**Application:** An application is a software component to perform certain task.

**Web Application:** A web application is a software component to perform data exchange over internet. Web app developed by using **web technologies** and run on **web browser**.

**Web Browser:** A web browser is a web technologies compiler. It compilers HTML, CSS & JS to browser understandable format and display the data.

**Web Technologies:**

HTML, CSS, Java Script.

* Web technologies are developed under **World Wide Web**.
* WWW given restriction we shouldn’t create any alternative for web tech (HTML, CSS, JS).
* WWW given feasibility to develop library/ frame work on the top of web tech.

**CSS Frame work:**

* Bootstrap: CSS FX developed by twitter. BS providing ready made classes.
* Sass/ SCSS frame work: SASS frame work helps to develop CSS using OOPS principles.
* OOPS Principles:
  + Encapsulation
  + Abstraction
  + Inheritance
  + Polymorphism
  + SASS FX will not fallow 100% OOPS Principles.
* Material UI: Material UI providing styled element.
* Etc..

**JS Frame work:** FX providing resources to develop an application.

* FX having set of libraries which provide syntax.
* Angular

**JS Libraries:** Lib providing syntax to develop an app.

* JQuery
* React JS
* VUE JS
* Kendo JS.
* ETC….

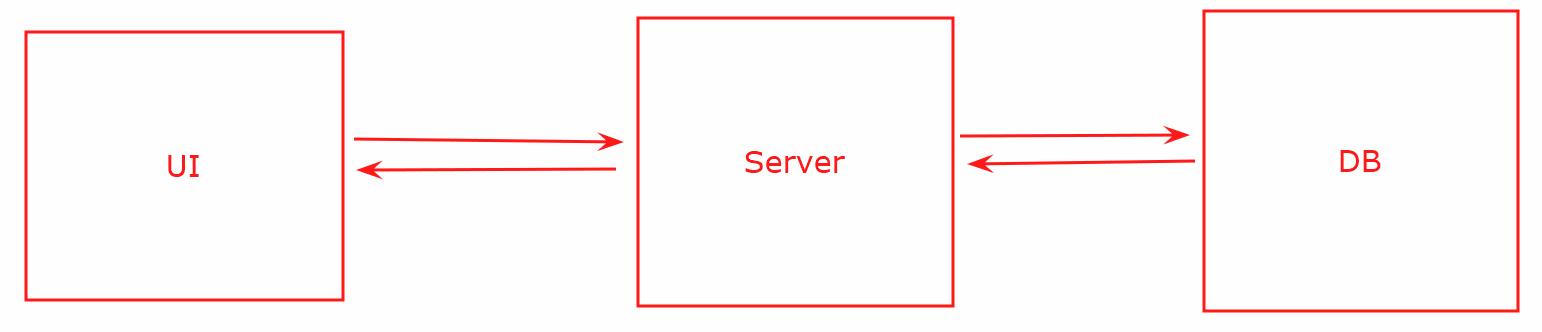
**Diff b/w Programming language and Script:**

JS is not a programming language, it is scripting language.

* Programming language compiled by the compiler and converts to binary format.
* Scripting language compiled by the transpire. Transpiler translate script at run time (while pgm executing).

**3 tier architecture:** Any software application fallows this 3 tier architecture.

* It having 3 tiers.
* Tire1 -> UI
* Tire2 -> Server
* Tire3 -> DB



**Data Center:** Data Center providing space to store the information. DB, server will run on data center.

**Data Base**: DB is a physical information storage.

**Server:** Server is a bridge b/w the DB and UI to exchange the data. Server is used to perform data validations.

**UI:** User Interface display the data.

**Pgm language naming conventions:**

1. **Camel Case**: starts with lowerLetter. Ex: firstName
   * variables, function names
2. **Pascal Case**: Starts with UpperCaseLetter. Ex: EmployeeDetails.
   * Class name
3. **Chain case**: rep by -. Ex: employee-detail
   * CSS class, folder names, file names.
4. **Snake case**: rep by \_. Ex: employee\_detail.
   * Folder, files name.

**Skeleton of HTML**:

<!DOCTYPE html>

<html>

<head>

To add Meta information. (css, js, meta, keywords)

</head>

<body>

Display Content.

</body>

</html>

**Web Applications:** A web application is a software component, it was developed by using web technologies and run on web browser.

**Types: 2 types.**

1. Static web app – HTML & CSS.
2. Dynamic web app – Static web app + JS -> User intractable web app.

**HTML:** Hyper Text Markup Language.

* Purpose: Display the content in the browser.

**CSS:** Cascading style sheet.

* Purpose: Beautification, beautifying the content to be display.

**JS – Java Script:** JS adds events & actions to make user intractable web app.

**React:** React was developed by Facebook, to develop web & mobile app’s.

* **React JS:** Develop web app.
* **React Native:** Develop mobile app.

**Core Components:**

There are 3 core components.

1. Elements
2. Tags
3. Attributes

**HTML Intro:** HTML was developed by Tim Berners Lee (1980 - 1990).

Officially released in 1992.

**Types of elements:**

Elements are 3 types.

1. Inline elements
2. Block level elements
3. Empty elements

These 3 types are categorized into 2 types.

1. Schematic elements
2. Non-Schematic elements

**Schematic elements:** The elements which are used for specific purpose.

Ex: table, image, iframe, link, button, etc….

**Non-Schematic Elements:** The elements which are used for general purpose.

Ex: p, label, span, div, Etc….

**Folder structure of web app:**

* Sample-web-app
  + Index.html: In real time project entire app loads in single html file. So, it is called root html file.
  + Assets: Application specific common files.
    - Images
    - JS
    - CSS
  + Components: app specific other features.

**Relation b/w files:** to refer a file (image, html, ….) in another file we must know relation b/w files.

1. Sibling relation: files are in same folder having sibling/ side by side relation files.
   * Rep by ./
2. Parent child relation: files are in diff folders having parent and child relation.
   * Rep by ../

**Ex**:

<body>

        <h2>Welcome to sample web app...!</h2>

        <img src="./assets/images/web-app.jpg">

</body>

**Working with images: There are 3 types of images using in real time project.**

1. Inline image: added by using img element.
2. Back ground image: added by using css
3. Image link: (inline/ bg image) buttons, icons, ….

Ex:

<style>

            body {

                background-image: url('./assets/images/web-app.jpg');

                background-repeat: no-repeat;

            }

        </style>

**Img attributes:**

* Width, height, alt: when image not able to download display some text

**Block level element:** The element will take 100% width in the browser.

* Block level element having css prop display: block as default.

Ex: div, h1 to h6, p, …..

**Inline elements:** The element will take width as per the content in the browser.

* By default all the elements are inline element.

Ex: span, label, all input elements, img, ….

**Empty element:** An element will not rep any content.

Ex: hr, br

**Element:** Element rep the content to be display.

**Tag:** tag rep the element. Ex: paragraph element rep by p tag.

Image -> img tag

Table cell -> td

**Attribute:** Att will add appearance and behavior to an element.

Ex: style, class, onclick, ….

**Int Que:**

* Types of images?
* What is an element and types of elements?
* Schematic element vs Non-Schematic element?
* What is a tag and attribute?

**Light Weight:** In terms of software, light weight means, the software will take less memory in size.

**Scripting:** Script compiled at run time. Ex: Java Script, Perl & Shell, Python.

**Programming language:** Compiled at compiling time. Ex: Java, Python, C#, ….

**Browser:** Browser compiles HTML, CSS & JS.

**Design:**

**Static: HTML, CSS, BS**

**Dynamic**: Static + JS

**Multipage App -> Project**

**React JS**

**Project**

**Redux**

**Application:** An app is a software component, to perform certain task.

Types:

1. Web app (Internet)
2. Mobile app (Internet)
3. Windows app (Intranet (internal))
4. Gaming app (Intranet)
5. N/W app
6. Embedded Systems
7. VLSI app
8. Data Science
9. Etc

**Web App:**  A web app is a software component, it was developed by using **web technologies** and run on **web browser**.

**Web Browser:** Web browser transpiles web technologies to human untreatable content.

**Note:** Compilations are 2 types.

Compilation means converting code to human understandable.

1. Compilation – Done by compiler (JAVA, .Net, …..)
2. Transpilation – Done by transpiler (Web Technologies, Python script, …)

**Web Technologies:** HTML, CSS, JS.

* Web Tech. maintained by world wide web.
* Web tech. started in 1980, into market by 1995.

**HTML:** Hyper text markup language.

* Developed by Tim berners-lee (1980, 1992).
* Initial Idea: To maintain research information docs in digital format and send to others.

**HTML Core Components:**

1. Elements
2. Tags
3. Attributes

**Element:** Element rep the content to be display.

Ex: paragraph, headers, table, table cell, image, etc…

**Tag:** tag rep the element.

Ex: paragraph – p tag, headers -> h1 to h6, image -> img, etc…

**Note:**  we can create our custom element. React works on custom element principle.

Custom element – user defined element.

**Attribute:** Attribute rep the appearance and behavior to an element.

Ex: Appearance – style, class

Behavior -> action -> onclick, onchange, all event attr.

**Skeleton of HTML:**

<!DOCTYPE html>

<html>

<head>To add meta information.</head>

<body>Rep content</body>

</html>

**Structure of web app:**

* Create a folder
  + Index.html -> root html/ entry html file
  + Assets
    - Css
    - Images
    - Js
  + Source folder – app specific files (login, ….)
    - Login folder
      * Login.html
      * Login.js
      * Login.css

**Types of elements:** elements are 3 types.

1. Block level element
2. Inline element
3. Empty element

**Block level element:** An element takes 100%/ full width in the browser.

Ex: div, p, table ….

Block level element having css prop -> display: block;

**Inline elements:** An element takes width as per the content in the browser.

Ex: label, span, img, a, ….

**Empty Element:** An element not rep any content.

Ex: hr, br

**Int Que:**

1. What is a web app?
2. Types of web app?
3. What is a HTML?
4. What are the core components of HTML?
5. What is an element and types of element?

**Task:**

1. Create web app structure and create sample app?

**Element categories:** into 2 types.

1. Schematic elements
2. Non-schematic elements

**Schematic elements:** Elements which are used to specific purpose.

Ex: image, iframe, anchor element, etc…

**Non-Schematic elements:** Elements which are used for generic purpose.

Ex: span, label, div, p, etc….

**Tag:** tag rep the element.

**Types:**

1. Table tags: table, tr, td, th.
2. Image tags: img.
3. Form tags: form, all input elements, select.
4. Frame tags: iframe.
5. List tags: order list, unorder list, data list -> ol, ul, li, dd, dt, dd.
6. Meta tags: script, link, title. Tags rep under head section are called meta tags.
7. Etc.

**IDE: Integrated development environment.**

* IDE is a platform to write, build and run code.

Ex: notepad++, vs code, …

**Open project in VS Code:**

1. Go to files -> open folder.
2. Through cmd: go to the folder address bar and type cmd.-> **code .** and press enter.

**Table element – Used as a grid:**

* Display data in rows and columns.

**Ex:**

   <table>

            <thead>

                <tr>

                    <th>Product ID</th>

                    <th>Product Name</th>

                    <th>Product Price</th>

                    <th>Product Image</th>

                </tr>

            </thead>

            <tbody>

                <tr>

                    <td>100</td>

                    <td>Dell laptop</td>

                    <td>37000</td>

                    <td>

                        <img src="./assets/images/laptop.jpg">

                    </td>

                </tr>

</tbody>

</table>

**Styles:** Style adds beautification to content.

There are 3 ways to add style to an element.

1. Inline style
2. Internal style
3. External style

**Inline style:** Using style attribute adding style to an element.

* Inline means adding style with in the element using style attribute.

**Syntax**: <tag style=”key:value; key: value”></tag>

Ex: <table style="width: 100%; background-color: rgb(237, 233, 233); border: 1px solid black;">

**Internal Styling:** Adding styles to an element with in the html page under head section using style element.

**Meta –** Meta will not display any content, it will adds extra parameters to a content.

Ex:

<style>

            table {

                width: 100%;

                background-color: rgb(237, 233, 233);

                border: 1px solid black;

            }

            th {

                border: 1px solid black;

            }

            td {

                border: 1px solid black;

            }

        </style>

**External Styling:** Creating separate CSS file and add that file in the html page under the head section using link element.

* Create index.css file and add styles.

Ex: <link href="./index.css" rel="stylesheet">

**Inline vs internal vs external styles:**

* **Inline** styles applicable for that element.
* **Internal** styles applicable only for 1 html page and we can reuse those style with in that html page.
* **External** styles where ever we need we can using within the project.

**Int Que:**

1. What is schematic and non-schematic elements?
2. What is a tag and types of tags?
3. How many ways to apply style to an element?
4. Internal vs inline vs external styling?

**Task:**

* Create products grid/ table and apply inline, internal and external styling?

**Styling to an element in internal/ external:**

1. Tag based styles
2. Id based styling
3. Class based styling
4. Attribute based styling
5. Relation based styling

**Tag based styling:** using tag and apply style to an element.

* In real time project use tag based styling to apply common style across the app.

**Syntax:**

tag {

styles;

}

Ex:

table {

width: 100%;

}

**ID based styling:** id rep by #.

* Apply style to specific id.
* In entire app, only 1 id should be maintain.

Syntax:

#id {

Styles;

}

Ex:

#products {

    width: 100%;

}

<table id="products">

**Class based styling:** rep by .className.

* Apply common styles.

Syntax:

.class-name {

Styles;

}

Ex:

.recent-products {

    width: 100%;

}

<table class="recent-products">

**Attribute based styling:** Apply styles based on attribute.

Rep by [attr]

Syntax:

[attrkey=”attrvalue”] {

Styles;

}

Ex: [class="products"] {

    background-color: aqua;

    padding: 10px;

}

<h2 id="recentProducts" class="products">Recent Prodcuts</h2>

**Custom attribute:** in html 5.0 custom attribute was introduced.

Rep by **data-\***. In place of \* we can mention any name.

Ex:

<td data-offer="a">37000 10% offer</td>

[data-offer="a"] {

    background-color: lightgreen;

}

**Int Que:**

1. What is an attribute and how to apply attr based styles?
2. What is a custom attribute?
3. How many ways to apply styles to an element in internal/ external?
4. Diff b/w id, class and attribute based styling?

**Task:**

Create static app and apply styles? Take products data and display?

**Relation based or combinatory selectors:**

**DOM:** Document Object Modal.

**There are 4 global objects** in JS.

1. Window object -> parent of all the objects.
2. Document object -> loads HTML, CSS & JS
3. Location object -> url specific information
4. Screen object -> screen resolutions.

* HTML loads in the DOM as tree structure. In this structure elements have 2 relations.

1. Parent & child
   1. Immediate child’s
   2. Any child’s
2. Sibling relation
   1. Immediate sibling
   2. Any sibling

**Immediate child’s:** In DOM tree structure element will have child and grand child elements.

Child elements wrt an element is called immediate child’s.

* Immediate child is rep by **>**

Syntax: css\_selector > css\_selector {

Styles;

}

Here CSS selector can be a tag, id, class or attribute.

Ex:

div > h2 {

    background-color: brown;

}

**Any child:** Childs, grand child’s under an element are any child.

Syntax: rep by space.

css\_selector css\_selector {

Styles;

}

Ex:

div h2 {

    background-color: brown;

}

**Immediate sibling:** Side by side elements are immediate elements.

Syntax: rep by +

css\_selector css\_selector {

Styles;

}

Ex:

h2 + section {

    background-color: brown;

}

**Any Sibling:** all the side by side elements.

Syntax: rep by ~

css\_selector ~ css\_selector {

Styles;

}

Ex:

h2 ~ section {

    background-color: brown;

}

**Iframe:** iframe is used to inject a document in another document (HTML).

<iframe src=”path”></iframe>

**Hover pseudo class:** this will add hover effect.

.menu-item:hover {

    background-color: blue;

}

**Int Que:**

1. What is a DOM?
2. What are the global objects in JS?
3. What are combinatory selectors?
4. Hover pseudo class?
5. What is the use of iframe?

**Task:**

* Create a static web page with menu.

**Multipage App:** In Multi page app, we will create separate pages for each feature (login, reg, etc…).

* In MPA, all the pages will be loaded in one root page (index.html).
* In MPA, iframe is used to load the required page in the index.html.
* In MPA, anchor tag invoke the required page to be load in iframe.

**Anchor tag:** Anchor tag is an action element it will open document/ page/ html file in the same window or another tab.

Syntax:

<a href=”path” target=”target type”></a>

**Create home page in the source folder**

Source

-> home

->home.html

->home.css

**Anchor tag target types:**

1. \_self
2. \_blank
3. \_top
4. \_parent
5. Custom type

\_self: self loads the html page with in the same iframe.

Ex:

 <a href="../home/home.html" target="\_self">Home</a>

\_blank: blank loads the html age in the new tab.

Ex:

<a href="../about/about.html" target="\_blank">About US</a>

\_top, \_parent: these will load the html page in the same browser tab by changing the url.

<a href="../contact/contact.html" target="\_top">

   Contact

</a>

**Custom target:**

Custom target must be starts with \_

Ex: \_container, \_document, etc..

To the anchor tag action path in the specific iframe, iframe name must be same as target.

Ex:

<a href="../home/home.html" target="\_container">Home</a>

<iframe src="" class="container" name="\_container"></iframe>

**Note:** in the parent html page have the css/ js file references, even in the child html pages also must be give the references.

* In multi page app, a common css (fonts, text colors, etc) needs to be refer in all the html pages.
  + 1 css file is 50 kb and refer this css file in 20 html pages, here 1000kb ~= 1mb. Unnecessarily 950k was wasting.

**Int Que:**

1. What is a MPA?
2. What are the targets of anchor tag?
3. How to create custom element?
4. What is the use of iframe name?
5. Drawback of MPA?

**Task:**

* Create MPA (Home, contact, about)?

**\*** By using MPA we can add all the pages in one container.

\* By using MPA, we need to give multiple references of common files.

**CSS Position:** Position property defines the position of an element in/ on the screen.

* Static
* Fixed
* Relative
* Absolute
* Sticky

**Static:** By default all the elements are display static.

* Static in the screen.

**Fixed:** Fixed prop is used to define an element at specific fixed position.

* Fixed on the screen.
* Use cases: Popups, chat box, scroll icon, fly out, …..etc.
* Position: fixed prop use another 4 props to define the exact position of an element.
  + Top
  + Left
  + Bottom
  + Right

Ex:

<div style="position: fixed; background-color: cadetblue; width: 250px; height: 40vh; right: 10px; bottom: 10px;">

            Test

        </div>

**Common CSS selector:** We can apply styles commonly for diff selectors at a time.

* Each selector is separated by ,

**Syntax**:

Selector, selector {

Styles;

}

**Opacity:** Opacity defines the intensity of a color.

Opacity from 0 to 1;

**Int Que:**

1. What is a position CSS prop?
2. Use of position fixed prop?
3. Position supporting props?
4. Opacity?

**Task:**

Create chat box and popup?

**Position relative and absolute:**

* Display/ show an element wrt to another element.
* For ex, hover on menu item show sub menu. Menu item -> p: relative, sub menu item -> p:absolute.
* Real time use cases: Menu & Submenu, element and tooltip. Hover action to show some element.

Ex:

<!DOCTYPE html>

<html>

    <head>

        <link href="./index.css" rel="stylesheet">

    </head>

    <body>

        <h2>Menu & Submenu</h2>

        <div class="menu">

            <label  class="menu-item">

                <a href="#" class="menu-item-link">Home</a>

            </label>

            <label  class="menu-item">

                <a href="#" class="menu-item-link">Products</a>

                <ul class="sub-menu">

                    <li class="sub-menu-item">

                        <a href="#" class="sub-menu-item-link">Electronics</a>

                    </li>

                    <li class="sub-menu-item">

                        <a href="#" class="sub-menu-item-link">Mobiles</a>

                    </li>

                    <li class="sub-menu-item">

                        <a href="#" class="sub-menu-item-link">Laptops</a>

                    </li>

                </ul>

            </label>

            <label  class="menu-item">

                <a href="#" class="menu-item-link">Contact US</a>

            </label>

            <label  class="menu-item">

                <a href="#" class="menu-item-link">Registration</a>

            </label>

        </div>

    </body>

</html>

.menu, .sub-menu {

   background-color: black;

   padding: 5px;

}

.menu-item-link, .sub-menu-item-link {

    color: white;

    text-decoration: none;

    padding: 10px;

}

.menu-item {

    position: relative;

}

.sub-menu {

    display: none;

    position: absolute;

    margin: 0px;

    list-style-type: none;

    left: 0px;

    top: 20px;

}

.menu-item:hover .sub-menu {

    display: inline-block;

}

**Note**: Parent element should have position relative and child element should have position absolute.

Ex:

.tool-tip {

    display: none;

    position: absolute;

    top: -20px;

    left: 25px;

    background-color: black;

    border-radius: 8px;

    color: white;

    padding: 2px;

}

.tooltip-item:hover .tool-tip {

    display: inline-block;

}

**Int Que:**

* Position relative & absolute?
* Creating menu and submenu?

**Task:**

* Create menu and submenu and tool tip.

**Bootstrap:** BS is a CSS frame work. BS providing predefined classes to build web responsive app.

* BS was developed by twitter.

**Bs.min.css file** is a minification file. Minification removing the white space.

**BS Providing classes for:**

* Text
* Background
* Button
* Table
* Images
* Layouts
* Card
* Grid
* Etc..

**BS Container classes:** Container classes adds space to the web site.

2 container classes.

1. Container
2. Container-fluid

**Spaces CSS:** There are 2 types of spaces we can add.

1. **Inner space – padding**
2. **Outer space – margin**

**Padding:** Add inner space.

Ex: <input type="button" class="btn btn-primary" value="Click Me...!" style="margin: 30px;">

We can add padding all sides/ 2 sides/ specific side.

|  |  |  |
| --- | --- | --- |
|  | CSS padding | BS Class |
| All sides | Padding: 10px;  padding: 1px 2px 3px 4px; | p-1 to p-5 |
| **Specific side**  Top  Right  Bottom  Left | Padding-top:10px;  Padding-right:10px;  Padding-bottom:10px;  Padding-left:10px; | Pt-1;  Pe-1;  Pb-1;  Ps-1; |
| **Co-ordinates**  X (left to right)  Y (top to bottom) | Padding: 10px 20px;  10px -> top, bottom  20px -> left, right | Px-1;  Py-1; |
|  |  |  |

**Button Classes:**

btn btn-\*

Here \* is a suffix class.

**Text classes:** text-\* (\* is a suffix class)

**Back ground classes:** bg-\* (\* is a suffix class)

**Table classes:** table table-\* (\* is a suffix class)

**BS Suffix classes:**

1. Primary – blue
2. Secondary – gray
3. Success – green
4. Danger – red
5. Info
6. Warning
7. Dark
8. Light
9. Default

Based on the context we can use the respective class.

**BS table classes:**

Ex:

<table class="table table-bordered table-hover table-striped">

            <thead class="table-success">

**Int Que:**

1. What is a Bootstrap?
2. What are the BS suffix classes?
3. BS bg, text, table, button classes?
4. What is diff b/w padding and margin?
5. What are the BS padding & margin classes?

**Task:**

* Create a static website with BS?

**Dev Tools:** Developer to debug the code.

* Open: right click -> inspect page, press F12 on browser (fn + F12).

**Dev tools contains**:

1. Select element
2. Toggle Laptop & Mobile
3. Elements -> html page
4. Console -> test JS code, log the scenario data.
5. Source -> debugging
6. Network -> All the API calls and files downloaded from the server logs.
7. Application -> Storages and other items.
8. Etc…

**Responsive web app:**

Web site response (look and feel) change as per the resolution.

There are 2 ways to design.

1. CSS flex box & Media Query -> flexibly arrange the elements.
2. BS row and col system

**Flex box:**

Display: flex;

Flex-direction: row/ column;

Flex-wrap: wrap/no-wrap;

Flex item CSS:

Flex: 30%; (item width).

Flex-grow: 1; (item size - dynamic).

**Media Query:** media query helps to change/ override the css props as per the specified resolution.

Syntax:

@media only screen and (max-width: size) {

Override class css prop

}

Ex:

@media screen and (max-width: 700px) {

    .flex {

        flex-direction: row;

    }

}

Max-width -> 0 to specified width.

Min-width -> specified width to infinite.

<meta name="viewport" content="width=device-width, initial-scale=1.0">

To work with media query we need to add viewport meta tag.

**BS responsive design classes:**

BS providing responsive classes: row and col-\* (here \* is the size of a column).

BS col sizes from col-1 to col-12.

Here each column size is 8.3333%

Col-1 -> 8.3333%

Col-2 -> 16.6666%

….

Col-12 -> 100%



**Dashboard:** Dashboard display the summary of data in the project.

**BS Card System:** card, card-header, card-body, card-footer.

**BS form classes:** form-group, form-control

<div class="form-group">

   <label>Email</label>

   <input type="text" class="form-control">

</div>

**Int Que:**

1. Viewport meta tag?
2. BS responsive design classes?
3. BS card system classes?
4. BS form classes?

**Task:**

* Use BS responsive classes design
  + Layout
  + Dashboard

**Sassy CSS:** It provides dry code.

Dry -> Don’t repeat yourself.

* Developed in 2006 by Hampton.
* Purpose: Reduces no. of lines of CSS code.
* SASS: Syntactically awesome style sheet.
* SCSS: Sassy CSS.
* Browser will not understand Sassy CSS. Browser understand only css, so we need to convert sassy css to css.
* Here, conversion is done by compiler.
* There are paid and free compilers are available.
* Here, will use Node JS compiler.

**Sassy CSS – OOPS:**

**OOPS Principles:**

1. Encapsulation: Creating a class itself we can achieve encapsulation.

Nesting SCSS, Variables

1. Abstraction: Hiding potential information and exposing required information.

Importing

1. Inheritance: Code reusability. Having parent (base) and child classes (derived).

Mixing, Extend

1. Polymorphism: Many forms.
2. Overriding: Variables overriding (Def: Parent and child class having same function signature and overriding the parent class method in child class).
3. Overloading: Method parameters (Def: In a class creating a methods with same name and same return type and vary in input params/ optional params).

**Sassy CSS Concepts:**

1. Variables
2. Nesting
3. Mixin
4. Extend
5. Import

**Node JS:** Node JS is a web app compiler. It compiles source (SCSS, TS, React JS, etcc..) to CSS and JS.

* Def: Node JS is a frame work. It is used to develop web app.
* Install: from node js official website.
* When installation of node js, node package manager will be downloaded.
* To add any package (lib/ FX) use cmd > npm package\_name.
* Cmd’s to check
  + Node JS > node –v
  + Npm > npm –v

**Create Node JS Project:** npm init

* Go to the folder address bar and type cmd. It will open cmd > npm init
* It will ask few question simply press enter key.
* It will create with package.json file
* Add index.html and index.scss files.
* To convert scss files to css files we need to add scss package.

Cmd> npm install package\_name

Cmd> npm install sass

* Add the compilation cmd in the packahe.json file under scripts.
  + sass source\_file target\_file
  + “compile”: “sass index.scss index.css”
* In index.html refer index.css only.
* > npm run compile

**Cmd to compile when there is change in the scss file: use –w (watch)**

“compile”: “scss –w source\_file.scss taget\_file.css”

When ever change and save the scss file then css file will modify by the watch cmd.

**Nesting:**

section {

    $fontSize: 50px; -> Syntax of creating a varaible

    div {

        color: red;

        font-size: $fontSize;

    }

    p {

        color: brown;

        font-size: $fontSize;

    }

}

Variable scope:

2 scopes.

1. Local scope
2. Global scope

**Local scope:** variable under the function/ class.

**Global Scope:** create variable outside of a class.

**Overriding variable (override the variable value):**

$fontSize: 50px;

.section {

    $fontSize: 25px !global;

div {

…

}

}

Import: we can split the classes into multiple files and import all the files in one file.

* At the end one CSS file only generate.

Create index.scss and index1.scss

In index.scss add below

@import ‘./index1.scss’;

**Note**: once we inported index1.scss in index.scss, varaibles in the index1.scss file will be available for index.scss

**Mixing & include:**

* Creating a set of css using @mixing and use it in class by using @include.

Ex:

@mixin font-text {

    font-family: 'Times New Roman', Times, serif;

    font-size: $fontSize;

    text-decoration: underline;

}

.section1 {

    div {

        color: red;

        @include font-text;

    }

    p {

        color: brown;

        @include font-text;

    }

}

**Extend:** We can extend one class by another class.

div {

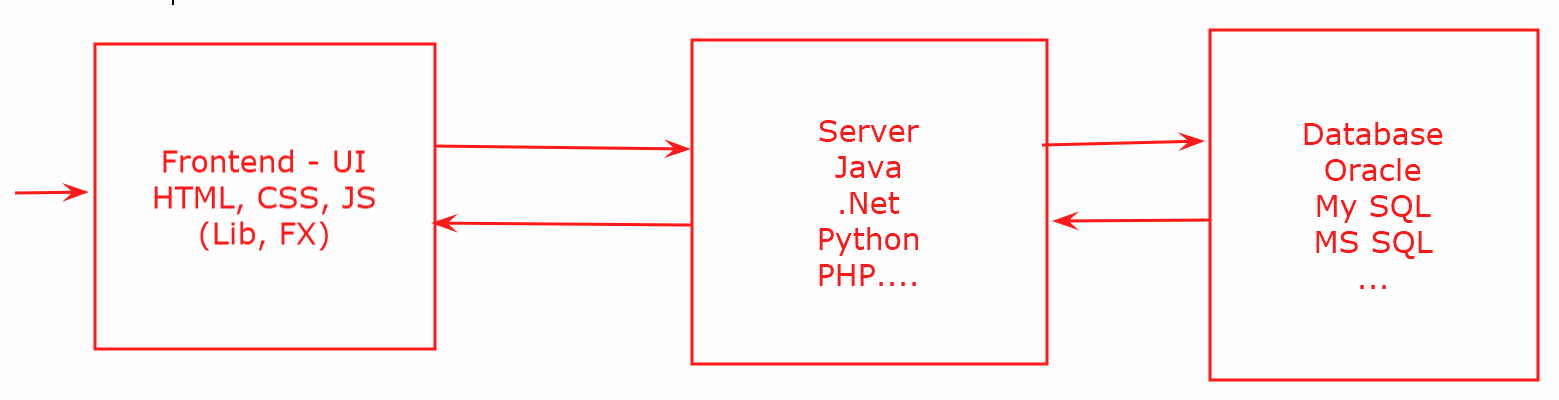
        color: red;

        @extend .font-text;

    }

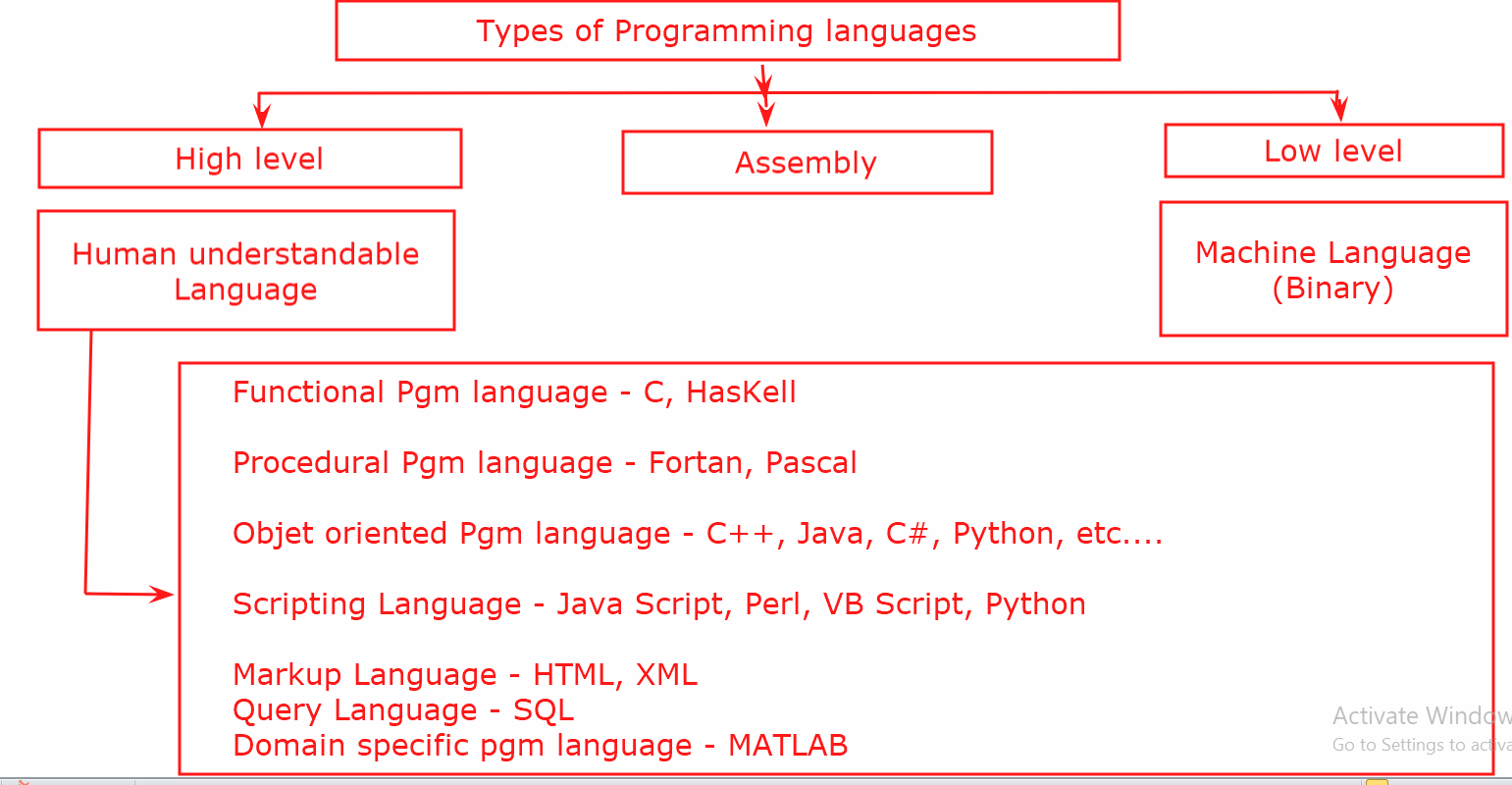
**Web App:** A web app is a software component, that will be developed by using web technologies and run on web browser.

A web app developed in 3 parts (3 tier architecture).



**Programming language:** A pgm language contains set of statements to perform certain task on operating system.

* A pgm language is a mediator/ comm. B/w developer and machine (computor).



**Note:**

* High level pgm language converted to low level pgm language by frame work compiler.
* Low level pgm language converted to human understandable by Operating system.

**Compiler types:**

1. Compiler: Convert entire pgm to byte code. (any error at specific code entire compilation fails).

Ex: Java, C#, ….

1. Transpiler: execute line by line. (any error at specific code, before code lines will execute).

Ex: Java Script, HTML, ….

**Pgm language cases:**

1. camelCase: starts with lower case letter.

Ex: employeeDetails

Use: variables, function

1. PascalCase: First letter in a word should be capital.

Ex: EmployeeDetails

Use: Class Name

1. chain-case: 2 words are joined by -.

Ex: bg-color, employee-details.js

Use: CSS, folder, files names.

1. Snake case: rep by \_.

Ex: emp\_details, private/ functional level variables ( \_name ).

Use: folder, files names.

**IDE: Integrated development environment**

Ex: Visual Studio Code.

**Web App Folder structure:**

1. Index.html -> root html page.
2. Index.css
3. Assets
   1. Images
   2. CSS (common)
   3. JS (common)
4. Source: Application specific feature file.
   1. Registration
   2. Login

**(VS Code ->** file -> open folder -> select req. folder**).**

**Int Que**:

1. What is a pgm language?
2. What is a web app?
3. What are the web technologies?
4. Types of pgm languages?
5. Compiler types?
6. Cases to write pgm language?
7. Web app folder structure?

**Task**:

1. Create web app folder structure and execute.

**Once OS loads the browser window, there are 4 objects created by the browser.**

1. Window
2. Location
3. Document
4. Screen

**Window:** Window is the parent of all the objects.

**Location:** It rep URL information.

**Document:** It rep the HTML document. By using document object we can manipulate the HTML.

**Screen:** It rep screen information.

**Java Script:** JS is a dynamically typed client side scripting languages.

* By using JS we can build user intractable web app’s.
* JS was developed by Brendan Eich in 1995.

**Apply JS to an element:** There are 3 ways.

1. Inline
2. Internal
3. External

**Inline:** Adding JS code with in the HTML element is called inline JS.

* JS link to an element through the behavioral attribute.

[

HTML contains 3 core components: element, tag & attribute.

**Attribute:** Att add appearance and behavior to an element.

]

Note: Behavioral attr’s are prefix with on.

Ex:

<body onload="alert('Welcome')">

       <h2 style="background-color: lightblue;" onclick="var x=10; alert(x);"> VKC trainings </h2>

</body>

**Internal JS:** Adding JS code under head section using script tag is called internal JS.

<script type="text/javascript">

            function onLoad() {

                alert('Welcome');

            }

            function headerClick() {

                var x = 10;

                alert(x);

            }

</script>

**External JS:** Creating separate JS file and add the JS file reference in the HTML page.

<script type="text/javascript" src="./index.js"></script>

**Note:** External JS is recommended.

1. Code reusability (we can give multiple references).
2. Reduce complexity.
3. Performance (JS files are cached, any changes in the server new changed JS file will not refled due to cache. So, we need to implement versioning).

**Int Que:**

1. What are the global Objects in JS?
2. What is a JS?
3. How many ways to apply JS to an element?
4. Diff b/w inline, internal and external JS?

**Tasks:**

* Create web application with JS code (inline, internal, external)?

**Hosting JS app:** Web app can be hosted by using node js cmd.

**Install node js:** download from official website and open downloaded msi file and click on next,… finish.

**Check node js installed/ not:**

* Open control panel and check installed software.
* Cmd> open cmd prompt > node –v
* Along with the node js node package manager will be installed (npm).
* Check version of npm: cmd> npm –v
* npm is used to add the any nodejs package, run and buld the js app’s.
* install http-server cmd to run web app.
  + npm install –g http-server (-g -> globally installation).
* Run web app -> open web app and go to the app address bar and type cmd. Run below cmd.
  + http-server –p 3000 (-p: port [any electronic signal run on specific port]).
  + Browser -> <http://localhost:3000/>

Java Script was developed by taking the references of functional programming language (c), Java and scripting language (Perl, Python).

* JS officially released in 1995.
* While working on JS we must fallow below:
  + Identifiers
    - Name of a variables, functions.
  + Comments
    - Comment adds to the logic to explain about the logic.
    - 2types
      * Inline -> //
      * Multiline /\* …..\*/

Ex:

/\*

Jira: 120 - Creating a onload logic

\*/

* + Statements: A statement is an instruction to perform a task.
    - Data type, function, if statement, loops, storages, timers, etc…
  + Whitespace: space, tab space, new line. Even whitespace also take some space.
    - To reduce the file size we are doing magnification (removing white space).
  + Note: Java script is a case sensitive.

**Int Que:**

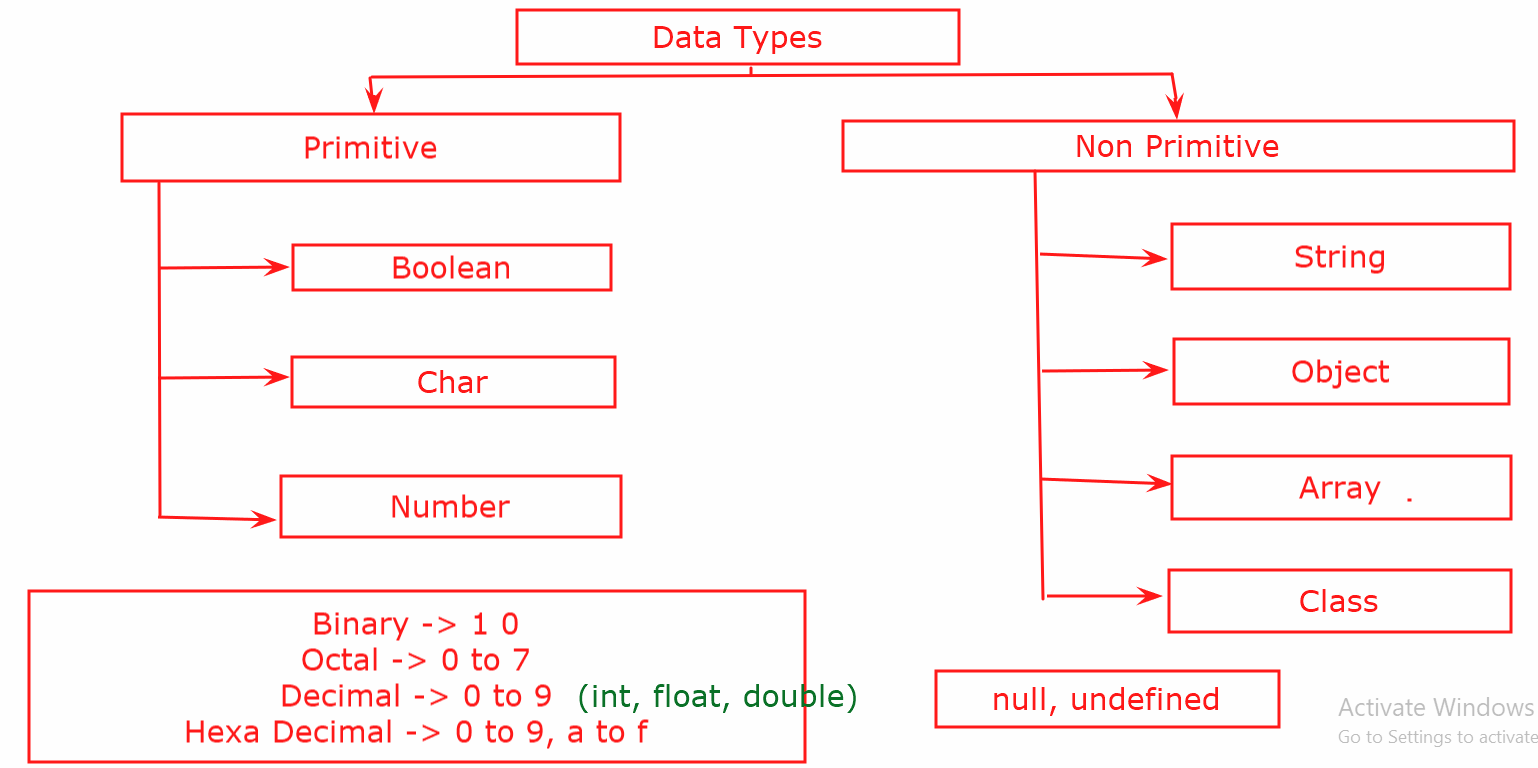
* How to install node js?
* How to check node js and npm version?
* How to host web app?
* How the cache works?
* What is an identifier?
* What is a statement?

**Task:**

Create web app. and host it?

**Data Type:** A data type is a data carrier.

* Data type is used to create a variable and variable will holds the data.



**Java script is a dynamically typed pgm language. Don’t have specific data type.**

**In JS we are using var, let, const to declare a variable.**

Syntax:

Var/let/const variableName = data;

Variables deals about:

1. Declaring a variable
2. Dynamically typed/ Loosely typed variable
3. Variable scope
4. Variable hoisting

**Declaring a variable:**

var name; -> single variable declaration.

var fName, lName; -> multiple variable declaration.

**Check data type of a variable:** using typeof()

var fName = "Durga", lName = "Prasad";

    lName = 10;

    alert(typeof(fName)); //String

    alert(typeof(lName)); //Number

**Note:** In JS we can create a variable without var/let/const, bcs JS is a dynamically typed scripting language.

**Loosely typed variable:** create a variable and assign string data and in the next line of pgm assigning number/ other data type.

(Creating a variable without var/ let/ const).

Ex: above ex create lName as loosely typed variable.

**Variable scope:** A scope of a variable defined by the position of declaration.

There are 3 scopes.

1. Global scope
2. Local scope
3. Block scope

**Global Scope:**

* Creating a variable outside of a function, those variables having global scope.
* Loosely typed variables in the function becomes global variables.

Ex:

<input type="button" onclick="save()" value="save">

<input type="button" onclick="reset()" value="reset">

function save() {

    fName = "Prasad";

    alert(fName);

}

function reset() {

    alert(fName);

}

**Local Scope:** Creating a variable with var/ let/ const in function becomes local scope variables.

Ex:

function save() {

    var fName = "Prasad";

    alert(fName);

}

**Block scope variable:** Creating a variable inside of a code block (ex for loop).

* Let is a block scope variable.

Ex:

function save() {

    var fName = "Prasad";

    if(1==1) {

        var x = 10;

        let y = 11;

    }

    alert(x);//10

    alert(y);//error

}

**Hoisting:** assigning data to a variable before declaration is called hoisting.

* In JS var created variable and function will fallow hoisting.

Ex:

function save() {

    x = 10; // Assigning data to x

    var x;  // Declaring x varaible

    alert(x);

}

* Let, const doesn’t support hoisting.

**Const:** Once assign data to a variable we can’t change.

Ex: const x = 10;

X = 11; -> error

const y;

y = 10; //work

y = 11; //error

**Int Que:**

1. What is a data type? And types?
2. What a variable and syntax?
3. How to declare a variable?
4. What is mean by loosely typed variable?
5. Scope of a variable?
6. What is hoisting?

**Task:**

* Create application and test varaibles.

**Function:** A function encapsulate the logical unit.

* A function having a set of statements to perform certain task.
* Function is a logical unit.

1. General function (no i/p params, no return type).
2. Parameterized function.
3. Returnable function.
4. Anonymous/ variable defined function.
5. Lambda function/ Arrow function.
6. Callback function
7. Self invoking function/ immediately invoking function (SIF/ IIF)
8. Generator function

**General function:**

**Syntax**: function functionName() {

//Logic

}

Ex:

function add() {

    alert(1 + 2);

}

**Parameterized/ argument function:** A function create with params/ args is called parameterized/ argument based function.

**Syntax:** function functionName(params/ args) {

//logic

}

**Types of params:**

1. **Mandatory params**
2. **Optional params: assigning default value to a param.**

**Ex:** function add(a, b, c=1) {

    alert(a + b + c);

}

1. **Named params: (not work in JS)**

function arthematicOperations() {

    add(b=1, a=-5);

}

function add(a, b, c=1) {

    alert(a + b + c);

}

**Returnable function**: a function with return type is called returnable function.

* A function perform a logic and return the result to another function.
* To create reusable function.

Ex:

function arthematicOperations() {

    let result = add(1, 2);

    alert(result);

}

function add(a, b) {

    return a + b;

}

**Anonymous/ variable defined function:** A function without a name is called anonymous function.

* Anonymous function assigning to a variable is called variable defined function.
* Anonymous function used as a callback function.

**Syntax**: var/let/const variableName = function(params) {

//Logic

}

**Ex**:

var add = function(a, b) {

    return a + b;

}

**Arrow function:** In ES6 anonymous functions are created as arrow function.

Syntax: var/let/const variableName = (params) => {

//Logic

}

Ex:

var add = (a, b) => {

    return a + b;

}

**Int Que:**

1. What is a function?
2. Types of function?

**Task:**

Create a function examples (for all types)?

**Callback function:** A function passing as i/p param to another function and executing the passed function is called callback function.

* Getting API responses.
* Performing a task after certain interval of time.
* Executing an action based on another action.

**SIF/ IIF:** A function called by itself is called SIF/IIF.

Syntax:

(function() {

//Logic

})();

Ex:

(function() {

    alert(1);

})();

**Note:**  function supports hoisting (before declaring a function we can call a function).

* Variable defined functions will not support hoisting.

**Timers in JS:** Timers are used to perform a task / execute logic at certain time interval.

There are 2types of timers.

1. setTimeout
2. setInterval

**onload:** onload is an event to notify once the html loaded.

**setTimeout:** is a JS event to perform a task/ execute a logic at certain time period.

//After web loads, after 5secs show alert.

//1000 -> 1 sec

function onLoad() {

    setTimeout(handler, 5000);

}

function handler() {

    alert(1);

}

function handler() {

    alert(1);

}

const handler = function() {

    alert(1);

}

const handler = () => {

    alert(1);

}

function onLoad() {

    setTimeout(() => {

        alert(1);

    }, 5000);

}

**How to cancel the timeout before execution?**

**clearTimeout(timer);**

**setInterval:** is a JS event to perform a task/ execute a logic contentiously at certain time interval.

Syntax: setInterval(handler, interval);

Ex:

setInterval(()=>{

        let date = new Date();

        console.log(date);

    }, 1000);

**clearInterval?**

**Strict mode:** strict mode introduced in ES6. Strict mode won’t all developer assigning data to a variable with out declaration.

* Syntax: “use strict”; -> add in line 1 in JS file.

**Data Types:**

**Boolean:** true/ false;

**null:** assigning null value to a variable.

Ex:

let x = null;

alert(typeof(x)); //Object

alert(x); //Null

**undefined:** not assigning any value to a variable.

let y;

alert(typeof(y)); //undefined

alert(y); //undefined

**Comparing 2 values with equal:**

== -> this will check data of variables only.

=== -> check data + data type.

Ex:

if(x === y ){

    alert("Null == Undefined");

}

else {

    alert("Null != Undefined");

}

let a = 1;

let b = "1";

a==b? true

a===b? false

**String:** string is a series of char’s.

String rep: 3 ways.

1. Single quotes: ‘krishna’
2. Double quotes: “krishna”
3. Back ticks: `krishna`

**String concatenation:**

1. Operator overloading: joining 2 strings by +
2. Interpolation:

Ex:

let firstName = "Varun";

let lastName = "Kumar";

let fullName = "Full Name: " + firstName + " " + lastName; // Operator over loading

alert(fullName);

fullName = `Full Name: ${firstName} ${lastName}`; // interpolation

alert(fullName);

**Int Que:**

1. What is a timer and timer types?
2. How to cancel/ clear timer?
3. What is a strict mode and how to use it?
4. Null and undefined types?
5. Comparison using == and ===?
6. String concatenation types?

**Task:**

1. Crate timers and display msg, cancel timers?
2. Ex on comparison and string concatenation?

**Object:** Object is used to rep coll. Of specific item.

* For ex: employee details, order details, ….
* Object contains key and value pair.
* There are 2 ways to declare an object.

**Literal:** Object declare by using {key: value, key: value}

**Ex:** let user = { name: “krishna”, email: “k@gmail.com” };

**Object type:** Object declare by using new Object({key: value});

**Ex:** let user = new Object({ name: “krishna”, email: “k@gmail.com” });

**Key rules**:

* Key should be unique.
* Key must be a string and single word.
* Key can be add/ without adding quotes we can use.

**Accessing object values:**

* By using object key.
* Ex: user.name

**Get all the keys from the object:**

document.write(Object.keys(user));

**Get all the values from the object:**

document.write(Object.values(user));

**Arrays:** an array is a coll of items. Items can be any data type.

* Arrays are rep in 2 ways.

**Literal:** array rep by [item1, item2, ….]

Ex: let attenddes = [“Sai”, “Krishna”, “Lakshmi”];

**Object type:** rep by new Array(item1, item2);

Ex: let attenddes = new Array(“Sai”, “Krishna”, “Lakshmi”);

Here, item can be a string, number, object, array.

**Array items are accessed by using index.** Index starts from 0.

**Get no. of items in the array:** using length.

**Copy array or object:** Object and array are reference data type. While we copy object/ array data from one variable to another variable instead of copying data address will copy.

Ex:

let user3 = { name: "Ramanji", email: "r@gmail.com"};

let user4 = user3;

document.write(user4.name); // Ramanji

user3.name = "Ram";

document.write(user4.name); //Ram

To fix above situation we are having 2 ways.

1. Create new object along with data.

Syntax: Object.assign(target, source);

Ex: let user4 = Object.assign(user3);

1. JSON conversion:

Convert object to stream of string format and convert stream of string to object/ array format.

Ex: let user4 = JSON.parse(JSON.stringify(user3));

JSON -> java script object notation.

**For loop:**

There are 3 ways.

1. For

Ex:

for(let i=0; i<users.length; i++) {

   document.write( users[i].name );

}

**For in: used for object.**

**Ex:**

//for in

        for(let i=0; i<users.length; i++) {

            for(let key in users[i]) {

                //item is a key

                //users[i].email -> static key

                //users[i][key] -> dynamic key

                document.write(users[i][key]);

            }

        }

        //for of

        for(let item of users) {

            //each time item is an object

            for(let key in item) {

                document.write(item[key]);

            }

        }

        //Array for loop

        users.forEach(item=>{

            for(let key in item) {

                document.write(item[key]);

            }

        });

        //Array for loop with callback

        users.forEach(callback);

        function callback(item) {

            for(let key in item) {

                document.write(item[key]);

            }

        }

**Int Que:**

1. What is an object and how many ways to rep object?
2. Copy an object?
3. What is an array and how many ways to rep array?
4. Copy an array?
5. Accessing object and array values?
6. For loop -> for, for in, for of?
7. While loop?

**Task:**

Ex on object and array and print data?

**Array Prop:** length

**Array Methods:**

**Array data manipulation functions:**

1. Add data to an array
   1. Push: it add item at last index position.

Syntax: arr.push(item);

Ex: let users = []; //array declaration

let user = {id: 1, name: "Gopi", email: "gopi@gmail.com"};

let user1 = {id: 2, name: "Siva", email: "siva@gmail.com"};

users.push(user);

users.push(user1);

for(let item of users){

    for(let key in item) {

        document.write(`${key} : ${item[key]} </br>`);

    }

}

* 1. Unshift: unshift add item to an array at 0th index position/ 1st item in an array and restructure the array items.

Syntax: arr.unshift(item);

let user2 = {id: 3, name: "Varun", email: "varun@gmail.com"};

Ex: users.unshift(user2);

* 1. Splice: splice add item to an array at specific index position and restructure the array.

Syntax: arr.splice(index, deleteCount, items);

Incase of adding item using splice use deleteCount is 0.

users.splice(1, 0, {id: 4, name: "Lakshmi", email: "vlakshmiarun@gmail.com"});

deleteCount delete items from the index position.

1. **Update item in an array**: array item updated by using array index.

Syntax: arr[index] = item;

Ex: users[3] = {id: 3, name: "Varun Dhavan", email: "varun@gmail.com"};

1. **Delete item in an array**:
   1. Pop: pop remove last index position item from an array.

Syntax: arr.pop();

Ex: users.pop();

* 1. Shift: shift remove 0th index position item from an array.

Syntax: arr.shift();

Ex: users.shift();

* 1. Splice: splice remove specific index item from an array.

Syntax: arr.splice(index, deletecount);

Ex: users.splice(1, 1);

1. Access array items: using index.
   1. Arr[index]
2. Array join function:
   1. Join: joining 2 arrays.
3. **Array search functions:** 
   1. Find -> return 1 fist matched item.
   2. Filter -> return all matched items.
   3. findLast -> return last matched record
   4. findIndex -> return index of 1st matched record.
   5. findLastIndex -> return last matched record.
   6. **Some:** return true at least 1 record matched.
   7. **Every:** return true all the records matched passed condition.
4. **Array looping function:**
   1. **forEach:** loop through the array items and not return anything.
   2. **Map:** loop through the array items and return modified array.

**JS to get/ set data to HTML elements:**

There are 2 ways to get/ set data to HTML elements in JS.

1. getElements Methods
2. querySelector methods

**getElements Methods:**

1. getElementById: based on id attribute we can get the element value.

Ex: function saveData() {

    let fullName = document.getElementById("fullName").value;

    alert(fullName);

}

1. getElementsByClassName: based on class name we can manipulate the element.

Ex: function saveData() {

    let elements = document.getElementsByClassName("form-control"); //O/p form is array list

    for(let element of elements) {

        alert(element.value);

    }

}

document.getElementsByClassName("btn")[0].classList.remove("btn-primary");

1. getElementsByTagname: based on tag name we can manipulate the element.

Ex: document.getElementsByTagName("h2")[0].innerHTML = "<i>Registration Form Saved Successfully...!</i>";

document.getElementsByTagName("h2")[0].classList.add("text-success");

1. getElementsByName: based on the element name we can manipulate the element.

Ex: document.getElementsByName("gender")[0].checked

**querySelector Methods:** 2 types.

Def: query the element based on tag name/ any attribute.

1. querySelector: using querySelector we can manipulate only 1 item at a time.

* It will return only one item.

Ex: function saveData() {

    document.querySelector("h2").innerHTML = "Hi...";

    document.querySelector("#fullName").value = "Hi...";

    document.querySelector(".btn").classList.remove("btn");

    alert(document.querySelector('[type="radio"]').checked);

}

1. querySelectorAll: it will return all the matched records. (array list)
   * We can manipulate all the items.

Ex: function saveData() {

   const elements = document.querySelectorAll("input");

    for(let element of elements) {

        alert(element.id);

    }

}

**API:** Application Programming interface.

Ex: JAVA API, .NET API, Python API, AWS API’s.

* UI – JSON server API.
* JSON Server installation.
  + npm install –g json-server
* Create a folder db and add file db.json.
* Run JSON server:
  + Go the db folder address bar and type cmd.
    - Cmd> json-server --watch <filename>
      * Ex: json-server --watch db.json
    - Cmd> npx json-server --watch db.json

Db.json file:

{

    "users": [

        {

            "id":1,

            "name": "krishna"

        }

    ],

    "products": [

        {

            "id": 100,

            "name": "lenovo"

        },

        {

            "id": 101,

            "name": "dell"

        }

    ]

}

**API Action verbs:**

1. Get
2. Post – insert (save)
3. Put – update
4. Delete

**Calling API:**

1. XML HTTP Request - JS
2. Fetch - JS
3. Ajax – Jquery
4. AXIOS – React JS
5. Etc.

**Fetch:** fetch is used to perform promised based API calls.

Fetch operations can be implemented in 2 ways.

1. Async and await
2. Callback

**Asynchronous api call:** JS is a interpreted scripting language, it will execute one by one. But API call’s will take time to execute, so we will use async and await (asynchronous) approach to wait until api call got success.

Syntax:

async function functionName() {

let res = await fetch(url); // until this line execute next line will not excute.

}

Ex:

<script type="text/javascript">

            async function getProductsData () {

                let res = await fetch("https://dummyjson.com/products");

                let data = await res.json();

                alert(JSON.stringify(data));

            }

</script>

**Int Que:**

1. What is an API?
2. What are the API action verbs/ methods?
3. How many ways to call an API?
4. What is a fetch and promise?

Task: Create an app and perform get api call?

**POST/ Save/ Insert data:**

**On which basis we need to save/ update:**

* When save id is 0
* When update id is >0
  + So, based on id we can update.
  + When update data we can store id in 3 ways on the UI.
    - Hidden field.
      * <input type=”hidden” id=”userid” value=””>
    - Window object
      * Window.userid = id;
    - Storages
* If id != 0 then update otherwise save.
* Once save or update we need to reset field values.
* Once save or update done, we need to call get call and get added/updated data.

function resetFields() {

                window.userid = "0";

                document.getElementById("fullName").value = "";

                document.getElementById("email").value = "";

                document.getElementById("phoneNum").value = "";

                document.getElementsByName("gender")[0].checked = false;

                document.getElementsByName("gender")[1].checked = false;

}

**Task:** delete API.

**Fetch method api call with callback:**

Fetch method having then is a function to perform callback.

Syntax:

fetch(url).then(res=> res.json()).then((res)=> bind).catch((exception)=>{ alert(exception) }).finally( ()=> { … });

Here **catch** block will catch the error.

**finally** block will execute if API call success or not.

**Services:** A service is a JS file to implement reusable code and to perform api calls.

Types:

1. Feature service. Ex: login service, reg service, etc.
2. Common service. Logic sharing
3. Api/ context service. Perform api calls.

**Implementation:**

* Create services folder.
  + context.service.js
  + common.service.js
  + registration.service.js

**Int Que:**

1. What is an API call and types?
2. How many ways we can perform fetch API call?
3. What is a service and types?

**Task:**

1. Reg page update and delete?

**Auto mobile application:**

1. Admin part: Create master data, and product information.
2. User part: User can buy the products.

Master data: a data which is a source of another data is called master data.

Ex: country list, states list, etc.

**Admin activities:**

1. Vehicle types
2. Categories (ele, fual )
3. Parts
4. Product/ item details

**PreventDefault:**

* By default form having default action/ event is submitting page. To stop default behavior we need to use evemt.preventDefault();

Ex:

<form onsubmit="saveData(event)">

 function saveData(e) {

                e.preventDefault();

            }

**Validation:** Checking user provided data is valid or not.

* We need to show the validation message when user provided data against to business rule.

**Types of validations:**

* Required field validation.
* Length validation (max, min)
* Email validation
* Password validation (match/ comparison)
* Etc..

**Note:** HTML form by default performs browser validations.

**AddEventListener:**

* Using JS we can attach the event to a HTML element.

Syntax:

eleRef.addEventLstener(“event”, callback);

Ex: <input type=”button” id=”save” value=”save”>

Document.getElementByID(“save”).addEventListener(“click”, (event)=>{

//logic

});

function validatePartType() {

                let isPartTypeValid = true;

                if(document.getElementById("partType").value == "") {

                    document.getElementById("partTypeValidationMsg").classList.remove("d-none");

                    isPartTypeValid = false;

                }

                else {

                    document.getElementById("partTypeValidationMsg").classList.add("d-none");

                    isPartTypeValid = true;

                }

                return isPartTypeValid;

            }

Validation div with class:

<input type="text" class="form-control" required>

<div class="invalid-feedback">

      Please provide a Spare Part Modal.

</div>

Valid class -> valid-feedback

**File uploading:** to save file to the DB, we need to be convert file to binary format.

Real time to save file:

* DB – Saving binary format
* 3rd party websites
  + Share point.
  + Cloud (AWS/ Azure).
  + Etc.

**React JS:** React JS is a JS library. It is used to develop/ build rapid application development.

**History of React JS:**

* React was developed by Facebook in 2013.
* React having 2 things:
  + React JS – Web app development.
  + React Native – Mobile app development.

**Drawbacks of Multi page App (JS):**

* Each page requires reference of CSS, JS.
* No interaction b/w pages. Each page is an independent file/ page.
* Complexity in data binding.
* Preparing raw html in JS as a string. We can’t able to find any html mistake.
* We can’t able to build reusable code block.
* Referring a HTML page in another page we need to give physical path.
* Etc…

**Features/ Benefits of React JS**:

* SPA App – Single Page Application. We can interact easily b/w the pages.
* Using SPA we can build reusable pages (component).
* No independent pages.
* Easy to bind data.
* We can prepare raw HTML directly as a component.
* By default react having routing feature.
* We can build minified files.
* No multiple references of CSS, JS files (parent component references applicable to child components).
* Easily share data b/w the components.
* React having state management. We can hold variable data.
* React having support of so many 3rd party libraries.
* Etc…

**React App Design pattern:** React fallows SOC Principle.

SOC – Separation of concerns.

Each concern is a component. We will build app with set of reusable and intractable components.

**Software requirements of React JS app:**

* Node JS
* IDE – Visual Studio Code.

**Node JS:** Node JS is a frame work to build we applications.

* Node JS using Java Script as a programming language.
* Node JS having a compiler to compile the Node JS projects.
* Node JS having package manager (npm) to add any packages/ references to node js app.
* Node JS providing set of npm commands to create, build and Run Node JS app’s.
* Installation – download the Node JS package and install.
* How to check – Control panel, using cmd.
  + Cmd – node –v
* Once Node JS installed npm, npx also installed.
  + Cmd – npm –v

**Create React JS App:**

* Cmd – npx create-react-app <ApplicationName>
* Npm install create-react-app
* Create-react-app <ApplicationName>
* Select a folder Go to the address bar and type cmd.

**Folder structure of react JS app:**

1. Node\_modules: node modules contains set of libraries, which are used to create, build and run react app.
2. Public: index.html -> this is a root html page, entire app we are having single html page, so it is called SPA.
3. Src: source folder. It contains root components and application specific feature component.
   1. Index component – entry component (a component is a JS file (JS, CSS)).
   2. App component – app component called from index component and all other components loads from app component.
4. Package json files
   1. Package.json file: It contains app dependency versions and cmd’s to run.
   2. Package-lock.json file: As a developer nothing to change in this file.
      * It contains app dependencies and physical url’s of package.

**Int Que:**

* What is a react js?
* Diff b/w MPA and SPA?
* Features of react js app?
* React js App set up?

**Task:**

* Set up react JS app?

**Component:** components are the building blocks of react js app.

* A component is a JS file.
* A component contains a feature of an application (design, code binding and data).
* Components are created in 2 ways.
  + Functional components
  + Class Components
* Types of components.
  + Feature component
  + Shared component
  + Common component
  + Atomic component
  + Etc…
* Root component -> index component (index.js file, it called from index.html file).
  + App.js called from index.js and we can call our components from app component.

**React.StrictMode: Avoid c**reating a variable without declaration.

**Note:** we can’t able to write raw html in the JS file.

**JSX – Java Script XML:** JSX helps to write a page designing. Generally, page designing implemented by HTML, but we can’t able to write HTML in JS file. So, react team introduced concept of JSX.

**Virtual DOM:** Virtual Dom converts React JS specific syntax like JSX to required format like HTML, etc.

* Virtual Dom preparing before the real DOM.
* Virtual DOM is a copy of real DOM.

**Functional Component:** Creating a component with JS function is called functional component.

A functional component can be created in 3 ways.

1. General function with return type

**Ex**:

Export function App() {

return (<div></div>)

}

**Syntax**:

export function <FunctionName>() {

return JSX;

}

1. **Variable defined, anonymous function:**

Ex: const App = function() {

return <div>…</div>;

}

Syntax: const VariableName = function() {

Return JSX;

}

1. **Arrow function:**

**Ex:** const VariableName = () => {

return <div></div>

}

**Export:** export is a keyword in JS, it helps to export a JS function to be use in another JS file.

**Rules:**

* A functional component name must be Pascal case.
* A functional component must be return.

**Create custom component:**

S1: create a components folder and create a components.

Src-> components -> products folder -> project.js, product.css

**Product.js**

export function Product() {

return <h2>Products</h2>

}

**App.js:**

S1: import component

import {Product} from “./components/products/product”;

S2: Use component in JSX.

<Product></Product>

**Int Que:**

1. What is JSX?
2. What is Virtual DOM?
3. What is export keywords?
4. What is component?
5. Component types and creational types?
6. How to import a component?

**Task:**

1. Create a react js app and create some component?

**Adding bootstrap to react JS app:**

**Adding a package to react JS app:**

Cmd> npm install/i –g <packagename>

-g for global.

Once adding package below changes we can notice:

1. Package version will be added in package.json file
2. Package version and physical URL will be added in package-lock.json file
3. Package physical files will be added in node\_modules

To use package after adding we need to give reference of package in required file.

**BS:** >npm install bootstrap

BS is a css file we can add in app.js/ app.css as well.

App.js -> import 'bootstrap/dist/css/bootstrap.min.css';

App.css -> @import url('bootstrap/dist/css/bootstrap.min.css');

**Styling in React JS:**

* Style – in react js styles will be added as an object.

Ex: <h2 style={{color: “red”}}>VKC Trainings</h2>

* Camel case styling: backgroundColor.
* For background-color -> will get console exception.
* **Class –** className

**Data Binding:**

1. Interpolation/ string data binding -> by using { variableName }, <h2>{ course }</h2>
2. Event binding -> eventName={ ()=>{ function()} }

Ex: onClick={ ()=>{ saveData()} }

**Ex:**

const App = () => {

  let header = "VKC Trainings";

  let headerStyles = {color: "red", backgroundColor: "green"};

  return (

    <div>

      <h2 style={ headerStyles }>{ header }</h2>

      <input type='button' className='btn btn-primary' value="Click me!" onClick={ (event)=>{ alert(1) } }></input>

      <Product></Product>

    </div>

  );

}

**Styling types**:

1. Inline -> style={{colo: “red”}}
2. Internal -> create a variable and assign data (data as an object and bind)

(internal means within the same file we can re-use).

1. External styling – create a separate css file and import in js file.

**Int** Que:

1. How to add a package in react js?
2. What are the changes will happen when adding package?
3. Styling in react JS?
4. What is camel case styling?
5. How to implement data binding in react JS?
6. How to add class?

Task:

1. Implement products grid?

|  |  |  |  |
| --- | --- | --- | --- |
| Action | JS | RJS – Functional component | RJS – Class Component |
| On load | Onload event | useEffect() | componentDidMount |
| Get element reference | Document.getElements,  querySelector | ref, useRef | ref, createRef |
| Hold data for next action/ store data | Window.varaible,  JS storages  Hidden varaibles | useState() | this.state,  this.setState()  Redux |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**React JS functional hooks:** React JS functional hooks are used to perform a task in functional component work flow.

* useEffect
* useState
* useRef
* useMemo
* useReducer
* useNavigate
* useForm
* etc….

**useEffect:** useEffect is a react JS functional hook. It is used to perform functional components onload actions. For ex, bind data, show something once component loads, get data and bind, etc..

**Syntax**: useEffect(callback, [dependencies]);

**Ex:** useEffect(()=>{ getCategoryTypes(); }, []);

**useRef:** useRef is a react js functional hook. It is used to get reference of an element and we can manipulate the element.

**Syntax:**

S1: create a ref variable.

Ex: let tableDataRef = useRef();

S2: use ref/ attach ref to html element.

Ex: <tbody ref={ tableDataRef }></tbody>

S3: access ref.

tableDataRef.current

(document.getElementById(tableDataRef))

tableDataRef.current.innerHTML = tableData;

**useState:** useState is a react js functional hook. It is used to maintain state of a variable.

Syntax: const [stateVarible, setStateVariable] = useState();

Note: in api call normal variable is not capable to bind data, we need to use useState.

const [categories, setCategories] = useState([]);

 setCategories(await res.json());

<tbody ref={tableDataRef}>

                   {

                    categories.map((item)=>

                        <tr>

                                <td>{item.id}</td>

                                <td>{item.category}</td>

                                <td>

                                    <i>&#x270D;</i>

                                </td>

                            </tr>

                    )

                   }

                </tbody>

**Int Que:**

1. What is a functional hook?
2. What is a useEffect?
3. useState?
4. useRef?

**Task:**

1. use the functional hooks create save and get forms?

**Higher order function:** A function taking another function as a parameter.

Ex: array.forEach, arrrayMap, etc…

**Higher order components:** A component taking another component and produce new component is called HOC.

Ex: index.js, app.js, …

* Parent component is a HOC.

**Nested component:** A component calling inside of another component is called nested component.

* Called component is HOC/ parent component.
* Calling component is a child component.
* By using nested component/ **parent & child relation component**, we can create reusable components.

**Shared components – Reusable components: (Dynamic component):** A component will be created dynamically based on the data.

* Components -> shared -> add component folders.

**Note:** In real time a component filename can be created in 3 ways.

1. Filename.js
2. Finename.component.js
3. Filename.jsx (specific to react js jsx file extension)

Ex:

export const Grid = ({header, customStyles}) => {

    return (

        <div>

            <h2 style={customStyles}> { header } </h2>

        </div>

    )

}

<Grid header="Products Information" customStyles={{ color:"white", backgroundColor: "green"}}></Grid>

**Props**: props are used to pass data from parent to child and child to parent component.

* Passing object as i/p parram: ex: {header, customStyles}
* Passing props as i/p. props rep above object
* Props.children -> passing JSX nesting to another component.

<Grid header="Products Information"

                  customStyles={{ color:"white", backgroundColor: "green"}}

                  >

                    <thead class="table-primary">

                        <tr>

                            <th>ID</th>

                            <th>Categoty</th>

                            <th></th>

                        </tr>

                    </thead>

</Grid>

**Int Que:**

1. HOC?
2. Nested component?
3. How to pass data b/w components?
4. What is a shared/ reusable component?
5. What is a props? How many ways to pass data?
6. Props.chldren?

**Task:**

* Create nested components and pass data?

**Passing data from child to parent:** Using props.event();

**Parent to child -> props.property**

**Child component: grid component.**

<i onClick={ ()=>{ editData(item);  } }>&#x270D;</i>

function editData(item) {

        //alert(id);

        props.editGridItem(item); //exposing event from property.

}

**Parent component:**

<Grid editGridItem= { (item)=>{ editItem(item)  } }

function editItem(item) {

        alert("Parent component");

        alert(JSON.stringify(item));

    }

**Routing/ Navigation:** In SPA routing helps to navigate b/w the components/ pages.

* Install react-router-dom

S1: Create menu component.

import { Link } from "react-router-dom";

export function Menu() {

    return (

        <div className="bg-dark p-2">

            <label className="text-light">

                <Link to="/home" className="text-light px-2">Home</Link>

            </label>

            <label className="text-light">

                <Link to="/product" className="text-light px-2">Products</Link>

            </label>

        </div>

    )

}

S2: Create app router component to implement routing.

import { Route, Routes } from "react-router-dom";

import { Home } from "./components/home/home";

import Product from "./components/product";

export function AppRouter() {

    return (

        <Routes>

            <Route path="/home" element={ <Home></Home> } ></Route>

            <Route path="/product" element={ <Product></Product> }></Route>

        </Routes>

    )

}

S3: In app component add the parent element as browser router element and call app router inside of it.

<BrowserRouter> -> import from react-router-dom

      <Menu></Menu> -> we created menu component

      ….

      <AppRouter></AppRouter> -> we created approuter component

</BrowserRouter>

S4: in menu component instead of anchor tag use **Link** tag from the react router dom and instead of href us **to**.

**Routes:** Routes is a parent element to define the route.

**Route:** Route is used to define the browser router path and respective component to be load.

**BrowserRouter:** Add this in app.jsas a parent element. It defines the cleanest url.

* It encapsulates the routes.

**Int Que:**

1. How to pass data from child to parent component?
2. What is routing?
3. How to implement routing in react JS?
4. Explain about BrowserRouter, Routes, Route?

**Routing types:**

1. Path routing: Navigate to specific path (Ex: home, products)
2. Default routing: Once the domain loads load the default page. (Ex: [www.website.come](http://www.website.come) -> home page).

Rep by -> path=”/”

1. Routing with params: In the routing we can pass data as dynamic. For example id

Ex: [www.website.com/products/1](http://www.website.com/products/1) -> here 1 is param.

S1: create product details with static data.

S2: create the route for product details in app router file.

S3: add the menu in menu component.

S4: add the route with params in the app router file.

<Route path="/product-details/:id" element={ <ProductDetails></ProductDetails> }></Route>

Here :id is a dynamic data to routing.

S5: Add the grid click row action.

<tr onClick={ ()=> { props.gridRowAction(item.id) } }>

(Here grid is a shared component, exposing the grid row click action).

S6: Navigate to product details from call back action.

* Navigate from JS code.

**useNavigate:** useNavigate is a react JS react-router-dom hook. It is used to navigate to a router from the JS code.

Ex:

import { useNavigate } from "react-router-dom";

const naviagete = useNavigate(); -> create a variable to navigate a router.

Use the navigate.

naviagete("/product-details/"+id);

**Get id from routing params:**

**useParams:** useParams is a react JS react-router-dom hook. It is used to get the params from the router.

import { Link, useParams } from "react-router-dom";

const params = useParams(); (or) const { id, name } = useParams();

alert(params.id);

**Not defined routing – not found:** Once the user is looking for specific routing which is not found then show the not found component.

* Path rep by \*

<Route path="\*" element={ <NotFoundPage></NotFoundPage> }></Route>

**Int Que:**

1. Types of routings?
2. Routing with params?
3. useNaviagate()?
4. useParams()?
5. Not found route?

**Class:** A class is a user defined data type.

* Class was introduced in JS in ES6.
* By using class we can implement OOPS concepts.
* But class in JS supports partial OOP’s concepts.

Syntax:

class ClassName {

constructor() {

}

Class variables

Class methods

}

**Constructor:** Constructor is used to define the class fields (variables).

**Types of constructors:**

1. Default constructor: If we are not created a constructor at run time system will create a constructor.
2. Constructor without params.
3. Constructor with params.

**OOPS Principles:**

1. Encapsulation: Encapsulating the variables and functions inside of a class is called encapsulation.
   * Creating a class itself we can achieve encapsulation.
2. Abstraction: Exposing required information and hiding potential information is called abstraction.
3. Inheritance: Creating a reusable class with parent and child relation is called inheritance.
4. Polymorphism: Poly means many forms. Deals with method level (method overriding & overloading).

**Class inheritance:**

Class ClassName extends AnotherClassname {

}

**Skeleton of Class Component:**

**import React from ‘react’;**

**export class ClassName extends React.Component {**

**constructor() {**

**super();**

**this.state = { key: value, … };**

**}**

**render() {**

**return JSX;**

**}**

**}**

**React.Component:** It is a base component to create a class component. React.Component provides event life cycle methods to build class component logics.

**super():** super method is a base class constructor. It is used to pass data from child class to base/ parent class.

**this.state:** In React class components are state full components. To implement state by using this.state.

**this.setState (key: value):** this.setState method is used to set state of a variable.

**render():** render is a class component event life cycle. It is used to render the JSX.

this.state = {

            categories: []

        }

import React from "react";

import { Grid } from "./shared/grid/grid.component";

export class ProductClass extends React.Component {

    constructor() {

        super();

        this.state = {

            categories: []

        }

    }

    url = "http://localhost:3000/categoryTypes/";

    products = [

        {

            id: 1,

            name: "Laptop - Dell",

            price: 32000

        },

        {

            id: 2,

            name: "Laptop - Lenovo",

            price: 30000

        },

        {

            id: 3,

            name: "Laptop - Acer",

            price: 28000

        }

    ];

    async getCategoryTypes() {

        let res = await fetch(this.url);

        this.setState({categories : await res.json()});

    }

    componentDidMount() {

        this.getCategoryTypes();

    }

    //In class component create a function without function keyword.

    render() {

        return (

            <div>

                <h2>Products - Class Component</h2>

                <table className="table table-hover table-striped table-bordered">

                <thead>

                    <tr>

                        <th>ID</th>

                        <th>Name</th>

                        <th>Price</th>

                    </tr>

                </thead>

                <tbody>

                    {

                        this.products.map((item) =>

                            <tr>

                                <td>{ item.id }</td>

                                <td>{ item.name }</td>

                                <td>{ item.price }</td>

                            </tr>

                        )

                    }

                </tbody>

            </table>

            <Grid header="Products Information - Class Component"

                  customStyles={{ color:"white", backgroundColor: "green"}}

                  categories={ this.state.categories }

                  >

                    <thead className="table-primary">

                        <tr>

                            <th>ID</th>

                            <th>Categoty</th>

                            <th></th>

                        </tr>

                    </thead>

                </Grid>

            </div>

        );

    }

}

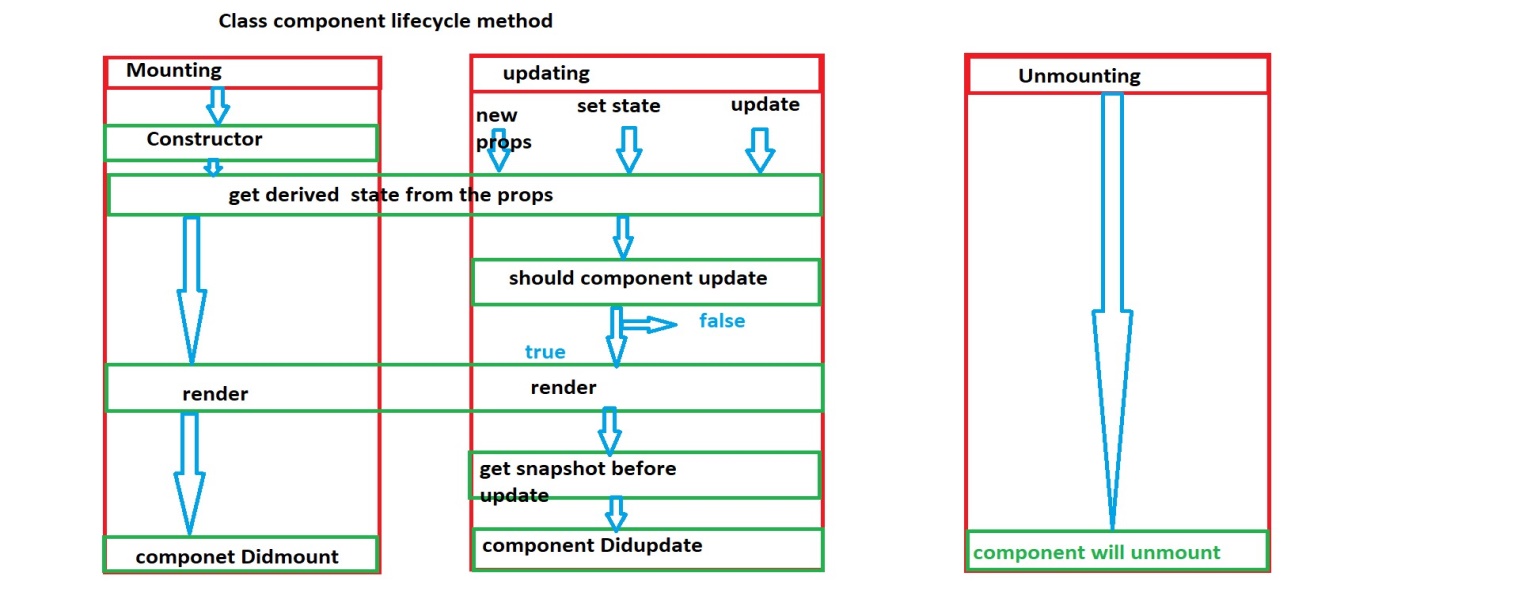
**this:** this is a keyword it is used to access class members (variable, method) with in the class.

**Note:** in class let, var, const and function keywords don’t use.

**Class component life cycle events:**

There are 3 phases in class component life cycle.

1. Mounting (loading)
2. Updating
3. Un Mounting



**Constructor**: define onload varaibles.

**getDerinedStateFromProps**: it will take state, props as i/p params. This is a static function.

It will call in mounting, updating phases.

**Render:** loads JSX code.

**componentDidMount:** onload actions. API call, etc..

**shouldComponentUpdate:** it will take a decision to update the component (render the JSX or not).

Return true -> component will render. False -> component will not render.

**getSnapShotBeforeUpdate:** before data is going to update in real time we can take data backup/ copy.

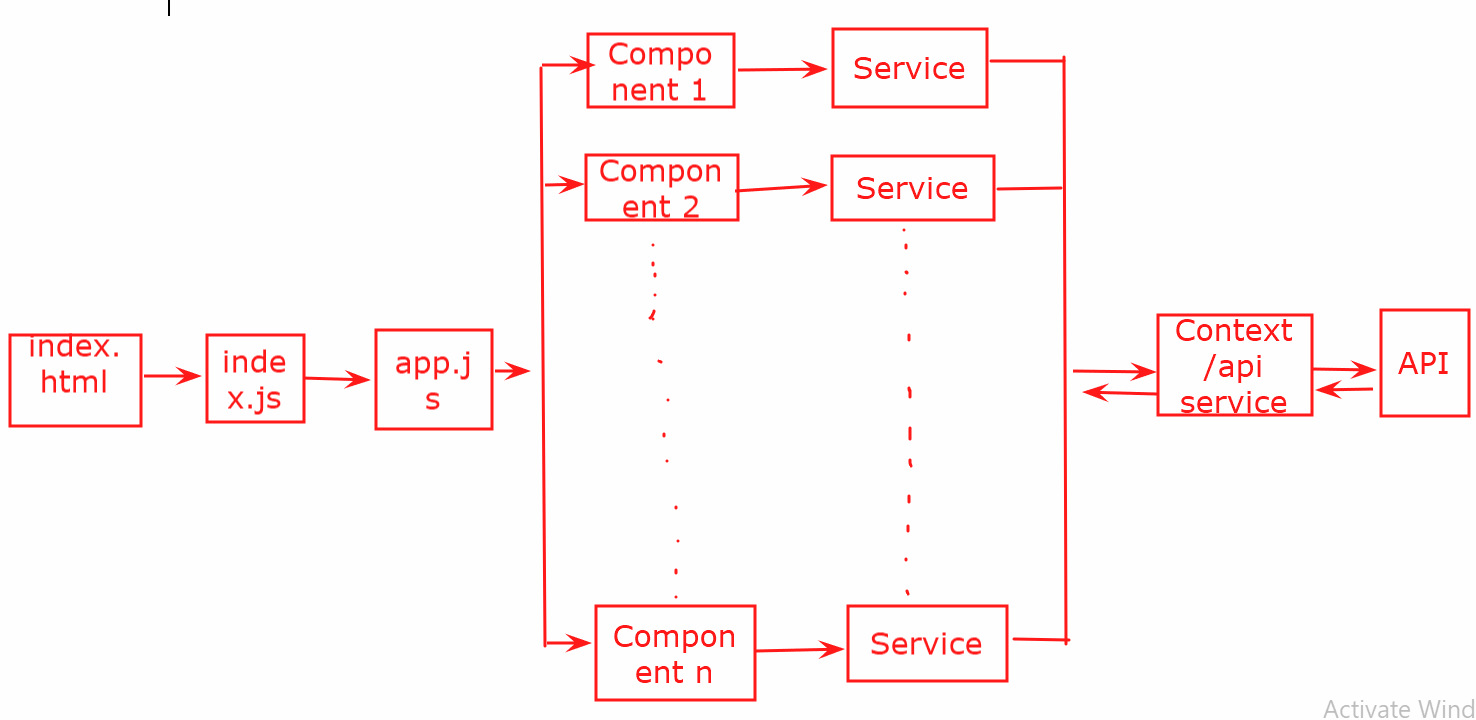
**componentDidUpdate:** it will call in the updating phase and this is a final event in the updating phase.

**componentWillUnmount:** only one event in the un-mounting phase. It will call when the component is going to unload.

**Pure Component:** Type of a class component. It will implement shouldComponentUpdate function directly.

* React.PureComponent.

**React JS project app flow:**



**React JS Projects Steps:**

S1: Create a project.

S2: Add BS.

S3: Add assets.

S4: components -> shared, atomic, layout -> header, footer, menu

S4: Add react-router-dom

S5: create app.router.jsx

S6: Services -> context.service.js, storage.service.js

**React Fragment:**

React Fragment is used to define parent element. Situation: every component JSX needs one parent element.

React.Fragment is a dummy element. Used as a parent. It doesn’t have any styles.

There are 2 ways to rep:

1. <></>
2. <React.Fragment></React.Fragment>

**Context Service/ API Service:**

**AXIOS:** AXIOS is a library. It is used to perform API calls and get the response.

* Install axios. > npm I axios
* Axios exposing action methods (get, post, put & delete) to perform API calls.
* Get -> get the data
* Post -> insert/ save data
* Put -> update data
* Delete -> delete data. But in real time project we will not delete the data, bcs compliance issue. Instead of delete we will change the status (active -> true/ false) of the record.

import axios from "axios";

export function getData(url) {

    return axios.get(url);

}

export function saveData(url, data) {

    return axios.post(url, data);

}

export function updateData(url, data) {

    return axios.put(url+data.id, data);

}

export function deleteData(url, id) {

    return axios.delete(url+id);

}

**Diff b/w axios and fetch:** axios by default implements options and convert api response to json format.

**Environment.config.js file** -> it will have list of configurations.

* Src-> environment -> environment.config.js

const Configurations = {

    apiUrl: "http://localhost:3000/"

}

export default Configurations;

**In menu.service.js file:**

import Configurations from "../environment/environment.config";

const url = Configurations.apiUrl+ "parts/";

import { getData } from "./context.service";

export function getPartsData() {

    return getData(url);

}

Exposing functions from js file as object:

const api = {

    getData: (url)=>getData(url),

    saveData: (url, data)=>saveData(url, data)

}

export default api;

One api call depend on another api call?

1. Call api in success of another api call
2. Use async and await

**Module in JS:** A module providing set of business logics. In JS create ting a JS file with functions and exposing required function from the JS file will achieve modularity.

* This module file will be imported in another files and we can use.

**Conditional rendering:** Based on a condition display an element/ component.

(condition -> comparison of 2/ more values).

**Syntax**: { condition && <Component></Component> }

Note: Whenever data binding is there better to use conditional rendering.

**Ternary operator:** if and else case we can create as ternary condition.

Condition ? value1: value2

Ex: <ul className={ item.subMenuList.length > 0 ? "dropdown-menu": ""}>

React JS project validation implementation:

 const [nameValid, setNameValid] = useState(true);

    const nameRef = useRef();

 function saveData(event) {

        event.preventDefault();

        validateNamefield();

    }

    function validateNamefield() {

        if(nameRef.current.value != "") {

            setNameValid(true);

        }

        else {

            setNameValid(false);

        }

    }

 <form className="container" onSubmit={(event)=>{

            saveData(event)

        }}>

            <div className="form-grom">

                <label>Full Name</label>

                <input type="text" className="form-control" ref={nameRef} onChange={()=>{ validateNamefield() }}></input>

                {

                    !nameValid && <div className="text-danger">Name field is required.</div>

                }

            </div>

**useTransition:** this hooks will helps to notify the transition completed or not.

Syntax: const [isPending, setTransition] = useTransition();

setTransition(()=>{

…

});

Once transition executed then isPending becomes true.

Use case: display loder.

**SOLID Principles:**

* S – Single responsibility
* O – Open & closed principle
* L – LIVOSK substitution principle
* I – Interface segregation principle
* D – Dependency Inversion

**React hook form:** This lib will validate the form and save the form data.

* npm i react-hook-form

Steps: import useForm hook and create an object.

S1:

    const {

        register, //register the controls

        handleSubmit, //save data

        formState: { errors }, //validation

      } = useForm();

S2: register the element.

<input type=”text” {…register(‘fullName’)}/>

S3: add error message.

{ errors.fullName && <div class=”text-danger”>Please enter field.</div>}

<input type="text" {...register('fullName', {required: true, minLength: 3, maxLength: 5})} className="form-control"></input>

                {

                    errors.fullName && errors.fullName.type == "required" && <div className="text-danger">Name field is required.</div>

                }

                {

                    errors.fullName && errors.fullName.type == "minLength" && <div className="text-danger">Name must have minimum 3 char's.</div>

                }

                {

                    errors.fullName && errors.fullName.type == "maxLength" && <div className="text-danger">Name max length is 5.</div>

                }

S4: add handleSubmit in form submit.

onSubmit={handleSubmit((data)=>{ saveData(data) })}

S5: Update the form field.

Using setValue – add in useForm();

setValue(key, value);

setValue(“fullName”, “krishna”);

**React notification/ testify message:**

npm i react-toastify

**Int Que:**

1. useTransition hook?
2. SOLID principles?
3. React hook form?
4. Toastify message?

**Task:** Create reg page and save data and display toast message?

**Authentication & Authorization:**

**Authentication:** Validating a user.

* User having the access to the website or not.
* Types:
* Form Authentication – login with username and password.
* Windows authentication – Specific n/w accessing the url.
* Token based authentication – For user credentials generating the token and validating.
* Single Sign On (SSO – office mail/ Gmail/ specific account credentials to be used for another account).

**Storages:** In web app, browser storages are used to store some amount of website related information.

There are 3 types:

1. Local storage (Permanent store)
2. Session storage (session store)
3. Cookies (specific time period store)

**Local Storage:** Local storage stores the data permanently in system, until user clear the data manually.

**Session Storage:** Stores the data in particular session (open the website to close the website).

**Set the data:**

localStorage.setItem(key, value);

* Always value should be a string. In-case of object/ array store the stream of string data.

**Get the data:**

sessionStorage.getItem(key);

**Remove the data:**

localStorage.removeItem(key);

**Clear the storage:**

localStorage.clear();

**cookies:** stores the application specific information for certain time period.

document.cookie = “key=value; expires=date; path=’/’;”;

Ex:

let date = new Date();

            date.setHours(date.getHours()+1);

            document.cookie = `UserDetails=${JSON.stringify(user)}; expires= ${ date }; path='/'`;

get cookie:

document.cookie

**Redux:** Redux is a JS lib.

* Redux is used for state mgt lib.
* Redux is used for notification b/w the (data transfer) independent components.
* Redux works on broadcast and subscriber principle.
* Install> npm install redux

**S1: import** createStore from redux

import { createStore } from "redux";

S2: Create reducer function

/State holds the data

    //Action updates the data to state based on the action type.

    //Action is an object type, it have 2 keys {type: "", data: ""}

    const reducer = (state, action) =>  {

        switch(action.type) {

            case "products":

                return { ...state, products: action.data}; //compy the prev state and add the new values in to the state

            case "recordsExists":

                return { ...state, recordsExists: action.data}

        }

    }

S3: Create store variable.

const store = createStore(reducer);

S4: Subscribe (in useEffect) store to get data.

store.subscribe(() => { //When the store values changes subscriber will be called

        setProducts(store.getState()?.products);

});

S5: dispatch action to store.

let action = {type: "products", data: data }

store.dispatch(action);//dispatch send data to store and then to reducer function

* Dispatch triggers the reducer function of a store.

**useContext:** useContext is a react js functional hook. It is used pass data from parent to child component (drill down).

* Use case useContaxt is used to pass user details from parent to all child components.

S1: create user content service and create the context variable.

import { createContext } from "react";

export const UserContext = createContext();

S2: pass data from userContext to ite’s nested component.

<UserContext.Provider value={user} > -> get the user by subscribe the store.

….

</UserContext.Provider>

S3: get the user context value in the component.

const userDetails = useContext(UserContext);

**@reduxjs/toolkit:** create store was depreciated in redux instead of this use the **@reduxjs/toolkit.**

import { configureStore } from "@reduxjs/toolkit";

export const store = configureStore({reducer: reducer});

**useReducer:** useReducer is a reactjs functional hook. It was developed from redux concept.

* useReducer is used to maintain complex state.
* Unlike using useState, in useReducer we can create single state variable.

S1: import useReducer.

import { useReducer } from "react";

S2: create reducer function.

S3: create reducer state and dispatch variable.

const initialValues = {products: [], recordsExists: false};

const [state, dispatch] = useReducer(reducer, initialValues);

* useState maintain single variablestate, but useReducer maintain multiple variables state values.

S4: set data to a state: using dispatch function.

dispatch({type:”products”, data: data});

S5: accessing state values in JSX.

state.products

**useMemo:** useMemo is a hook. It is used to memorize the values.

* Means it will hold the previous value and generate new value.
* Use case: filtering data

const memoVerial = useMemo(callback, dependencies);

When the dependencies changed memo function will be called.

    const filtertedProducts = useMemo(()=>{

        return state.filterValue ? state.products.filter(item=>item.sparePart.toLowerCase().indexOf(state.filterValue?.toLowerCase())>-1) : state.products;

    }, [state.products, state.filterValue]);

**Destructor:** Assigning object to a multiple variables.

Ex:

const {id, name} = useParams();

**Note:** When the route path parameter change component will not reload, to get the data again pass the route parameters as dependencies to useEffect.

**Share Component:** A component shared by the multiple components.

* Shared component works on parent and child relation.
* Shared component fallows single responsibility.

Ex:

import { useEffect } from 'react';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

export function Notification(props) {

    useEffect(()=>{

        if(props.message) {

            toast(props.message);

        }

    }, [props.message]);

    return (

        <ToastContainer></ToastContainer>

    )

}

<Notification message={message}></Notification>

**Duplicate records:**

* Loop the array and write the condition in the array.

**Int Que:**

1. Shared component?
2. Multiple routing params?
3. Destructor?
4. Duplicate records?

**Atomic Component:** Atomic component is a part of a component.

**API calls:**

* Api calls can perform in

1. Parallelism: calling the api’s at a time.
2. Concurrency: one api dependent on another api call the one after another.

**Hook:** is used to perform a task in the functional component life cycle.

* Hook starts with use.

**addEventListener:** instead of adding action to an element in html page we can add in js file using addEventListener.

selector.addEventListener(action, callback);

<input type='text' id="ele" className='form-control m-2'></input>

  function onload()

  {

    document.getElementById("ele").addEventListener("blur", (event)=>{

      alert(event.target.value);

    })

  }