unix Time Stamp, an integer value that is no. of milliseconds from 1st Jan 1970.

Creating clock by using above Date methods.

In JavaScript we have 2 time delay methods.

i) Set Timeout(): The setTimeout() method of the window or workerGlobalScope mixing (and successor to window.setTimeout()) sets a timer which executes a function or Specified piece of code once the timer expires.

Syntax Section:

```
var timeoutID = scope.setTimeout(function [, delay, param1,
                                          Param2, ...]);
var timeoutID = scope.setTimeout(function [, delay]);
var timeoutID = scope.setTimeout(code [, delay]);
```

function:

A function to be executed after timer expires.

delay optional:

The time in milliseconds (thousandths of a second), the timer should wait before the specified function (or) code is executed.

If this parameter is omitted, a value of 0 is used, meaning execute "immediately" or more accurately as soon as possible.

Note: The actual delay may be longer than intended.

→ Executes only one time

→ built in window method

→ Works in node.js also.

~~setInterval()~~ Eg:

```
setTimeout(function () {
    var d = new Date();
    var h = d.getHours();
    var m = d.getMinutes();
    var s = d.getSeconds();
    var x = document.getElementById("demo");
    x.innerHTML = "<h1>${h}:${m}:${s}</h1>";
}, 5000);
```

HTML

```
<div>demo</div>
```

O/p: After 5 sec ⇒ current time will display.

→ 8:14:35

ii) SetInterval(): The setInterval() method, offered on the window and worker interfaces, repeatedly calls a function or executes a code snippet, with a fixed time delay b/w each call.

→ It returns an interval ID which uniquely identifies the interval, so you can remove it later by calling clearInterval(). This method is defined by the window (or) Worker GlobalScope mixin.

→ It will executes multiple times with timedelay. This keeps on updating. and also can be used in node.js.

Ex: setInterval(function() {

.....as code as setTimeout

},1000);

O/p: It will update time for every 1sec continuously.

→ To stop this, we use clearInterval().

→ These methods are used at client side & Server side (i.e; both NodeJS & JS).

Syntax Selection:

Var intervalID = scope.setInterval(func, delay[, param1, param2,...]);

Var intervalID = scope.setInterval(code, delay);

function:

A function to be executed every delay milliseconds. The function is not passed any parameters and no return value is expected.

delay:

The time in milliseconds (thousandths of a second), the times should delay in b/w executions of the specified function or call. (code is executed)

clearInterval(): The clearInterval() of the window (or) Worker GlobalScope mixin cancels a timed repeating action which was Previously established by a call to setInterval().

Syntax section:

scope.clearInterval(intervalID)

intervalID:

The identifier of the repeated action you want to Cancel. This ID was returned by the corresponding call to setInterval().

Its worth nothing that the pool of ID's used by setInterval(). and setTimeout() are shared, which means you can technically use clearInterval() and clearTimeout. interchangeably.

Mouse & Keyboard Events

dblclick: The event occurs when the user double-clicks.

click: The event occurs when the user clicks on an element.

input: The event occurs when an element gets user i/p.

mousedown: The event occurs when the user presses a mouse button over an element Mouse event.

mouseup: The event occurs when a user releases a mouse button over an element.

mouseenter : The event occurs when the pointer is moved onto an element mouse event.

mouse leave : The event occurs when the pointer is moved out of an element.

mouse move : The event occurs when the pointer is moving while it is over an element.

mouse over : The event occurs when the pointer is moved onto an element.

mouse out : The event occurs when a user moves the mouse pointer out of an element.

KeyBoard Events

Keydown : The event occurs when the user is pressing a key.

Keypress : The event occurs when the user presses a key

Key up : The event occurs when the user releases a key.

⟹ addEventListener ( ): This is one & Eventhandler. It will help to handle the Events.

Ex: Var button = document.getElementById ("button");
button.addEventListener ('Click', function () {

}, 1000);

O/p : when we click on Submit, the timer runs.

html code :

```
<div class = "container">
    <div id = "demo"></div>
    <button class = "btn btn-success" style = "margin-top:
                10px"; id = "button"> Success </button>
</div>.
```

04/02/2019
Monday

Angular JS

1.] Angular JS is a front end framework to built (or) create an application in single page. [Single page Application (SPA)].

2.] Developed by Misky Havery and his team in 2009 from GOOGLE.

3.] The initial version of Angular JS is Angular JS 1.x.

4.] Later, the versions of it are Angular 2/4/5/6/7.

5.] Latest version of it is "Angular 7" (currently used version.)

6.] Presently, Google is maintaining Angular and made it open source.

7.] Angular 2 to 7 is written in Type Script.

8.] Angular JS written in Java Script and controlled by "MVC architecture." [Model view Controller] [Angular JS 1.x (2009 - 2015)]

9.] Angular JS is using this MVC architecture at its initial stage.

10.] "Angular 2" onwards, it is "component based architecture." (written in typeScript)

11.] Angular JS is used to avoid multiple http requests.

MCQ:

MVC: It is a software architecture, designed pattern.

→ Model & view architecture is followed in all languages and accepted by all browsers.

→ Angular JS uses MVC & MVW architecture.

→ CDN – Content Delivery Network.

Angular 1.7x ↳ extended - outdated.

NPM: → It is a container. all libraries are present in this container. All Node related

NPM [Node Package Manager]

JQuery
Bootstrap
Angular
React
Embar JS
Moment JS

TypeScript [src: www.typescriptlang.org]

1] TypeScript is a typed superset of JavaScript that compiles to "plain JavaScript." [Any browser, Any host, Any OS. Open Source]

Why? → Browser know only JavaScript, doesn't know TypeScript.

2] "TypeScript is JavaScript for application-scale development."

3] TypeScript is a strongly typed, Object oriented, compiled language.

4] It was designed by "Anders Hejlsberg" (designer of C#) at Microsoft.

5] TypeScript is both a language and a set of tools.

6] TypeScript is JavaScript plus (+) some additional features.
Object oriented feature + static typed.

** Q: How do we install TypeScript : [compulsory question.]
100% MCQ:

```
npm install -g typeScript
```
global (Entire computer)

```
npm -v
```
to check the version

To check TypeScript Version :

```
tsc -v ----> version.
```
type script compiler

For typeScript Extension : [.ts]

TypeScript is totally used for development (Back End)

Babel :

Babel is javaScript Compiler.

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ / typeScript code into a backwards compatible version of JavaScript in current and older ~~versions~~ browsers (or) Environment.

021 Tuesday

Working with TypeScript in command prompt:

TypeScript = JavaScript + Some other features. [Object Oriented feature + Static typed].

The main things, we should do is,

⇒ If there are only one (or) 2 files, we can use node js for compilation of typeScript.

⇒ When ts file is opened in browser, it results an error in the console "file is not executable".

* Now, cd file name [in which file the Angular JS should be applied]

To Compile ts file, first we have to convert into JS file. Since ts not supported by browser, we have convert ts → JS

cmd prompt → tsc file.ts.
filename

```
<body>
<script src="./chandu.ts"></script>
</body>
```

⇓ we will be getting Error

Since, ts file not supports browser

∴ we need to convert into JS file.

To convert ⇒

cmd prompt

tsc chandu.ts <Enter>
filename.

* cd.. → to go to previous folder

cd (Go to drive)

cd capAngular (file location & file name)

dir <Enter>

tsc filename

dir

rm chandu.js
→ to remove files.

ls →
to get no. of folders.

⇒ commands for compiling ts files.

```
tsc chandu.ts
    (or)  node.js chandu.ts
```

* TS without browser – nodejs.

tsc main.ts (filename) | node.js main.ts (filename)

* let str : any ;
  str : "chandu"
  str : 10
  str : true
  str : undefined
  str : null

str : string = "chandu"
let num : number = 10;
let bool : boolean = true;
let array : string [ ] : ["java", "nodejs", 1];
let array1 : any [ ] = ["java", 10, true];
let array2 : number [ ] = [10, 90, 30];

*enum :

enum Color {red, blue, yellow}

→ enums allow us to define a set of named constants.
→ Using enums can make it easier to document intent, (or) create a set of distinct cases.
→ TypeScript provides both numeric & String based enums.

* Numeric enums :

We'll first start off with numeric enums, which are probably more familiar if you are coming from other languages.

Eg: enum Color {red, blue, green}
let backgroundColor = color.red;

Instead of using functions in JS, In TS we use one Object-Oriented feature "Enum."

Enum make it Simple.

Eg: 1] Using Var : (Not blocked)

```
function test() {
    for (var i = 0; i< 10 ; i++) {
    console.log(i)
    };
    console.log('finally' + i);
}
test();
```

No Error is found here.
Since, Var keyword.

2] Using let : (Blocked)

```
function test() {
    for (let i=0; i<10; i++) {
      console.log(i)
    };
    console.log('finally' + i); // Error is found here.
}                                  Since, let keyword.
test();
```

i is valid only within this loop.

1.] Var : if we declare variable inside loops that can be used at anywhere within the function.
We can reinitialize in this case.

2.] let : If we declare variable inside loop that should be used inside that loop only. We can't use it outside loop.
We can't even reinitialize let.

Modules:

1] Starting with ECMA Script 2015, Java Script h
modules.

2] Type Script shares this concept.

3] Modules are executed within their own scope, no
scope; this means that variables, functions, clas
declared in a module are not visible outside the
unless they are explicitly exported using one of th
forms.

4] Conversely to consume a variable, function, class, int
etc. exported from a different module, it has to
imported using one of the import forms.

Internal Modules:

1] Type Script supports two modes in which you can apply mo
2] The first one is internal modules.
3] Type Script allows you to define modules within your
typeScript files.
4] The Syntax is very similar to namespaces in Java and .N
and looks like this:

```
Module MyInternalModule {
    export class MyClass { }
}.
```

5] When you compile your type Script files your modules are
converted into variables that nest as
ame space like Objects

Expression)

```
var MyInternalModule;
(function (MyInternalModule) {
    var MyClass = (function () {
        function MyClass() {
        }
        return MyClass;
    })();
    MyInternalModule.MyClass = MyClass;
})(MyInternalModule || (MyInternalModule = {}));
var x = new MyInternalModule.MyClass();
```

External Modules :

External modules can also be used when you build a program using typeScript.

The use for external modules is however quite different from internal modules.

While internal modules can be seen as namespaces for your code, external modules must be viewed as ... well modules.

```
export class MyClass {
}
export var x: MyClass instance = new MyClass();
```

Installation of Angular: To install Angular related stuff, we use Angular CLI. ← this will help to install all the angular related stuff into the folder.

→ Angular Command Line Interface (CLI)

The installation is done with the cmd container, npm (node package manager)

** The command used to install Angular in Command prompt.

```
npm install -g @angular/cli
```

(npm: node package manager; -g: global)

if we get error, npm cache clean --force.

to find the version, ng -v. [tools of angular]

(ng: angular)

Available commands (tools) in Angular:

add: Adds support for an external library to your project.

*build <b>: Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.

Config: Retrieves or sets Angular configuration values in the angular. json file for the workspace.

doc <d>: Opens the official Angular documentation <angular.io> in a browser, and searches for a given keyword.

e2e <e>: Builds and serves an Angular app, then runs end-to end tests using Protractor.

help: Lists available commands and their short description.

lint<l>: Runs linting tools on Angular app code in a given Project folder.

new:<n>: Create a new workspace and an initial Angular app.

run: Runs an Architect target with an optional custom builder configuration defined in your project.

serve <s>: Builds and serves your app, rebuilding on file changes.

test<t>: Runs unit tests in a project.

update: Updates your application and its dependencies.

See https://update.angular.io/

version <v>: Outputs Angular CLI version.

xi18n: Extracts i18n messages from source code.

lint (l): Runs linting tools on Angular App code in a given Project folder.

~~xi18n~~: ~~Extra~~ ng → Angular.

* To check the tools of angular

ng -v

* To check the angular project.

ng new Jspiders

* Sass – Syntactically awesome style sheet.

* Less –

* Styles -

Web Pack : Used to bundle all the packages, css files, html files etc.. into a single file. It is JavaScript.

ng serve : To create server and bundling all packages.

Web pack & Bable automatically compiles ts files to js files.

* In cmd prompt → "The file location (Path) code ." to open Vscode

* src
  - app — root component of angular.
  - assets — images
  - environments — development & Production.
  - browserlist — To put which browsers support project.
  - index.html — root html file in angular.

Conform MCQ

Decorator: Decorator that marks a class as an Angular component and provides configuration metadata that determines how the component should be processed, instantiated and used at run-time.

→ Components are the most basic UI building block of an Angular [illegible] tree of angular components.

Class decorators:

Decorators: There are 4 types of Decorators. They are

i] Class Decorators
ii] Property Decorators
iii] Method Decorators
iv] Parameter Decorators.

@ app.module.ts
⇓
@ Ngmodule.

** i] Class Decorators:

Eg: @ Component and @ NgModule.

1) Angular offers us a few class decorators These are the top-level decorators that we use to express intent for classes. They allow us to tell Angular that a Particular class is a component, or module. for example, the decorator allows us to define this intent without having to actually put any code inside the class.

2) It converts class into component making use of metadata.

ii] Property decorators :

used for properties inside class.

Ex: @Input and @Output.

iii] Method decorators :

used for methods inside classes.

Ex: @HostListener.

iv] Parameter decorators :

used for parameters inside class Constructors.

Ex: @inject:

import {component} from '@angular/core';

↓

We can get this by installing auto-import.

@Component ({ → to call Object.

Selector : 'chandu'

→ template : '<h1> Hello angular </h1>'

Inline

});

export class ChanduComponent {

* Go to cmd prompt to create component.

ng g c header → filename

↓ ↓

generate component

* Go to App

↓

Create new file (name.component.ts)

↓

Now, go to the file

↓

Create class name with export keyword

↓

@component (through auto import we get the import statement)

↓

Now add meta tags inside.

↓

Now copy the Selector.

To link Bootstrap in Angular :

View menu

↓

terminal

↓

npm install bootstap

↓ → src

node modules ↓

styles.css ← styles.css

↓

Bootstrap

↓ dist

↓

bootstrap.css

@import '~bootstrap/dist/bootstrap/css/bootstrap.css';

for html in root file.

i.e; index.html.

* In course component .ts

@ component {

<button class = "btn btn-primary"> Submit </button>

* To introduce interpolation in angular {{ }}

(or)

expression.

* In cmd,

rm.main.js — to forcefully remove the file from folder.

R.

Interpolation : A template expression produces a value and appears within the ({{ }}) double curly braces. Angular executes the expression and assigns it to a property of a binding target; the target could be an html element, a component

Ex: @component ({

chandu.components.ts.

`{{title}}`

class chandu Component {

title : String = "This is an angular application";

}.

MCQ:

Directories :

① ngFor.
② ngIf
③ ngSwitch

In course.component.ts':

```
<ul class = "list-group">
  <li class = "list-group-item mt-3"
      *ngFor = "let lang of language">
      {{lang}}
  </li>
</ul>
```

language : string[] = ["java", "angular", "python", "react"];

}

** ngFor is a structural directive for iterating array values.

*To call methods in angular.

@component ({

{{getTitle()}}

});

class courseComponent {

title : string;

getTitle() {

this.title = "hello nodejs";

06/02/21
Wednesday

MCQs.

Routing: The path which is given after the link of web page (/...) is called Routing.

https://localhost:4900/login (about, Contact...)
routing

Single page development:

Creating all pages in Single page.

Ex: Gmail (login, Compose, Sentbox, Outbox, Spam .. etc..)

Http Status: File success - 200-300
Page not found - Error 404.

Steps to create a Router:

Step 1: Initially, we have to create components as how many we want by using the command through cmd prompt.

command prompt - "ng g c login (component name)":
g → Generate
c → component
login → about, Contact
Page not found..

Step 2: Go to the code through "code ." and inorder to configure the created components, we have to go to parent module file [module.ts] and import router module to create a single page application.

i] Import RouterModule.

```
import { RouterModule } from '@angular/router';
```

Step 3: After go to import array and then we have to add RouterModule in that array.

```
RouterModule.forRoot([
    {object 1};
    {----};
])
```

In that RouterModule, we have 2 methods. They are:

i) forRoot

ii) forChild

In that methods, we have to create objects with the components we have created.

```
imports : [ RouterModule.forRoot([
    { Path : " ", component : HomeComponent }
    { Path : 'login', Component : LoginComponent }
    { Path : 'register', Component : RegisterComponent }.
    { Path : '**', Component : PageNotFoundComponent }.
])]
```

this is compulsory for all the pages inorder to give Page not found error.

* If user gave invalid data (routing) then it will give pagenotfound error.

We can create objects in another way by using variable.

* Create one variable with array and store all the objects in array.

then call that variable in a "RouterModule.forRoot()".

Eg: import {RouterModule, Routes} from '@angular/router';

varname
const Router : Routes = [
annotation

{Path : 'login', component : loginComponent},

{Path : 'register' component : RegisterComponent},

:

{Path : '**', component : Pagenotfound Component}

]

imports : [

RouterModule.forRoot(Router)

],

Step 4 : We have connect router with HTML.

Go to parent (root) html file.

<router-outlet></router-outlet> ← In HTML file.

connection b/w router & html

It is predefined by router for connecting b/w router & html.

Step 5 :

In order to make single page we have to link with bootstrap3 and html.

* In order to remove refreshing air for every component, we have to add router-link instead of href to avoid page refreshing.

* Create header component and write the "nav" code :

```
<nav class = "navbar navbar-expand-sm bg-success navbar-dark">

  <ul class = "navbar-nav">
    <li class = "nav-item">
      <a class = "nav-link" routerLink = "/about"> About </a>
                                                    Component name.
    </li>
    <li class = "nav-item">
      <a class = "nav-link" routerLink = "/login"> Login </a>
    </li>
    <li class = "nav-item">
      <a class = "nav-link" routerLink = "/register"> Register </a>
    </li>
    <li class = "nav-item">
      <a class = "nav-link" routerLink = "/Contact"> Contact </a>
    </li>
  </ul>
  <ul>
  Full class = "nav-item navbar-nav ml-auto navbar-right">
    <li class = "nav-item">
      <a class = "nav link routerLink = "/reg register"> Register </a>
    </li>
    <li class = "nav-item">
```

```
<a class="nav-link" routerLink="/contact">Contact</a>
</li>
</ul>
</nav>
```

to make these components move to right in navbar.

> In the "app.component.html"

.root html for app.component.

```
<app-header></app-header>
<router-outlet></router-outlet>
```

> In order to get page details in each component, we have to write html code in each component, html file seperately with credentials & data to enter the end user.

> To get middle, total file then we can write container class in root html.

```
<div class="container">
```

which ever components we want to get in the middle.

```
</div>
```

```
<form action = "#" class = "mt-4 col-md-4 offset-4">.
  <div class = "form-group">
    <label for = "username" > Username </label>
    <input type = "text" class = "form-control"
        id = "usevname" name = "username" required >..
  </div>
  <div class = "form-group">
    <label for = "password" > Password </label>
    <input type = "password" class = "form-control"
        id = "password" name = "password" required >
  </div>
</form>
```

* We have to create each component pages in each component files.

1] Angular Router:

1] The Angular router is a core part of the Angular platform.

2] It Enables developers to build single page Applications with multiple views and allow navigation between these views.

2] The Router-Outlet:

1] The Router-Outlet is a directive that's available from the router library. where the router inserts the component that gets matched based on the current browser's URL.

2] You can add multiple outlets in your Angular application which enables you to implement advanced routing scenarios.

<router-outlet></router-outlet>.

3] Routes and Paths:

1] Routes are definitions (objects) comprised from atleast a path and a component (or a redirect to path) attributes.

2] The path refers to the part of the URL that determines a unique view that should be displayed and component refers to the Angular component that needs to be associated with a path.

3] Based on a route definition that we provide (via a static RouterModule.forRoot(routes) method); the Router is able to navigate the user to a specific view.

Path:

1] The path can take a wildcard string (**).

2] The router will select this route if the requested URL doesn't match any paths for the defined routes.

3] This can be used for displaying a "Not Found" view or

redirecting to a specific view if no match is found.

Navigation Directive :

The Angular Router provides the routerLink directive. to create navigation links. This directive takes the path associated with the component to navigate to.

Eg: <a [routerLink] = "'/contacts'"> Contacts .

Bindings:

1] Property Binding:

1) Property Binding simply means you're passing data from the component class and setting the value of a given element in the view.

2) Property binding is also a one-way data binding, where we bind a property of a DOM element to a field which is a

3) Property we define in our component type Script code.

4) Behind the scene, Angular converts string interpolation into Property binding.

⇒ DOM & html are one to one maping but not all.

i.e; Some attributes in html are similar to properties of DOM.

Instead of using interpolation "{{ }}" we use binding concept since binding is faster.

The attributes in html are properties in DOM.

body
img

body
img

2] Attribute Binding : [attr. attribute]

html

Property binding allows you to define Element attribute values from the component class. It is one way data-binding. As you can only set data from the component to the view.

<img [attr.src] = "imgURL">.

Class Binding: [class.cssclassname]

* Class Binding & Ngclass in Angular 2. It's easy to bind css classes to elements in your Angular 7 templates.
* You provide a class name with class.className blw brackets in your templates and then an expression on the right that should evaluate to true or false to determine if the class should be applied.

Style Binding:

Style Binding & NgStyle in Angular 7. It's easy to bind inline style in your Angular 2 templates.

Event Binding:

Angular 2 Event binding Example:

Event binding is a data binding from an Element to a component.

Data obtained from user actions such as keystrokes, mouse movements, clicks and touches can be bound to Component property using Event binding.

Directives : Building blocks.

1] Structural Directive.

2] Attribute Directive

3] Component Directive.

1] Structural Directives : (*)

→ Structural Directives are a key part of Angular, everyone should be familiar with.

→ They are responsible for manipulating DOM through adding, removing or changing the elements.

→ Even if you have written a structural directive yourself, you have probably been using *ngIf and *ngFor in your templates, pretty often.

→ There are 3 types in it.

1] ngFor : for looping purpose [Iterating values].

2] ngIf : for making conditional purpose.

3] ngSwitch :

Manipulating the DOM :

· for a directive to be able to manipulate the DOM, it is necessary to provide a way to do it without actually changing something via DOMAPI. In angular it is done by injecting a view container reference and a template reference to the directive.

Eg:1]

<div *ngIf = "courses.length > 0" >

<h1> courses are here </h1>

<ul class = "list-group" >

<li class = "list-group-item" *ngFor = let i of courses" >

{{i}}

<div *ngIf = "courses.length == 0" >

<h1> no courses here </h1>

export class className

{

courses = ["java", "Python", "nodejs"],

}

For Angular 2

2] From Angular 4-7 :

We can use a new tag <ng-template> and a template reference variable (#). By using this way by creating a "if else" condition by the template reference variable.

Eg: <ng-template # coursesList>

<h1> courses are here </h1>

<ul class = "list-group" >

<li class = "list-group-item" *ngFor = "let i of courses" >

{{i}}

</ng-template>

<div *ngIf = "courses.length > 0 then courseList else noCourses" >

```
<ng-template #nocourses>
    <h1> No Courses here </h1>
</ng-template>
</div>.
```

In class, code is same as in before code.

3.] Using * ngSwitch :

```
<div [ngSwitch] = "color" class = "lead mt-4">
    <h1 *ngSwitchCase = "'red'"> Red Color is applying </h1>.
    <h1 *ngSwitchCase = "'green'"> Green Color is applying </h1>
    <h1 *ngSwitchCase = "'blue'"> Blue Color is applying </h1>
    <h1 *ngSwitchDefault> no color is here </h1>.
</div>
</div>.
```

In class, color = "red";
courses = ["java"];

2.] Attribute Directive : i) ngStyle
ii) ngClass.

→ An attribute Directive changes the appearance or behaviour of a DOM element.

→ Attribute directives, as the name goes, are applied as attributes to element.

→ They are used to manipulate the DOM in all kinds of different ways except creating or destroying them. I like to call them DOM-friendly directives.

→ Directives in this categories can help us achieve one of the following tasks:

To Apply conditional styles and classes to elements.

<P [style.color] = "'blue'"> Directives are Awesome </P>.

Eg: <P style = "Padding : 1rem" [ngStyle] = "alertStyles">

hello Chandu

</P>.

<button [ngClass] = "calculateClasses()"

(click) = "Submit()"> test Class

</button>.

In class:

~~submit() {~~

~~console.log('Button submitted');~~

export class CourseComponent {

stateFlag = true;

advanced way will explain it later

toggleState() {

this.stateFlag = !this.stateFlag;

}

Submit() {

console.log('Button submitted');

};

CalculateClasses() {

return {

btn : true,

'btn-danger' : true,

'btn-extra-class' : this.stateFlag

};

}.

~~Forms~~:

3] Component directives:

@component () {

section : 'course' →
↳ Component directive inside component.

}

Angular forms :

1. Template driven form
2. Reactive form. (or) Model-driven form.

1.] Template driven form : used with directives.

→ It can be used to develop simple applications.

Built in classes for Angular form :

Dirty
Pristive
touched
Untouched
invalid
valid

→ Angular provides form-specific directives that you can use to bind the form input data and the model. The form-specific directives add extra functionality and behaviour to a plain HTML form. The end result is that the template takes care of binding values with the model and form-validation

→ In this tutorial, we will be using template-driven forms to create the signup page of an application.

→ The form will cover the most common form elements and different validation checks on these form elements.

Here are the steps that you will follow in this tutorial.

- Add FormsModule to app.module.ts
- Create a class for the user model.
- Create initial components and layout for the signup force.
- Use Angular form directives like ngModel, ngModelGroup, and ngForm
- Add validation using built-in validators.
- Display validation errors meaningfully.
- Handle form submission using ngSubmit

ngModel:

We need the ngModel in the form input and the input must be named too:

<input type="text" ngModule name = "firstName">

→ There are cases where we need pass an event listener to the input field, or pass the value of the input to our component,

→ We need assign a template variable to the input do to that.

```
<input type = 'text" ngModel
        name = "first Name"
        #firstName = "ngModel"
```

(change) = "doSomething (firstName)".

→ ngModel is an instance of the FormControl which has quite a number of controls which had include dirty, invalid, errors Pristine, touched, untouched, value etc. The form control class is use to track the state changes of our input.

08/02/2019 Friday:

ngForm :

The ngform is an instance of the FormGroup. The FormGroup represents the group of

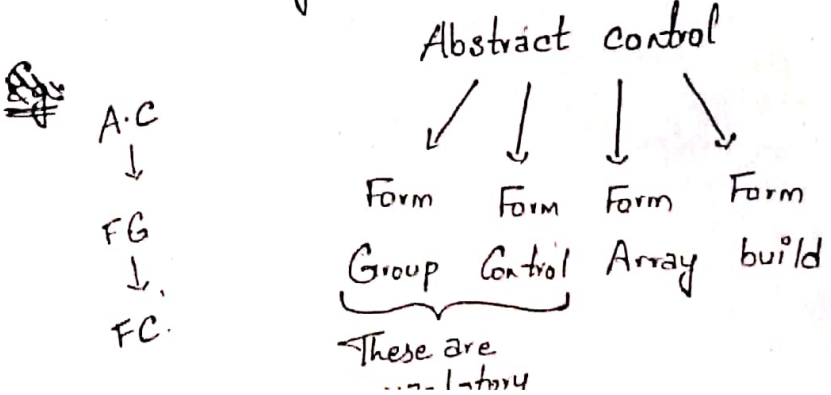
```
<form #f = "ngForm" (ngSubmit) = "submit (f)">
    <input type = "text" ngModel.
        name = "firstName"
        #firstName = "ngModel"
        (change) = "doSomething (firstName)"
</form>.
```

Eg: cd Desktop / CapAngular / Ispides → login.component.html.

2.] Reactive form (or) Model driven form:

→ Validators provides the min-length, required, max, name, Patterns.

→ If for multiple values - array is used.
Single values - validators. filename.



→ The process we will follow for we will written logic inside html in the template driven form. But in reactive form, logic will be written inside the form.

Eg:

```
export class loginComponent implements onInit {

  form = new FormGroup ({

    username : new FormControl ("", [

      validators.required, validators.minlength (3)

    ]);

  Password : :
```

Form Group : Tracks the value and validity state of group of formControl instances.

→ A form Group aggregates the values of each child Form Control into one object, with each control name as the key It calculates its status by reducing the status values of its children. For Example, if one of the controls in a group is invalid, the entire group becomes invalid.

→ FormGroup is one of the three fundamental building blocks used to define forms in Angular, along with FormControl and FormArray.

→ When installing a formGroup pass in a collection of child controls as the first argument. The key for each child registers the name for the control.

Form Control : - Tracks the value and validation status of an individual form control. This is one of the three fundamental building blocks of Angular forms, along with formGroup and formArray.

→ It extends the Abstract Control class that implements most of the base functionality for accessing the value, validation status, user interactions and events.

Ex:
```
<form action="" class="mt-4 col-md-4 offset-4" [formGroup]="form">
  <div class="form-group">
    <div class="alert alert-danger mt-2"
     *ngIf="username.touched && username.invalid">
     <div *ngIf="username.errors.required">username req</div>
      <div *ngIf="username.errors.minlength">min-length-3</div>
    </div>
    <label for="username">Username</label>
    <input formControlName="username" type="text" class="form-con
     name="username" id="username">
  </div>
  <div>
    <div class="form-group">
      <label for="password">Password</label>
      <input type="password" class="form-control" name="passw
       id="password">
```

Note: whenever we are using reactive form we have to import that in import array → ReactiveFormsModule.

* What are Reactive forms or Model Driven form :

→ Reactive forms a kind of the opposite to template-driven forms. Instead of defining the form in your template, the structure of the form is defined in code.

→ This approach uses the reactive form of developing the forms that favour the explicit management of data between the UI and the model.

→ With this approach, we create a tree of Angular form controls and bind them in the native form controls. As we can create the forms control directly in the component, it makes it a bit easier to push the data between the data ~~models~~ between the data models and the UI elements.

Form Control:

It tracks the value of the controls and validation of the individual control in the form.

Form Group:

Tracks the validity and state of the FormControl instance or, moreover, we can make FormGroup a collection of FormControl.

Form Builder:

This helps us to develop the form along with its initial value and validators.

Custom Validation:

Create a new file in login (or) register.

↓

username.validators.ts

↓

export class usernameValidators {

↓

create a property

↓

to make it available outside class declare using Access Specifier (static)

Create a annotations : Abstract control
↓ ↓
validation errors base class

Using Validators :

→ Angular provides us many useful validators, including required, minLength, maxLength and pattern. These validators are Part of the validators class, which comes with the @angular/forms package.

Custom Validators :

→ Angular, creating a custom validator is as simple as creating another class. They only thing you need to keep in mind is that it takes one input parameter of type AbstractControl and it returns an object of a key-value pair if the validation fails.

→ The type of the first parameter is Abstract control, becaus it is a base class of form control, formArray and formGroup and it allows you to read the value of control Passed to the custom validation function.

The custom validator returns either of the following :

* If the validation fails, it returns an object, which contains a key-value pair. If the validation does not fails, it returns null.

CannotContainSpace (x : AbstractControl) : ValidationErrors | null { }

Custom directive : (test)

→ You can define your own directive.

Create tcustom directive.ts

↓

@Directive decorator

↓

Structural directive.

Create a custom directive.

Eg: import { Directive, ElementRef } from '@angular/core';

@Directive ({

Selector : "[test]"

})

export class testDirective {

constructor (e1 : ElementRef) { Provides custom element apply style, add classes

e1.nativeElement.style.color = "red"

↓

native DOM element.

Custom Structural Directive :

For writing a custom Structural directive :

→ Import the Directive decorator (instead of the Component decorator).

→ Import the directive, ElementRef; you'll need them for any Structural directive.

→ Apply the decorator to the directive class.

→ Set the CSS attribute selector that identifies the directive when applied to an element ElementRef } from '@angular/core';

```
@Directive ({
    selector : '[myCustom]'
})
export class myCustomDirective { }
```

→ The directive's selector is typically the directive's attribute name in square brackets, [myCustom]. The directive attribute name should be spelled in lower CamelCase and begin with a prefix. Don't use ng. ← that prefix only belongs to Angular. The prefix belongs to Angular

The directive class name ends in Directive.

1. MCQ:

Hostlistner :

By using Hostlistner directive no need of calling in html. It automatically calls.

```
@Hostlistener ('click') Click() {
    alert ("Hello");
}
@Hostlistener ('mouseenter') mouseenter() {
    console.log ("mouse entered");
}
@Hostlistener ('mouseleave') mouseleave() {
    console.log ("mouse leaved");
}
}
```

automatically called in html by Hostlistener.

Method Decorators are very similar to property decorators but are used for methods instead. This let's us decorate specific methods within our class with functionality. A good example of this is @ HostListener.

This allows us to tell Angular that when an event on our host happens, we want the decorated method to be called with event.

@ HostListener :

Binds a css event to a host listener and supplies configuration metadata. Angular invokes the supplied handler method when the host element emits the specified event, and updates the bound element with the result.
If the handler method returns false, applies prevent default on the bound element.

@ HostBinding : (binds DOM elements)

Decorators that marks a DOM property as a host-binding property and supplies configuration metadata. Angular automatically checks host property bindings during change detection, and if a binding changes it update, the host element of the directive.

Ex: @HostBinding ('class btn-info') btn = false;

```
constructor (el : ElementRef) {
el.nativeElement.style.color = "red";
}
```

```
@HostListener('mouseenter') mouseenter() {
    console.log("mouse entered");
    this.btn = true;
@HostListener('mouseleave') mouseleave() {
    console.log("mouse leave");
    this.btn- false;
```

Pipes : used for filtering purpose to make it human readable.

("|") symbol is used represent the pipes, read as pipe operator.

Eg:

```
@component ({
    selector : 'course',
    template : `
    <div class = "container">
    <button class = "mt-4" test >Submit</button>
    <h1 id = "h1"> {{users.name | uppercase | lowercase}}</h
    <h1> {{users.age}} </h1>
    <h1> {{users.company | titlecase }} </h1>
    <h1> {{users.ratings | number }} </h1>
    <h1> {{ users.salary | currency : 'INR' : true}} </h1>
    <h1> {{ users.doj | date : shortDate }} </h1>
    </div>
})
export class courseComponent {
    users = { name : "siri",
              age : 21,
              company : 'capgemini',
              salary : 50,000,
              doj : new Date ('15/3/2019'),
```

Built in Pipes :

decimal
number
uppercase
lowercase
title case
currency
date
shortdate
long date
Percentage.

19/09/2019
Saturday

Creating Custom Pipes :

angular.io
↓
custom.pipe.ts
↓ → create cls implement pipeTransform
copy pipeTransform code ↓ into
custom.pipe.ts
↓

To use above code we have to create a decorator @ Pipe ( )

```
import {PipeTransform, Pipe
    from '@angular/core'
@ pipe ( ) {.
```

* By using CLI,

`ng g c component name` → generate Component

→ to directly create a custom component from cmd prompt.

`ng g d directive name` → generate directive

→ to directly create a custom directive from cmd prompt

`ng g P Pipe name` → generate Pipe

→ to directly create a custom Pipe from cmd prompt.

Pipe Transform: To create a pipe, you must implement this interface.

* Angular involves the transform method with a value of a binding as the first argument, and any parameters as 2nd argument in list form.

* @ pipe decorator: Declares a reusable pipe function and supplies configuration metadata.

* API is in JSON format.

Service:

Create courses.service.ts

↓

**whenever using service we have to use constructor. [mandatory]

In courses.service.ts

↓

Eg:
```
export class courseService
{
  getCourses ({
    courses = ["java", "Python", "ruby", "JS"];
  })
```

↓

Create a component about (or) with any other name

↓

In 'about.Component.ts'

Eg: class -- {

course

constructor() {

Not Unit testable so not followed this.

let service = new courseService();

this.course = service.getcourse();

(or)

By using dependency injection in angular.

Create a Constructor

↓

Import course service in about.component.ts

Eg: constructor(service.courseService) {

Standard way This is followed

this.courses = service.getCourses();

}

ngOnInit() {

}

}

→ Whenever using dependency inject have to register in providers[] in app.module.ts

Service cls

Eg: Cours service

Angular 2:

In Angular 5: (Other way)

→ Use decorator @Injectable so that we don't need to register in providers [ ] in app.module.ts.

* By using CLI,

`ng g c component name` → Generate Component

→ to directly create a custom component from cmd prompt.

`ng g d directive name` → generate directive

→ to directly create a custom directive from cmd prompt

`ng g P Pipe name` → Generate Pipe

→ to directly create a custom Pipe from cmd prompt.

Pipe Transform: To create a pipe, you must implement this interface.

* Angular involves the transform method with a value of a binding as the first argument, and any parameters as 2nd argument in list form.

* @pipe decorator: Declares a reusable pipe function and supplies configuration metadata.

* API is in JSON format.

Service:

Create courses.service.ts

↓

**whenever using service we have to use constructor. [mandatory]

In courses.service.ts

↓

Eg:
```
export class courseService
{
  getCourses(){
    courses = ["java", "python", "ruby", "JS"];
  }
}
```

↓

Create a component about (or) with any other name

↓

→ Our components and services are classes.

→ Every class has a special function called Constructor which is called when we want to create an Object (instance) of that class to be used in our application.

→ The old way of doing DI in Angular to make this fine piece of engineering work:

→ Angular has to aware of every single entity that we want to inject into our components and services.

→ Before the release of Angular 6, the only way to do that, was to specify services in the providers : [ ] Property of the @NgModule decorator.

[or @Component / @Directive but more on that later...].

Http Client

→ Http client is a RESTfull API
(Representational Stateless Transfer.)

→ SOAP API ← outdated used for XML

→ RESTfull API ← uses JSON

→ To connects API's, modify API's, Delete API's we usek the http methods.

→ To get API, we make use of Angular, JQuery, React, Ember..
Main use of http client is to make use of RESTfull API.

→ "JSONPlaceholder.typicode.com" website provides a platform to create fake API's.

Fake online REST API for testing & prototyping service.

- https://jsonplaceholder.typicode.com/posts.

Open terminal

↓

ng g c http

↓

Go to app.module.ts

↓

const Router : Routes [

{Path : 'http', Component : HttpComponent}

]

↓

header.component.html

↓

http.component.ts (built in service module)

import {HttpClient} from @angular/common/http

In class

users: any

Constructor(http : HttpClient) {

↳perform http requests.

http.get('https://jsonplaceholder.typicode.com/posts')

Put method

Post method

Get method

Delete method

subscribe((res)⇒

↳If valid Url coming through observable object.

(console.log(res));

HttpsClientModule ⟶ dependancy take care by this module.

∴ Register this module inside app.module.ts

Inside http.ts file.

⇓

```
Class -- {
  Posts : any ;
    Constructor (http : HttpClient) {
http.get ('https://api.github.com/users')
      Subscribe (user ⇒ {
        this. users = user ;
    })
     }.
```

```
<ul>
  <li. *ngFor="let post of posts"; index as i>
        .{{post}}
  </li>
</ul>
```

login / title
↳ These r taken from the link

* https://api.github.com/users. ←

* Open header.html & Link

```
<li class = "var-item">
< a router-link
```

JSON object to string
↓
JSON.stringify( ).

Create RestAPI - Post Request :

In http.component.ts :

```
Posts = ( );
url: sting . 'https://jsonplaceholder.typicode.com/
```

```
constructor (public http: HttpClient) {
http.get<any>(this.url).subscribe(Post => {
    this.posts = post;
})
};

createPost (input: HTMLInputElement) {
  let Post = { title : input.value }
  this.http.post<any>(this.url, JSON.stringify(Post))
                .subscribe(res => {
        Post.id = res.id;
        console.log(res);
        this.posts.splice(0, 0, post);
})
}
```

In http.component.html:

```
<ul class = "list-group">
  <li class = "list_group_item"
  *ngFor = "let post of posts; index as i">.
```

HttpClient:

→ In angular 4.3, the new http client module has been introduced. This new module is available in package @angular/common/http and a complete re-implementation of former http module and can be used to initiate.

→ Http request and process responses within your application.

→ To be able to use the HttpClient service within your components we first need to include HttpClientModule in the angular application. First we need to import HttpClient Module in the application's root module in file app.module.ts.

→ HttpClient will use the XML Http Request browser API to execute Http request.

→ Inorder to execute Http request of a specific type you can use the following methods which corresponds to Http verbs:

get
post
Put
delete
Patch
head
json

Observables: Observables provide support for passing messages b/w publishers and subscribers in your application.

→ Observables offer significant benefits over other techniques

for event handling, asynchronous programming and handling multiple values.

→ Observables are declarative that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes (or) until they unsubscribe.

Life Cycle hook purpose and timings:

- **ngOnChanges():**

  Responds when angular (re)sets data.

→ Ground input properties.

→ The method receives a single changes object of current and previous property values.

→ Called before ngOnIt() and whenever one or more data bound input properties change.

- **ngOnIt():**

  Initialize the directive/component after angular first displays the data-bound properties and sets the directive/component's input properties.

→ Called once, after the first ngON changes().

• ngDoCheck():

Detect and act upon changes that Angular can not or won't detect on its own.

→ Called during every change detection run, immediately after ngOnChanges() and ngONInit().

• ngAfterContentInit():

Respond after angular projects external content into the component's view / the view that a directive is in.

→ Called once after the first ngDoCheck().

• ngAfterContentChecked():

Respond after angular checks the content projected into the directive /component.

→ Called after the ngAfterContentInit() and every subsequent ngDoCheck().

• ngAfterViewInit():

Respond after angular initializes the component's views and child views/ the view that a directive is in.

→ Called once after the first ngAfterContentChecked().

• ngAfterViewChecked():

Respond after angular checks the components views and child views/ the view that a directive is in.

→ Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().

for event handling, asynchronous programming and handling multiple values.

→ Observables are declarative that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes (or) until they unsubscribe.

Life Cycle hook purpose and timings:

- **ngOnChanges():**

  Responds when angular (re) sets data.

→ Ground input properties.

→ The method receives a single changes object of current and previous property values.

→ Called before ngOnIt() and whenever one or more data bound input properties change.

- **ngOnIt():**

  Initialize the directive/component after angular first displ the data-bound properties and sets the directive/component input properties.

→ Called once, after the first ngON changes().

- ngDoCheck():

Detect and act upon changes that Angular cannot or won't detect on its own.

→ Called during every change detection run, immediately after ngOnChanges() and ngONInit().

- ngAfterContentInit():

Respond after angular projects external content into the component's view / the view that a directive is in.

→ Called once after the first ngDoCheck().

- ngAfterContentChecked():

Respond after angular checks the content projected into the directive /component.

→ Called after the ngAfterContentInit() and every subsequent ngDoCheck().

- ngAfterViewInit():

Respond after angular initializes the component's views and child views/ the view that a directive is in.

→ Called once after the first ngAfterContentChecked().

- ngAfterViewChecked():

Respond after angular checks the components views and child views/ the view that a directive is in.

→ Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().

• ngOnDestroy():

Cleanup just before Angular destroys the directive/component.

→ Unsubscribe observables and detach event handlers to avoid memory leaks.

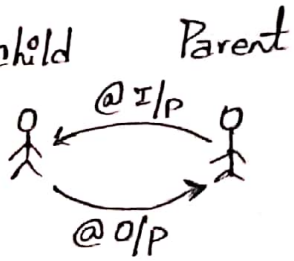
→ Called just before angular destroys the directive/component.

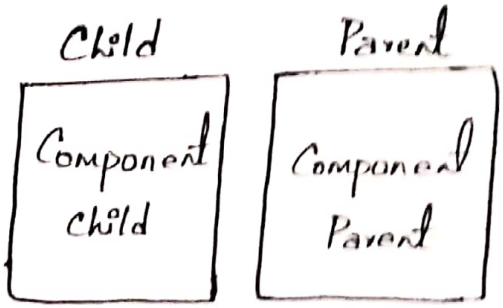
15/02/2019

• @Input, @Output Decorators:

@Input decorators:

To establish connection b/w parent to talk to child, we use @input.





app.html:

```
<app-child [myinput] = "inputText"> </app-child>
```

child.ts: → Property

```
@Input() myInput : String
```

app.ts:

In class

```
inputText = "hello i am your parent";
```

@Output:

to communicate b/w Child to Parent.

child.ts :

mandatory for @o/p.

```
@Output myOutput. EventEmitter <String> =
                    new EventEmitter( );
myOutputText = "hello : am your child";
        sendData( ) {
        this.myOutput.emit (this.myOutput Text).
```

child.html :

```
<button class = "btn btn-warning" > Senddata </button>
```

app.ts :

```
class --- {
getData (value){
 console.log(value)
}
```

app.html :

```
<app-child [myinput] = "inputText"
        (myOutput) = "getData ($event)
```

.. Decorator that marks a class as pipe and @input supplies configuration metadata.

@output :

→ @output decorator that marks a class field as an o/p Property & supplies configuration metadata. Declares a data bound o/p property, which angular automatically updates during change detection.

→ You can supply an optional name to use in templates when the component is instantiated that maps to the name of the bound property. By default the original name of the bound property is used for o/p binding.

5/02/2019

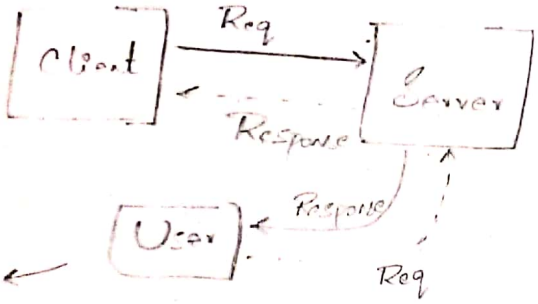
NODE JS

→ Single threaded, event bounded, I/p & o/p module.

→ NodeJS (web applications, IOT, ML, Streaming application)
(like Netflix, flite, Zoylete)

→ Java, SQL, J2EE, Ruby ... etc are blocking & Multithreaded.

→ By default, NodeJS is non-blocking code.



her response
if user ... here, then only it moves to other user's to implement
then ... (In java it blocking) [drawback results in error]

→ Ryan Dahl developed Node JS. in 2009.

He wrapped V8 with C++ since V8 is written in C++.

Non-blocking:

It can take multiple requests at a time from n users and executes all the requests.

V8 is Google's open source high-performance JS & web assembly Engine, written in C++.

How can a single threaded app accept multiple requests?

Event driven application, Event loop concept.

13/02/19

Introduction:

It is an runtime environment used to compile JS without browser.

> NodeJS is a JS runtime built on chrome's V8 JS Engine.

> Single - threaded - JS

> non - blocking code, event

> used for asynchronous calls.

> nodejs allows only global Object.

> Window functions such as alert results an error when executed in command prompt.

In command:

node filename.

→ Node JS is a server-side platform build on Google Chrome JS Engine (V8 Engine)

→ Node.JS was developed by Ryan Dahl in 2009.

→ Node.JS is a platform buildt on chrome's JS runtime for easily building fast and scalable network applications.
(used for chatting applications, network streaming applications [netflix, Yahoochat]).

Node JS uses an event-driven non-blocking I/o model.

3/2/19

Operating System - (OS):

In app.js : ← to use 3rd party moduleware, use this keyword.

```
const OS = require('os')
```
↳ built-in module (operating system)

The 'OS' module provides a no. of operating system related utility methods. It can be accessed using the above mentioned expression.

Install "cc-nodejs" from extensions

> used for importing & exporting similar to in angular.

```
Var test = os.arch(); → os architecture, CPU architecture.
console.log(test);
```

o/p: It displays the os architecture. i.e; 64 bit (or) 32bit.

```
Var test1 = os.cpus(); → displays configuration, model of CPU
Var test2 = os.freeman(); → It displays the RAM. size.
Var test3 = os.totalmem(); → results the max size of the memory.
Var test4 = os.homedir();
Var test5 = os.hostname();
Var test6 = os.platform();
Var test7 = os.release();
Var test8 = os.tmpdir();
Var test9 = os.userInfo(); username
Var test10 = os.networkInterface(); → LAN Interface, Bandwidth
Var test11 = os.type();                Local drives.
```

* The OS module provides information about the computer's operating system.

1) arch(): Returns the OS CPU architecture constant. Returns an object containing the operating system constants for process signals, error codes etc.

2) CPUs(): Returns an array containing information about computer's CPUs.

3) endianness(): Returns the endianness of the CPU.

4) EOL: It returns the end-of-line marker for the current operating system.

5) freemem(): Returns the no. of free memory of the system.

6) hostname(): Returns the hostname of the Operating system

7) networkInterfaces(): Returns the network Interfaces that has a network address.

8) platform(): Returns information about the operating system's platform.

9) release(): Returns information about the operating system's release.

10) tmpdir(): Returns the operating system's default directory for temporary files.

11) ~~totalmem()~~:

12) totalmem(): Returns the total memory of the system

13) type(): Returns the name of operating system.

14) uptime(): Returns the uptime of the OS, in seconds.

15) userInfo(): Returns the information about the current users.

24/09/2019
Tuesday

Event loop:

How is Javascript asynchronous and single-threaded?

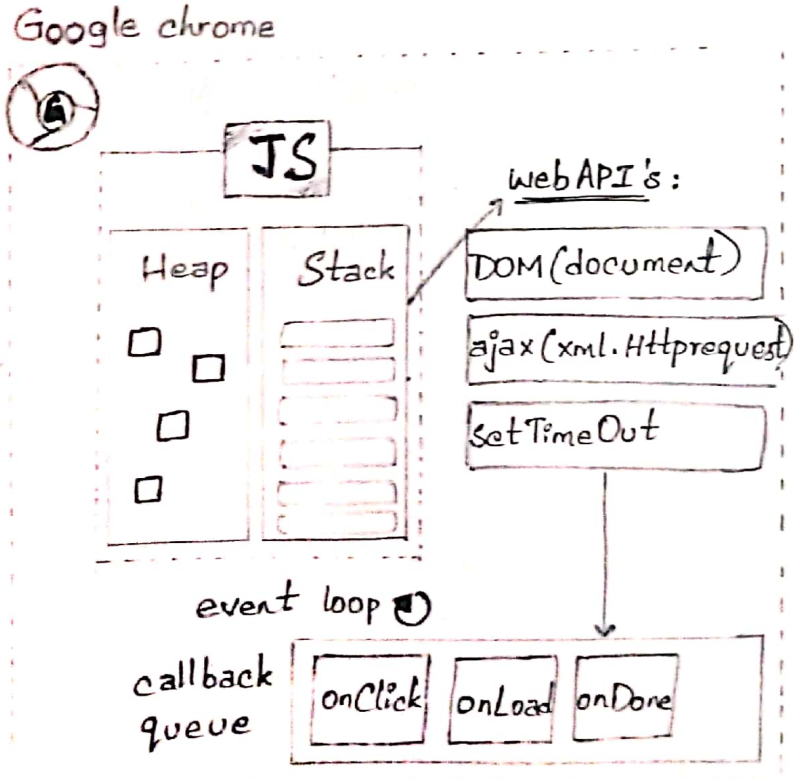
The short answer is that javascript language is single-threaded and the asynchronous behaviour is not part of the javascript language itself, rather they are built on the top of the core javascript language in the browser (or the programming environment) and accessed through the browser API's.

Heap: Objects are allocated in a heap which is just a name to denote a large mostly unstructured region of memory.

Stack: This represents the single thread provided for JavaScript code execution.

→ Function calls form a stack of frames.

Working of Event loop in chrome:



Fs module:

In app.js:

```
Var writeFile = WriteFileSync('writeme.text,' test);
```

↓

Synchronously writes a data to a file and replaces the text which is already exists.

```
console.log (writeFile);
```

Create Asynchronous calls:

```
Var readFile = ts. readFile ('readme text', 'utf-8',
```

↓

Asynchronous reads the content of a file.

```
callbackfunc ← function (err, data) {
        console.log(data);
    ts.writeFile ('writeFiletxt', data);
});
console.log(readFile);
console.log ('hello chandu');
```

O/P: Hello Chandu.

→ Since asynchronous calls, it executes the statements (or) requests in non-blocking way.

Node implements File I/o using simple wrapper around standard POSIX functions.

The node file System (fs) module can be imported using the following syntax.

```
Var fs = required ('fs');
```

Synchronous Vs Asynchronous :

→ Every method in the fs module has synchronous as well as asynchronous forms.

→ Asynchronous methods take the last parameter as the completion function call back and first parameter of the call back function as error.

It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, where as the second one does.

To create directory :

→ fs module provide mkdir function to create a directory.

→ 'mkdir' creates a directory with a mode of 0o777.

→ Nodejs is a call back function to connect with event loop.

app.js :

```
fs.mkdir('jspiders', function() {
    fs.readFile('readme.txt', 'utf-8', function (err, data) {
    fs.writeFile ('jspiders/writeme.txt', data);
    })
})
```

HTTP Module:

(see in nodejs.org)

Tomcat	Java
Apache Server	PHP

- 200-300 – Status code
- 404 – Page not found.
- 500, 501 – Server busy
- 200 – Success status

3 types of Server

→ file Server, FTP Server, Web Server.

To Create nodejs Server:

http module (↑ http)

```
const http = require('http');
const server = http.createServer
```

↳ function to create a simple nodejs server. ~~Object~~

callback function ← (function)

```
(function (req, res) {
```

req → request, res → response

```
res.writeHead (200, {'content-type': 'text/html'})
res.end (<h1> hello nodejs </h1>);
});
```

- Object → {'content-type': 'text/html'}
- writeHead ↓ write a response to a Client
- end ↳ End of the file (or) response.

```
const port = 4000;
```

↳ custom port

```
server.listen (Port, function (err) {
  if (err) {
    console.log (err);
  }
  else {
    console.log ('app is running on' + port);
  }
})
```

listen ↳ to listen the port is connect or not & return a error.

→ Content-type returns which type of file we use using. ex: textfile, html file, jsfile (or) any other type of file.

The built-in http module node js has a built in module called HTTP which allows Node js to transfer data over the Hyper Text Transfer Protocal (HTTP).

→ To include the HTTP module, use the require() method.

```
var http = require('http');
```

Node.js as a Web Server :

The HTTP module can create a HTTP Server that listens to Server ports and gives a response back to the Client.

→ Use the createServer() method to create an HTTP Server.

Ex:

```
var http = require('http')
// create a server object :
http.createServer (function (req, res) {
    res.write ('Hello World');
    res.end();
}).listen (4000); // the server object listens on port 4000.
```

the function passed into the http.createServer() method, will be executed when someone tries to access the computers on port 4000.

Add an HTTP header :

If the response from the HTTP Server is supposed to be displayed as HTML, you should include an HTTP header with the collect Content-type.

Ex:

```
Var http = require ('http');
http.createServer (function (req, res) {
 res.writeHead (200, {'Content-Type': 'text/html'});
 res.write ('Hello World!');
 res.end ();
}).listen (4000);
```

Remove file using fs module :

```
const fs : require ('fs-extra');
fs.unlink ('./jspiders/writeme.txt', (err) => {
if (err)
    throw err;
console.log ('successfully deleted');
});
```

```
npm install --save fs-extra
```

↓

so as to make use of write, mkdir, unlike have to install.

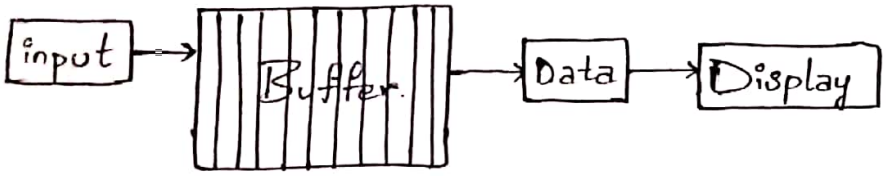
Remove directory using fs module :

→ Firstly the delete if there are any files in the directory i.e; unlink then only use "rmdir" to delete directory.

```
fs.rmdir ('./jspiders', function (err) {
if (err) {
 throw err;
console.log ('removed');
});
```

Buffer & Stream built in global modules in nodejs:

→ Buffer is the temporary storage or chunk of data transferring from one place to other.



→ Console.log (__dirname);
↳ it gives the path
Console.log (__filename);
↳ it gives the filename including the path.

} Global Objects in Nodejs.

→ It increases performance of the application.

app.js :

```
var readFileStream = fs.createReadStream
                    (__dirname + '/readme.txt');
var WriteStream = fs.createWriteStream
                    (__dirname + '/writeme.txt');
readFileStream.on ('data', function (@junk) {
    console.log ('file received ${junk}');
    writeStream.write (junk);
});
```

What is stream ?

Streams are objects that lets you read data from a source (or) write data to a destination in a continuous ~~function~~ fashion.

Problems with Large Data :

Speed : too low because it has to load all the requests

Buffer Limit : ~~+GT~~ 1GB

Stream Benefits:

→ Abstraction of (or) continuous chunking of data.

→ No need to wait for entire resource to load.

Stream is used in :

→ HTTP request & responses.

→ Standard input/output (stdline & stdout)

→ File reads & writes.

Types of Streams :

- Readable Stream :

Readable Stream is used to read Operations.

→ Standard input streams has data going into applications. This is achieved through read operation.

→ Input typically comes from the keyboard used to start the Process.

```
const fs = require('fs');
let data = '';
// create readable stream
let readableStream = fs.createReadStream('input.txt');
// set the encoding to be utf8.
```

```
readStream.setEncoding('UTF8');

// Handle stream events --> data, end.
readableStream.on('data', function(chunk) {
data += chunk;
});
readableStream.on('end', function() {
    console.log(data);
});
```

Writeable Stream :

Writeable Stream is used for write operations.

→ Standard output streams contain data going out of the application.

→ To write to stdout, we use write function.

Duplex Stream :

This is a stream which can be used for both read and write operations.

Transfer Stream :

A type of output stream where the output is computed based on input.

To Write a file using pipe :

```
var readFileStream = fs.createReadStream
                    (__dirname + '/readme.txt');
var writeStream = fs.createWriteStream
                    (__dirname + '/writeme.txt');
readFileStream.pipe(writeStream);
```

What is Piping in a stream ?

Piping is a process in which we provide the o/p of one stream as the i/p to another stream.

→ It is normally used to get data from one stream & to pass the o/p of that stream to another stream.

→ There is no limit on piping operation.

* npm install -g nodemon

↓

↓ node demon.

It starts Server

⇒ Nodemon is a tool that develop node.js based applications by automatically restarting the node application when file changes in the directory are deleted.

27/02/2019
Wednesday

Event Module Create an API :

```
else if (req.url === '/api')
{
   var api = [
       {
         name : 'chandu',
         age  : 22,
         company: 'capgemini'
       }
       {
         name : 'siri',
         age  : 22,
         company : 'Ispiders'
       }
   ]
   res.end (JSON.stringify (api));
}
```

→ By default, we use host : 3000 as server for node.js

Events Module :

→ Node.js has a built-in module called "Events", where you can create, fire and listen for your own events.

→ To include the built-in events module we use the require() method. In addition all event properties and methods are an instance of an Event Emitter Object. To be able to access these properties and methods, create

• create an Event Emitter object:

```
var event = require('events');
var eventEmitter = new events.EventEmitter();
```

• The Event Emitter Object:

→ You can assign event handlers to your own events with Emit Emitter object.

→ In the example below we have created a function that will be executed when a "test" event is fired.

• Node.js / Package.json :-

→ The package.json file is the heart of Node.js system

→ It is manifest file of any node.js project and contains the metadata of the project.

→ The package.json file is essential to understand, learn and work with node.js.

→ It is the first step to learn about development in node.js

What does package.json

The metadata information in package.json file can be categorized into below categories.

1) Identifying metadata properties :

It basically consists of the properties to identify the module / project such as name of project, current version of the module, license, author of the project, description about the project etc.

3] Functional metadata properties :

As the name suggests, it consists of the functional values / properties of project / module such as entry / starting Point of the module, dependencies in project, scripts being used, repository links of Node project etc..

Creating package json file :

A package json file can be created in 2 ways.

① Using npm init :

Running this command, system expects user to fill the vital information required as discussed above. It Provides users with default values which are editable by user.

Syntax : npm init.

② Writing directly to file :

Once can directly write into file with all the required information and can include it in the Node project.

Creating nodejs application :

→ In package.json, entry point is 'index.js'

cmd

↓

--- /Desktop

mkdir nodecap

↓

npm init

↓

Package Name : "jspiders"

version : :

description : It is a nodejs application.

27/09/2019

Express Js (12 Ques)

- Express Js.com ← is one of the official website. /en/resources/frameworks.html. [Frameworks built on Express JS

Web Applications :

Express JS : Express JS is a minimal & flexible node.js web application framework that provides a robust set of features for web and mobile applications.

→ This is backward framework, server side.

Frameworks built on Express JS :

1] Feathers : Build prototypes in minutes and production ready real-time apps in days.

2] ItemsAPI : Search backend for web and mobile applications built on Express & Elastic search.

3] Keystone JS : Website and API application framework / CMS wi an auto generated React.js Admin UI.

4] Kraken : Secure and scalable layer that extends Express b Providing structure and convention.

5] LoopBack : Highly extensible, open source Node.js framework for quickly creating dynamic end-to-end Rest APIs.

6] Mean : Opinionated full stack JavaScript framework that simplifies and accelerates web application development.

7. Sails: MVC framework for Node.js for building practical, Production-ready apps.
8. Hydra-Express: Hydra-Express is a light weight library which facilitates building Node.js Microservices using Express JS.
9. Blueprint: a solid framework for building APIs and backend services.
10. Locomotive: Powerful MVC web framework for Node.js from the maker of Passport.js.
11. GraphQL-yoga: Fully-featured, yet simple and lightweight GraphQL server.
12. Express Gateway: Fully featured and extensible API Gateway using Express as foundation.
13. Dinoloop: Rest API Application framework powered by typescript with dependency injection.
14. Kites: Template based Web-Application Framework.
15. FoalTS: Next-generation framework for building enterprise-grade Node.js applications (Type Script).

'--save'. It saves in package.json stores only metadata devdependencies, node modules.

→ The dev dependencies are used for development purpose.

→ The `express()` function' is a top-level function.

→ app.get('*');
↓
all pages.

[expressjs.com/en/guide/routing.html]

Routing: Routing refers to how an application's endpoints (URIs) respond to client requests. For an introduction to routing.

- You define routing using methods of the Express app object that correspond to HTTP Methods;
for Ex: app.get() to handle GET requests and app.post() to handle post request.
- For a full list, see app.method. You can also use app.all() to handle all HTTP methods. and app.use to specify Middle ware as a call back function
- These routing methods specify a call back function (sometimes called "handler functions") called when the application ~~listens for~~ receives a request to the specified route (end point) and HTTP Method. In other words, the application "listens" for request that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.

→ In fact, the routing methods can have more than one call back function as arguments. With multiple callback functions, it is important to provide next as an argument to the call back function and then call next() within the body of the function to hand off control to next callback.

The following code is an example of a very basic route:

```
var express = require ('express')
var app = express().
app. get ('/' function (req, res) { // respond with "hello nodejs"
 res. send ('hello nodejs')          when a GET req is made to
 })                                  Homepage.
```

Route Methods : [expressjs.com/en/4x/api.html#app.METHOD.]

A Route Method is derived from one of the HTTP methods, and is attached to an instance of the express class.

Express supports methods that correspond to all HTTP request methods: get, post and so on.

Eg: app.all()
app.delete()
app.disable()
app.disabled()
app.enabled()
app.engine()
app.get()
app.listen()
app.METHOD()
app.param()
app.post()
app.path()
app.put()
app.render()
app.route()
app.set()
app.use().

Route paths :

Route paths, in combination with a request method, define the endpoints at which requests can be made. Route paths can be strings, string patterns or regular expressions.

Ex: ① This route path will match requests to the root route, /

```
app.get('/', function (req res) {
  res.send ('root')
});
```

② This route path will match requests to /about.

```
app.get ('/about', function (req, res) {
  res.send ('about')
});
```

Response Methods :

The Methods on the response object (res) in the following can be send a response to the client, and terminate the request-response cycle. If none of these methods are call from a route handler, the client request will be left hanging.

Method	Description
1] res.download()	Prompt a file to be download
2] res.end()	End the response process
3] res.json()	Send a JSON response
4] res.jsonp()	Send a JSON response with JSONP support.

7] res.redirect() Redirect a request.

8] res.render() Render a view template.

9] res.send() Send a response of various types.

10] res.sendFile() Send a file as an octet stream

11] res.sendStatus() Set the response status code and send its string representation as a response body.

Middle Ware :

In Expressjs, we cannot update new values, so as to do that we make use of 3rd party middleware.

express-handlebars middleware gives proper template engine.

Installation :

```
npm install express-handlebars --save.
```

↓

views folder is mandatory to be created.

↓

After creating express-handlebars it is mandatory to maintain an extension of ".handlebars" for "html file" or else returns an error.

↓

views folder

↓

layout folder (folder)

↓

main.handlebars (file)

Get Bootstrap & copy the code & paste in "main.handlebars"

⇒ For navbar :

views folder
↓
Partial folder
↓
In partial templates ← _navbar.handlebars
↓
getbootstrap/examples
↓
navbar example (copy & paste in _navbar.handlebars)

⇒ In main.handlebars, {{> _navbar}}.

Middleware functions : [expressjs.com/en/guide/writing-middleware.html]

→ Middleware functions are functions that have access to the request object (req), the response object (res) and the next function in the application's request-response cycle. The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.

→ Middleware functions can perform the following tasks.

- Execute any code
- Make changes to the request and response objects.
- End the request-response cycle.
- Call the next middleware in the stack.

→ If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

Eg:
```
var express = require ('express');
var app = express();
app.get ('/', function (req, res, next) {
  next();
});
app.listen (3000);
```

HTTP method for which middleware function applies.

Path (route) for which the middleware function applies

The middleware function

call back argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

→ An Express application can use the following types of middlewares.

1.] Application level middleware.
2.] Router-level middleware
3.] Error-handling middleware.
4.] Built-in middleware
5.] Third party middleware.

Mongoose ODM :

Mongoose is Object Data Modelling (ODM) library for MongoDB & Node.js.

MongoDB is a schema-less No SQL document database. It means you can store JSON documents in it, and the structure of documents.

mongoose.js.com → official website.

→ Can vary as it is not enforced like SQL database.

→ Install Mongo DB

↳ mongodb.com

→ mlab.com

↳ contains all mongodb libraries.

Server.js :

```
const mongoose = require('mongoose');
const app = express();
mongoose.connect('mongodb' // Chandu : <Password> @ cls 25505
                 mlab.com : 55005 / nodecap', { userMo
                 Client : true.
}).then(() => Console.log('mongodb connected'))
  .catch(err => Console.log(err));
```

Url connect

Installation :

to install mongoose we have to first install nodejs & MongoDB.

Mongodb atlas :

New Project

↓

Project name :

Project author :

Clusters

↓

Overview → Create cluster

↓

Security → IP White list

↓

Access from any where

↓

URL 0.00.0/0

↓

MongodB users

↓

Add new user

↓

Chandu

<Password>

↓

Connect → Security

↓

28/02/2018

Setup connection security

↓

username
Password

↓

Choose a Connection method.

↓

Connect to an application

↓

Standard connection string

↓

Copy the URL connection string.

↓

In Server.js

In Server.js:

```
const mongoose = require ('mongoose');
mongoose.connect ('.....link.....');
.then (() => Console.log ('Mongodb is connected');
.catch (err => Console.log (err));
```

↓

Mongodb is connected.

To Create a homepage:

views folder

↓

ideas folder

↓

add.handlebars

↓

In server.js :

```
app.get ('/ideas/add', (req, res) =>
{
    res.render ('./ideas/add');
    console.log (req.url);
}).
```

To link bootstrap & jQuery :

Get bootstrap
↓
download
↓
bootstrap CDN
↓

Copy the bootstrap code and paste in main.handlebars.

Schema :

Everything in mongoose starts with a schema. Each schema maps to a MongoDB collection & defines the shape of the documents within that collection.

Create Models out of views so as to store the schemas.

Mongoose model → Defines a model & retrieves it.

Models defined on the mongoose instance are available to all connection created by the same mongoose instance.

Models

↓

Idea.js

In Idea.js :

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const IdeaSchema = new Schema({
  title : {
    type : String ;
    required : true.
  },
  details : {
    type : Sting ,
    required : true .
  },
    type : Date,
  default : Date.now()
});
mongoose.model ('ideas', IdeaSchema);
```

In server.js :

```
require ('/models/idea);
const Idea = mongoose.model ('idea');
```

npmjs.com → Body Parser (3rd part module to retrieve the data to DB.)

Body-Parser (middle ware) :-

→ Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

Note :

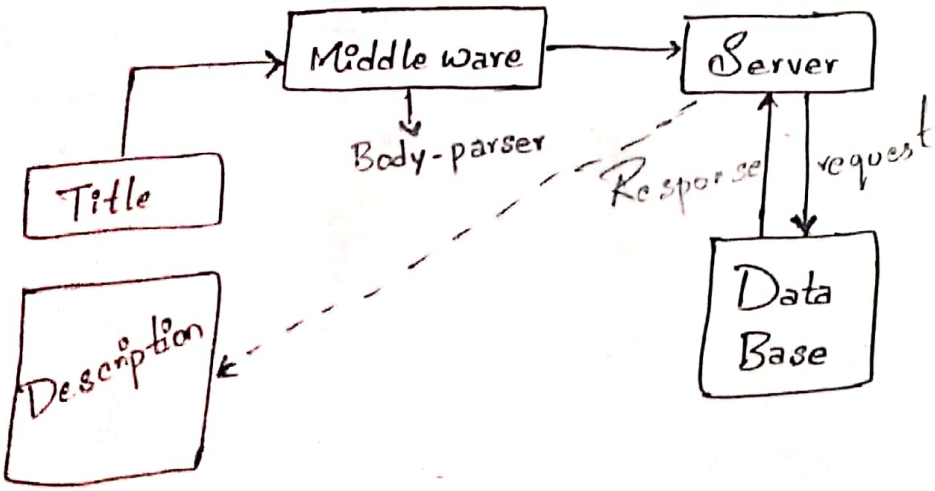
As req.body's shape is based on user controlled input, all Properties and values in this object are untrusted & should be validated before trusting.

Ex: req.body.foo.toString() may fail in multiple ways,

for Ex: the foo property may not be there (or) may not be a string & tostring may not be a function & instead of String (or) other user input.

Installation :

```
npm install body-parser --save
```



In server.js : // Schema and Validation.

```
app.post ('/ideas', (req, res) => {
   // res.send ('ok');
Console.log (req.body);
const.errors = [ ];
if (! req.body.title) {
  errors.push ({
     text : 'Please add some content' }).
};
if (errors.length > 0) {
  res.render ('./ideas/add', { errors : errors,
            title : req.body.title;
           details : req.body.details }) }
else {
const Users = {
       title : req.body.title,
      details : req.body.details }
     new ideas (Users).save().then(( ) => {
       res.direct ('./ideas') }) } } ).
```

To retrieve data from MongoDB :

```
app.get ('/ideas', (req, res) => {
    Idea.find().then(ideas => {  // It searches all the
        res.render ('./ideas/ideas', {   objects.
            ideas : ideas
    });
});
```

* find1() : Searches for specific object in the collection of Objects.

idea.js :

```
const mongoose = require ('mongoose');
const Schema   = mongoose.Schema ;
const IdeaSchema = new Schema ({ title : { type : String ,
                                    required : true } ,
                                details { type : String, required :true } ,
                    date : { type : Date ,
                                default : Date.now
                    }});
mongoose.model ('ideas', IdeaSchema).
```

Server.js :

```
const express = require ('mongoose');
const mongoose = require ('mongoose');
const app  = express ();
```

```
var body parser = require ('body-parser');
mongoose . connect ('Link copied from mongoDB');
.then (() => { Console. log ('Mongo DB connected');
}). catch (err => console. log (err))
    require ('./models/ideas');
const Idea = mongoose. model ('ideas');
```

Types of middlewares :

You can load application-level & router-level middleware with an optional mount path. You can also load a series of middleware functions together, which creates a sub-stack of the middleware system at a mount path.

1.] Application - level middleware :-

Bind an application level middleware to an instance of the app object by using the app.use() and app.method() functions where METHOD is the HTTP method of the request that the middleware function should handles such as (Get, Put or Post)

Ex:
```
var express = require ('express')
var app = express();
app. get ('/', function (req, res, next) {
  next();
});
  app. listen (3000);
```

→ The above example executes whenever the app receives a request.

→ This Ex shows a middleware function mounted on /user/:id, path of HTTP request on the /user/:id path.

```
app.use ('user/:id', function (req, res, next) {
    console.log ('Request Type:' req.method)
    next()
})
```

→ Here is an example of loading a series of middleware functions at a mount point, with a mount path. It illustrates a middleware sub-stack that prints request for any type of HTTP request to the /user/:id path.

Ex:

```
app.use ('user/:id', function (req, res, next){
    console.log ('Request URL:', req.originalURL)
    next()
}, function (req, res, next) {
    console.log ('Request type:' req.method)
    next()
})
```

Router handlers enable you to define multiple routes for a path. The ex below defines 2 routes for GET requests to the /user/:id path. The 2nd route will not cause any problems, but it will never get called because the first route ends the request-response cycle.

The Ex shows a middleware sub-stack that handles get requests to the /user/:id path.

Ex:
```
app.get('/user/:id', function (req, res, next) {
  console.log('ID', req.params.id)
  next()
}, function (req, res, next) {
  res.send('User Info')
})
// handler for the /user/:id path, which prints the user ID
app.get('/user/:id', function (req, res, next) {
  res.end(req.params.id)
})
```

⇒ To skip the rest of the middleware functions from a router middleware stack, call next('route') to pass control to the next route.

Note:

- next('route') will work only in middleware functions that were loaded by using the app.Method() or router.method() functions.

→ This ex shows a middleware sub-stack that handles GET requests to the /user/:id path.

```
app.get('/user/:id', function(req, res, next) {
// if the user ID is 0, skip to the next route
  if (req.params.id === '0')
  next('route');
// otherwise pass the control to the next middle function
in this stack.
  else
    next()
 },
function (req, res, next) {
  // (Send a regular response)
    res.send('regular')
})
// handler for the /user/:id path, which sends a special
response.
app.get('user/:id', function (req, res, next) {
res.send('special')
})
```

→ Middleware can also be declared in an array for reusability.

This ex shows an array with a middleware sub-stack that handles GET requests to the /user/:id path.

```
function logOriginalUrl (req, res, next) {
  console.log ('Request URL : 'req.originalURL') ne.
  next()
}
function logMethod (req, res, next) {
  console.log ('Request Type : ', req.method)
  next()
}
Var logStuff = [logOriginalUrl, logMethod]
app.get ('/user/:id', logStuff, function (req, res, next) {
  res.send ('user/Info')
})
```

Router-level Middleware :

Router-level middleware works in the same way as applicati level middleware except it is bound to an instance of express Router().

```
Var router = express.Router();
```

Load router-level middleware by using the router.use() & router.method() function.

The following ex code replicates the middleware system that is shown above for application-level middleware by using router-level middlewares:

```
Var app = express()
Var router = express.Router()
// a middleware fo func with no mount path. This code is
executed for every request to the router.
router.use (function (req, res, next) {
 Console.log ('Time:', Date.now())
 next()
})
// a middleware sub-stack shows request info for any type
of HTTP request to the /user/:id path.
router.use ('User/:id', function (req, res, next) {
console.log ('Request URL:,' req.originalURL()
next()
},
function (req, res, next) {
console.log ('Request Type:', req.method)
next()
})
// a middleware sub-stack that handles Get requests to the
/user/:id path.
```

```
router.get('/user/:id', function (req, res, next) {
  console.log('Request URL :', req.originalUrl()
  next()
},
function (req, res, next) {
  console.log('Request Type', req.method)
  next()
})

// a middleware sub-stack that handles GET request to the
   /user/:id path.
router.get('/user/:id', function (req, res, next)
{
  if (req.params.id === '0')
    next('route');
// otherwise pass control to the next middleware function in
   this stack:
  else
    next()
},
function (req, res, next) {
    // render a regular page
  }) res.render('regular')
// handle for the /user/:id path, which renders a special page.
router.get('/User/:id', function (req, res, next) {
  console.log(req.params.id)
  res.render('special')
})
```

// mount the router on the top.

app.use ('/', router);

→ To skip the rest of the router's middleware functions, call next ('router') to pass control back out of the router instance.

→ This ex shows a middleware sub-stack that handles get requests to the /user/:id path.

```
var app = express()
var router = express.Router()
// Predicate the router with a check & bail out when needed.
router.use (function (req, res, next) {
 if (!req.headers ['x-auth']) return
     next ('router')
   next()
})
router.get ('/', function (req, res) {
 res.send ('hello. user');
});
// use the router & 401 anything falling through
app.use ('/admin', router, function (req, res) {
 res.sendStatus (401)
});
```

Error-handling Middleware :

Error handling middleware always takes 4 arguments. You must provide 4 arguments to identify it as an error-handling middleware function. Even if you don't need to use the next object, you must specify it to maintain the signature. Otherwise, the next object will be interpreted as regular middleware and will fail to handle errors.

Define error handling middleware functions in the same way as other middleware functions, except with 4 arguments instead of 3, specifically with the signature.

```
(err, req, res, next));
app.use (function (err, req, res, next) {
console.log (err.stack)
res.status (500).send ('Something broke!');
3)
```

For details about error-handling middleware see error-handling.

Built-in middleware :

Starting with version 4.x. Express no longer depends on Connect. The middleware functions that were previously included with Express are now in seperate modules. See the list of middleware functions.

→ Express has the following built-in middleware functions.

1] express static serves static assets such as HTML files, images & so on.

2.] Express.json parses incoming requests with JSON payloads (Available with Express 4.16.0+)

3.] Express.Urlencoded parses incoming requests with URL-encoded payloads.

Third-party middleware :

Use 3rd party middleware to add functionality to Express apps. Install the Node.js module for the required functionality, then load it in your app at the application level (or) at the router level.

The following ex illustrates installing and loading the cookie-Parsing middleware function cookie-parser.

→ npm install cookie-parser.

```
Var expressson = require ('express');
Var app = express( )
var cookie parser = require ('cookie-parser');
// load the cooking-parsing middleware.
app.use (Cookie Parver( ));
```

For a partial list of 3rd party middleware functions that are commonly used with Express.

Body-Parser :

```
app.use (body.parser.urlencoded ({ extended : false }));
app.use (bodyParser.json( ));
```

The above mentioned statements are mandatory to use body parser.

Express handlebars :-

// middleware view engine.

```
app.engine('handlebars', exphbs({defaultLayout: 'main'}));
app.set('view engine', 'handlebars');
```

Express & Scaffolding :

- It is a generator used for bundling all the files.

```
npm install -g express-generator.
```

Now,

↓

```
express -h
```

- gives all templates engines present in express.

↓

```
express --view=pug Jspiders.
```

cat JSpiders
Package.Json

↓

```
cd Jspiders
```

↓

```
npm install
```

This automatically generates all the files.
i.e; views folder, public, routes, images folder.

↓

staticfiles

express generator is a basic boiler plate for ejs

↓

embedded javascript

Pug - standard.

Jade and Pug (Template Engine) :-

official website → Pugjs.org.

mkdir Pug

↓

cd pug

↓

" npm install pug -cli -g "

↓

" Pug -w ./ -o ./html -P "

- -w → watch root folder
- -o → open html file
- -P → Print something.

O/P :

watching index.pug.
rendered "-- file location."

Pug : Pug is a template engine for express JS, and for frontend & to write loop, variable.

Mandatory to follow indentation :

No opening & closing tags.

index.pug :

```
html
    head
        link(rel = "stylesheet", href = "style.css").
    body
        h1 hello
```

index.html:

```
<html>
    <head>
        <link rel="stylesheet" href="style.css" />.
    </head>
    <body>
        <h1>Hello</h1>
    </body>
</html>.
```

index.pug :

```
body
    h1 (class = "demo") This is title.
                (or)
    h1.demo . This is title.
    h1 (id="demo") (or) This is id.
    h1# This isId This is Id.
```

* #demo hello this is para

If any tag is not mention, it considers itself as

```
input (type = "text" name = "name" id = "name" autofocus)
textarea (name = " " cols = "30" rows = "10")
select (name = "lang" multiple = "true") disabled.
option (value = "java") java
option (value = "python") Python.
option (value = "javascript") Javascript.
```

```
input (type = "radio" name = "gender").
| Male
input (type = "radio" name = "gender"):
| Female
input (type = "checkbox", name = "language")
| JavaScript
input (type = "checkbox", name = "language")
| Node JS.
```

Variables in Jade & Pug :

```
- & const myClass = ['btn', 'btn-success', 'btn-sm']
- let myStyles = {'background-color': 'red', 'color': 'red'}
div (class myClass) hello div
div (style = myStyles) This is styles.
div This is each loop example
 - let array = ["java", "Python", "node js"]
for x in array.
    P = x.
head
    script.
        alert ('hello js');
        console.log ('hello Console');
    script (src = "app.js");
```

04/03/2019

Monday

Jade is an elegant templating engine, primarily used for server side templating in Node JS.

In plain words, Jade gives you a powerful new way to write markup, with no. of advantages over plain HTML.

Blocks of Text :

Let's say you have a paragraph tag & you want to place a large block of text in it.

Jade treats the first word of every line as an HTML tag

- so what do you do?

You might have noticed an innocent period in the first code ex is the

Adding a period (full stop) after tag indicates that everything inside that tag is on each line as an HTML tag.

Pug features :

Loops

JavaScript

Interpretation

Mixins.

Mixins :

→ Reusable code.

```
mixin comment(data)
  .comment
    .date = data.date
```

• name = data.name

. text = data.text.

html

head

≡

body

- const users = {date : '4/3/2019', name : 'alekya', text : 'This is mixin'}

- const users 1 = {date : '23/9/2019', name : 'Chandu', text : 'This is Chandu'}

+ comment (users)

+ comment (users 1)

Includes :

→ To include external pug file into other folder.