**HTML(Hypertext Markup Language)**

- html is used to structure a website . It is like a skeleton of your webpage

- Markup refers to HTML code

- keyword ‘**!DOCTYPE’(Document Type Declaration)** tells the browser about the type or version of HTML used

**meta**

- <meta > tags defines metadata about an HTML document . **Metadata** is information(data) about data .

- link , meta , title and style are Metadata elemets , go inside head tag

- <meta> tags are pieces of information you use to tell the Search engine and those viewing your site more about your page and the information it contains .

- <meta> tags are used for SEOs

**Empty and Non-empty tags**

- Empty tags don’t have any contents inside them and don’t have closing tags . ex: <br> , <u> , <img>

- Non-empty tags have contents inside it and have closing tags. ex: <head> , <body> , <span> , <title>

**Tables**

<table>

**<thead>**

**<tr>**

**<th>**name</th>

<th>id</th>

<th>roll</th>

</tr>

</thead>

**<tbody>**

**<tr>**

**<td>**rahul</td>

<td>101</td>

<td>2</td>

</tr>

<tr>

<td>rahul</td>

<td>101</td>

<td>2</td>

</tr>

<tr>

<td>rahul</td>

<td>101</td>

<td>2</td>

</tr>

</tbody>

</table>

**Forms , Inputs , checkbox&radios**

<form **action="backend.js" method=”get/post”**> <!--***action* attribute sends the form data to the backend file i.e backend.js**-->

<div>

**<label for="name">** Enter name : **</label>**

<input type="text" name="myText" i**d="name">**

</div>

<br>

<div>

<label for="role"> Enter role : </label>

<input type="text" name="myRole" id="role">

</div>

<br>

<div>

<label for="email"> Enter email : </label>

<input type="email" name="myEmail" id="email">

</div>

<br>

<div>

<label for="date"> Enter date : </label>

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

</div>

<br>

<div>

<label for="bonus"> Enter bonus : </label>

<input **type="number"** name="mybonus" id="bonus">

</div>

<br>

<div>

**<label for="myEligibility">**Are you eligible**</label>**

<input **type="checkbox**" name="check" **id="myEligibility**">

</div>

<br>

<div>

Gender:

*<label for="m">male</label>*

*<input* ***type="radio"*** *name="myradio" id="m">*

*<label for="fm">female</label>*

*<input type="radio" name="myradio" id="fm">*

</div><br>

<div>

Write about you : <br><**textarea** name="myText" cols="30" rows="10"></textarea>

</div>

<br>

<div>

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

<input **type="reset"** value="reset">

</div>

</form>

<**label> -** acts as a caption for the specified element .It increases the clickable area as clicking the label(text) as well activates input as well **Or** label associates label-text with the input-element

- radio-buttons and checkboxes should be wrapped under specific labels

- **radio button** is an input-type which is used to make only one-option selection while **checkboxes** are used to make multiple-selections .

- All related radios or checkboxes should have same **name** attribute to create a radiobutton group or checkbox group

- **required** attribute is used to make the input compulsory

- **checked** attribute is used to make the radios and checkboxes checked by default

- **Placeholder** is a text which is displayed in the input-filed before the input is taken

**<input type=”text” placeholder=”Input username”>**

- **method=”get/post”** attribute specifies how to send form-data(the form-data is sent to the page specified in the action attribute)

- The form-data can be sent as URL variables (with method=”get”) or as HTTP post transaction(with method=”post”)

- **In get request all the parameters are visible in url**

**<select> and <option> tags**

**<select** name="car" id="myCar"> #**This tag creates a dropdown menu consisting the given options in option tag**

**<option** value="ind" **checked**>Indica**</option>**

<option value="tata">Tata</option>

**</select>**

**Block and Inline elements**

**Block element** – A block element always has a top and a bottom margin and always starts on a new line

* It doesn’t tolerate any other element next to it . Ex:- <h> , <p> , <div> , <form> , <footer>
* takes up the full width(of its container) available .
* to make any element as block we use display property as **display:block**

**Inline elements -** These elements take as much width as necessary and they don’t start on a new line . ex: <span> , <a> , <br> , <button>, <b>

**class** and **id(identifier)**

- Classes and IDs are **elements selectors** . We can grab/access any element through it

[class\*=”test”] – it is an attribute selector , matches every element whose class value contain this value(i.e test) and apply its properties to all matched elements .

\* - this will select all elements

**-** classes and ids are the elements selectors which identify an element based on the value assigned to these attributes

- **only one id selector** is attached to an element Or an id is attached to only a single element, where as ,

- **multiple class selectors** can be attached to a single element Or a class can be attached to multiple elements

**- ‘id’** selector has higher precedence than **‘class’**

- elements can be grabbed by their specific ids and classes in order to perform any operation with that element

- Specifically **, Id’s** are used for webpages to interact with Javascript not for styling unlike classes .

- using classes , prevents the repetition of same style and properties with multiple elements

**Emmet**

- it is a shortcut to type full HTML code

ex:- <!-- Emmet Shortcuts -->

**div>ul>li**

<div>

<ul>

<li></li>

</ul>

</div>

**div+p+bq**

<div></div>

<p></p>

<blockquote></blockquote>

**div.newClass**

<div class=”newClass”></div>

**HTML Entities**

- An Entity is a piece of a text(“String”) that begins with an ampersand(**&**) and ends with semi-colon(**;**)

- These are used to display reserved characters(which would otherwise interpreted as HTML code) and invisible characters (like non-breaking spaces)

- Entities are also used to display such characters which are absent in a standard keyboard like , copyright etc

ex:-

|  |  |
| --- | --- |
| Entity | character |
| &amp; | & |
| &lt; | < |
| &gt; | > |
| &quot; | “ |
| &copy; | © |
| &nbsp; | Non-breakable space |
| &pound; | # |
| &empty; | Empty-character |

**HTML semantic elements**

- Semantic tags more specifically describes its usage by its name only . Ex : <footer> , <section> , <article>

– These elements clearly describe it meaning to both browser and the developer

ex:- <header> , <nav> , <article> , <section> , <footer> etc.

<article> - It is an HTML semantic , which specifies independent and self-contained content . It is used to contain articles types of materials like , Blogs , news-article

**& CSS(Cascading Style Sheets)**

- CSS gives styles or designs to the raw HTML or to our webpage

**Types of CSS**

1 . Inline - CSS is added to the elements directly using style attribute . ex:- <p style=”color:blue;”>

2 . Internal - CSS is kept inside the head tags in <style> tags

3 . External - CSS is kept in a separate .css style sheet

**CSS Specificity/precedence**

0 . !important [apply this keyword to next of any property to make it highest priority]

1 . Inline styles

2 . Internal / external styles [as per their corresponding location i.e styles written at bottom will get higher priority]

3 . Id

4 . classes , attributes / pseudo-classes

5 . elements . pseudo-elements

6 . \*

- **Inline CSS** has higher priority than embedded and external CSS

**- ‘!important’** keyword is used to prevent overriding any CSS

**Selectors**

- Selectors are used to find or select the HTML elements you want to style

- Selectors are used to target any HTML elements

- CSS selectors can be divided into 5 categories

1 . Simple selectors(selects elements based on name , id , class)

2 . Combinators selectors(select elements based on a specific relationship b/w them )

3 . Pseudo-class(select elements based on a certain state)

4 . Pseudo-elemnts (select and style a part of an element)

5 . Attribute (select elements based on an attribute or attribute value)

|  |  |  |  |
| --- | --- | --- | --- |
| **Simple Selectors** | name | example | description |
| #id | id selector | #id{color:red;} |  |
| .class | class selector | .class{color:red;} |  |
| element | element selector | p{color:red;} |  |
| \* | universal selector | \*{color:blue;} | selects all the elements on the page |
| element1 , element2 , element3 | grouping selector | h1 , p , h2 {color:red;} |  |
| element.class |  | h1.class{} |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Combinators** | name | example | description |
| space | descendant selector | div p{ properties} | properties applied over each child p inside it |
| > | child selector | div > p{properties} | properties applied to the the direct p child of div |
| + | adjacent sibling selector | div + p{properties} | applied to the p which is adjacent and just after a div |

**Pseudo class**

- Used to define a special state of an element

**selector:psudo-class** {propert:value;}

1 . a:link - property applied to un-visited link

2 . a:visited - visited-link

3 . a:hover - mouse over link

4 . a:active - selected link

5 . li:**nth-child(3) -** 3rd li will be styled

more examples

**li:nth-child(2n+0) -** value of n starts from 0 and so on . Hence , every second li will be styled .

**li:nth-child(3n+3) -** every 3rd li will be styled

**li:nth-child(odd) -** odd positioned elements style

**li:nth-child(even) -** even positioned elements style

6 . p : first-child - selects and style every p element that is first child of it’s parent

**Attribute selector**

- Used to style HTML elements that have specific attributes or attribute values

**a[target]** - All the anchor elements with ‘target’ attribute will be styled

**a[target=”\_blank”]** - with specific value of attribute

**[attribute\*=”value”]** - ex:- **[class\*=”container”]{properties}** - all elements with given class be selected

**Pseudo elements**

**-** used to style specified parts of an element

**selector :: pseudo-element** {propert:value;}

**h1 :: after**( content : value;) - It will add content value at the end of h1 element

**h1 :: before**( content : value;) - It will add content value before of h1 element

**Colors**

1 . **RGBA**(red(0-255),green(0-255),blue(0-255),opacity(0-1))

black - rgb(0,0,0)

white - rgb(255,255,255)

2 . **Hex color**

black - #000000 white - #ffffff

**CSS Properties**

**Border**

**border** : 4px(width) solid(style) green(color)

**border-radius** - defines the radius of element’s corner

Grabbing specific corners’ radius:-

**border-top-left-radius :** 10px **border-top-right-radius**: 4px

**outline : width style color** : Outline is a line that is drawn around elements , outside the borders , to make the element “stand out”.

**Background**

**background-image :** url(‘imgae.jpg’)

**background-repeat** : repeat-x / repeat-y / **no-repeat**

**background-position :** x-axis y-axis (left top/right bottom/center center/right top/10px 20px)

**background-size :** auto(default) / custom size/ cover

**background-attachment :** scroll/fixed/...etc

**background** : bg-color bg-image bg-repeat bg-position/bg-size bg-attachment ....etc (Shorthand property)

***Box-Model\*\****

- The CSS box-model is a box that wraps around every HTML element . It consists of margin , border, padding and the actual content.

- **box-sizing** property sets how the total width and height of an element is calculated

- In CSS by default , the width and height you assign to an element is applied only to the element’s **content-box** .

- By default , the actual width of an element = padding + border + width , and same for actual height .

Hence , to solve this issue we use box-sizing property .

**box-sizing : content-box ->** this is the default value . Any padding or border given to the element gets added over the content-box width , which increases the element’s size .

**box-sizing : border-box ->** By assigning this , we tell the browser to include the padding and border in the element’s(actually content-box’s) total width and height , by shrinking the size of content-box

**float & clear**

**float** : left / right - element floats left/right of the container

**clear** : left/right/both - the element is pushed below the left/right floated elements

**margin**

**margin:auto** - makes a block element in the horizontally-center of container

**Margin** – Provides space between border and outer elements

**Padding** – Provides space between border and the content of the element . With this element’s size increases by-default .

- If container element is non-floating and contents are floating then , the container collapses and the floated contents gets out of the container due overflowing . This issue is resolved by setting **overflow : auto** to the container’s class

**display**

**display:inline** -> the width&height of inline element can’t be changed unlike block element also , top&bottom margin/padding is not respected.

**display : inline-block** -> this allows an inline element to change its height and width

**display : block** – With this display property , an element act as **block** element or paragraph . **It starts on new line and takes the whole width of the container , consisting white-space above and below it**

**display:none** -> It hides the element as well as it erases the space acquired by it

**position**

**- top , bottom , right** and **left** properties are used with either of the **position** perperties

**position : static -** Default value . Element would be placed in normal document flow

**position : fixed -** Element positioned and fixed relative to the viewport

**position : sticky -** It sticks the element at the top of viewport

**position : relative -** element positioned relative to its normal position

**position : absolute** - element positioned relative to the closest non-static container .

**visibility : hidden** -> default value is **visible** , but the value ‘hidden’ hides the element but its space is reserved

**filter : invert(1) ->** It inverts the colors present in the image . We basically use it to change the image color from black to white

**initial and inherit keywords/values**

property(color , height etc) **: initial 🡨 This will set the default value to the property**

property(color , height etc) **: inherit 🡨 This will set the value as the property-value inherited from its parent**

**z-index : value(stack-order)**

- It works only for non-static elements like , absolute , relative etc .

- If the elements are overlapping to each other then **z-index** specifies the **stack-order** of the elements

- An element with greater stack-order is always in front of the lower stack-order element .

**Box-shadow and text-shadow**

**box-shadow : *horizontal(h-offset) vertical(v-offset) blurr-radius spread-radius color inset***

spread-radius - positive value increases the size of the of the shadow and -ve value decreases

inset - changes the shadow from an outer shadow to an inner shadow

- Text-shadow has the same format as box-shadow

**Animations**

- An animation lets an element to gradually change from one style to another

- To set an animation property to any element , we use animation and **@keyframes** statement

- @Keyframes is an **at-rule** which specifies/consists the animation code or new element-style.

- at-rules are CSS statements that instructs CSS how to behave

.box{

background-color: blue;

width: 250px;

height: 250px;

position: relative;

**animation-name: ani1;**

**animation-duration: 2s;**

**animation-iteration-count: 3;**

**animation-delay : 2s ;**

**animation-fill-mode: alternate;**

**animation-timing-function: ease-in-out;**

**animation-direction:reverse;**

}

**@keyframes** **ani1**{

from{

width:200px;

}

to{

width: 500px;

}

}

**Shorthand:-**

**animation : ani-name ani-duration ani-timing-function ani-delay ani-iteration-count ani-fill-mode ani-direction**

**- The animation-fill-mode property specifies a style for the target element when the animation is not playing (before it starts, after it ends, or both).**

**Transitions**

- It allows to change property values smoothly over a given duration .

- :hover pseudo class is used to trigger a transition over any certain element

#box{

display: flex;

background-color: yellow;

height: 100px;

width: 100px;

justify-content: center;

align-items: center;

color: rgb(255, 38, 0);

**transition-property: bg-color / all;** (it can be a single property or all properties)

**transition-duration: 5s;**

**transition-timing-function: ease-out ;**

**transition-delay: 1s;**

}

**#box:hover**{

background-color: blueviolet;

color: azure;

width: 300px;

height: 300px;

font-size: 20px;

border-radius: 5px;

}

**Shorthand:-**

**transition : trans-property trans-duration trans-timing-function ani-delay**

**Transition/Animation-timing-function** values ,

**- It specifies the speed curve of transition/animation**

ease - Specifies an animation/transition with slow start , then fast , then end slowly(default)

linear - same speed from start to end

ease-in - with slow start and fast end

ease-out - with fast start and slow end

ease-in-out - with slow start and slow end

**Differenece b/w Animation and Transition**

|  |  |
| --- | --- |
| Animation | Transition |
|  |  |
|  |  |
|  |  |

**Transform**

- This property applies a 2D or 3D transformation to an element . It allows you to rotate , scale , move , skew etc elements

Format:- **transform : function()**

**trasnsform : translate(**Xpx **,** Ypx**)** -> this method moves an element from its current position to the given coordinate

**transform:rotate**(20deg)

**transform:scale(2,3)** -> this method increases/decreases the size of an element as per the parameters

transform : translateX(x) / translateY(y) / skew(x,y) / skewX(n) / skewY(n) / scaleX(n) / scakeY(n) / marix()

**Variables**

Declaration : **--variable\_name : property ex:-** --primary\_color : red

Usage - **color : var(--variable\_name , fallback value(used if the variable is not found))**

- If variable declared in any element’s css block , it will be a local variable and will be only accessible in that element block only .

ex: h1{ --primary-color : red ; color : **var(--primary-variable , black**)} -> scope here is local for h1

**:root** - It’s a selector that matches the document’s root element i.e HTML element . Hence to make a variable as global variable , we declare those variables inside the **:root** only .

**:root{--primary-color:blue}** -> scope of the variable is global

**Responsiveness**

**Responsive design** - a design which gets adapted to all the devices and sizes of viewport

**Units for responsiveness**

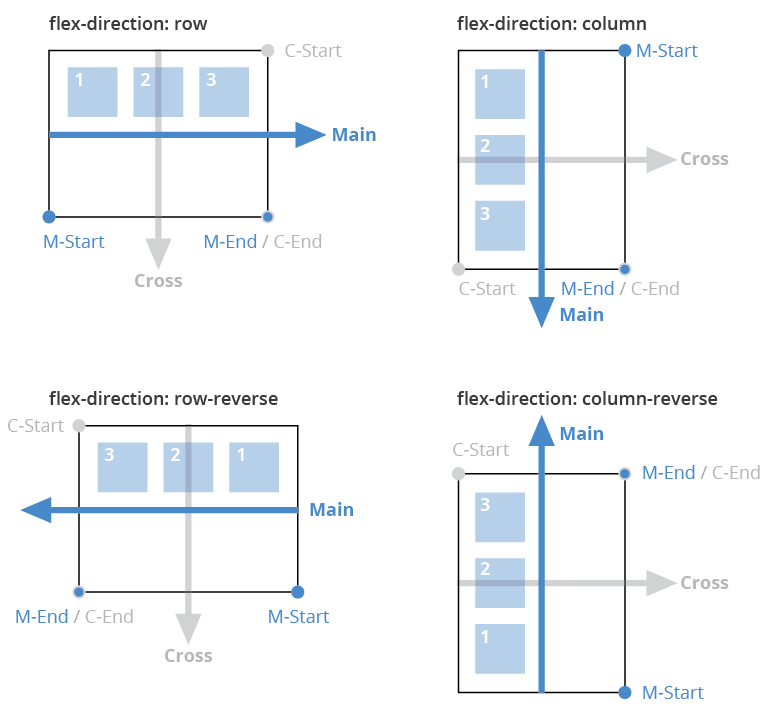
font-size : 10em ; it means 10 \* font size in parent element

font-size : 10rem ; it means 10 \* root size of HTML(16px) ; **1 rem = 16px**

height : 10vh(viewport height) ; <- 10% height of viewport

width : 10vw(viewport width) ; <- 10% width of viewport

**Flexbox**

****

- It is a one-dimentional layout method for arranging items in rows and columns . Flex-items expand(flex) to fill additional space or shrink to fit into smaller spaces

Properties of flex container

**display : flex** -> to initialize the container as a flex-box

**flex-direction : row(default)/column ->** flex-items will be arranged in the row or column

**flex-direction : row-reverse / column-reverse**

**flex-wrap : wrap / no-wrap(default)/wrap-reverse** -> Items will be wrapped as the container shrinks

***flex-flow : flex-direction flex-wrap ; (shorthand)***

**justify-content(item’s space oriented property)** -> this property defines how the browser distributes space b/w and around content items along **main-axis(horizontal)** of the flex-container and the inline- axis of a grid container

**justify-content : start(default) / center / space-between / space-around / space-evenly**

**align-item(item’s oriented property) ->** it specifies the alignment of the items on the **cross-axis(vertical)** inside a flexible container

**align-item : stretch(default) / center / flex-start / flex-end**

Properties of flex item

**order : value(num)** -> This property specifies the order of a flexib;e item relative to the rest flexible items ,i.e larger the order laster the position

**flex-grow:value(num) -** it specifies how much the item will grow as viewport widens relative to the rest of the flexible items inside the same container .For an item Larger the value wider the width relative to rest items ,as the viewport getting wider

**flex-shrink:value(num)** - it specifies how much the item will shrinks as viewport shrinkens relative to the rest of the flexible items inside the same container .For an item , Larger the value , lower the width relative to rest items ,as the viewport getting shrinker

**flex-basis:valuepx -** when flex-direction is row then it controls width , but when it is column then it controls height

***flex : flex-grow flex-shrink flex-basis ; (shorthand)***

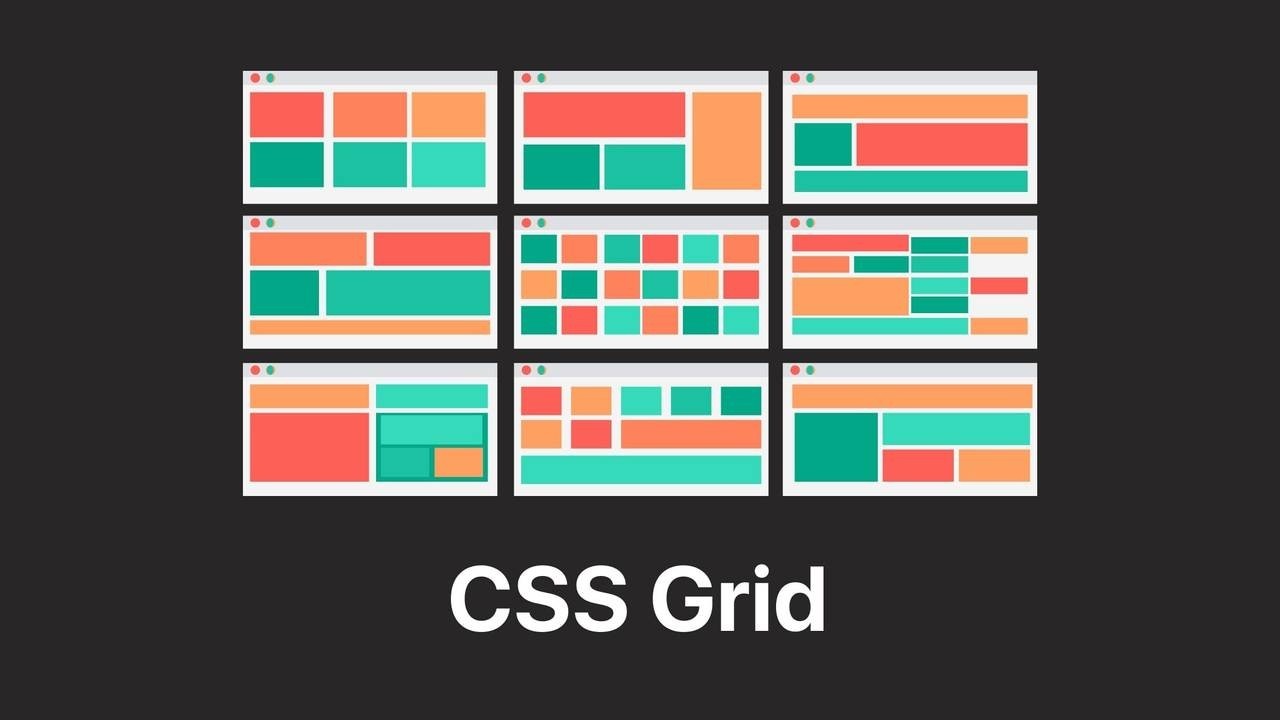
**align-self** - it overrides a grid or flex-item’s align-item value . In grid it aligns the item inside the grid area and in Flexbox it aligns the item on the cross-axis(vertical)

**align-self : auto(default)/flex-end/flex-start/center**

**CSS Grid**

- It is a two dimentional grid-based layout system which lets us to lay contents out in rows and columns unlike flexbox which works only for single-direction either row or column .

**display : grid** -> To initialize the container as a grid



Properties of Grid

**grid-template-columns/rows**-> These specify the size and numbers of columns/rows on a viewport

ex:- grid-template-columns : 100px 100px 100px **OR** repeat(3,100px) , it denotes 3 columns of width 100px

grid-template-columns : **2fr 4fr 1fr , it will divide the whole screen in the ratio og 2:4:1**

**grid-gap** (now its name is simply **gap**) - simply sets gap between rows and columns

**grid-gap(**short-hand for gap**) : grid-row-gap grid-column-gap ;**

**fr** (Fractional Unit) - this unit automatically divides the available space into row/column with provided fractions/ratios

ex:- grid-template-columns : 1fr 2fr 3fr ; it will make only 3(no matter how many items are there) columns of 1:2:3 ratio by taking all width of grid container

But , in the case of rows , as many numbers of rows will be created as the number of items

ex:- grid-template-rows : 1fr 2fr 3fr ; it will only affect first 3 rows and remaining rows will be sized automatically

**grid-auto-rows**-> As it is discussed above that all the rows are not sized by grid-template-rows , hence the remaining rows will be sized by setting the value to this property .

-**Spanning** is the process of merging multiple grid items to create a single item

**grid-column/row-start : 1** <- it specifies that from which line to start in a column/row

**grid-column/row-end :** **3** <- it specifies that to which line to end in a column/row

**grid-column/row : 1 / span 3** <- shorthand property for above span-properties . It will span 3 elements instead 1 to 3 lines

grid-template-columns **:** repeat**(auto-fit , minmax(400px , 1fr))** <- It will make the grid responsive

**auto-fit -** it will automatically decide the number of columns to create as per the viewport-size

**minmax(min , max)** - it’s a CSS function which defines a width-range of a column/row in grid

**grid-template-areas**

- It specifies areas within the grid layout . Grid items are named by **grid-area** property and these names are used in **grid-template-areas** property to define the item’s area or size and position in the grid . OR

- grid items **span(merge**) a particular number of rows and columns which defines their sizes . Like in the following example

<head>

<style>

**.three{grid-area: a;}**

**.one{grid-area: b;}**

**.two{grid-area: c;}**

**.four{grid-area: d;}**

.container

{

display:grid ;

**grid-template-areas: 'c c c c'** <- c or second element spans 4 columns of first row

**'a b b b'**

**'d d d d';**

grid-gap: 5px;

}

.item{

color: white ;

padding:10px;

background-color: red ;

text-align:center;

}

</style>

</head>

<body>

<div class="container">

**<div class="one item">loremispun12</div>**

**<div class="two item">Lorem ipsum </div>**

**<div class="three item">recus </div>**

**<div class="four item">recusan </div>**

</div>

</body>

**Media Queries**

- Media query is a CSS3 technique , which uses the @media rule to include a block of CSS properties only if a certain condition(**browsers’ characteristics like , width , height or scree-resolution , screen-width etc**) is true

**@media** media-type(screen , all ,print)and **(condition)**

**{**

**properties**

**}**

@media **only screen** and (min-width:400px) and (max-width:500px)

**{**

#box-2{

display: block;

background-color: blanchedalmond;

}

**}**

***margin : auto –*** with this property , the element gets **horizontally centered** withinits container

**clear –** this property specifies on which sides of an element , floating elements are not allowed to float

**clear : both -**  no floating elements are allowed on either the left or the right side

**::after** – it is a selector inserts something after the content of each selected elements

* To insert the after/before content , **content** property is used

**display : flex** - we use this to make the flex container flexible

**box-sizing** – This property defines how the width and the height of an element are calculated i.e should they include padding and borders or not .

**box-sizing : border-box** - by this value , the width and height properties include content , padding and border .

**Responsiveness** – It is the technique by which we change the layout of the website according to screensize/viewport and resolution .

* **Media queries**
* **Flexbox**
* **CSS grid**

**display : relative** - element will be positioned relative to the normal position .

**position : absolute** – the element is positioned absolutely to its first positioned parent

**flex-wrap** – This property decides whether flex items are forced onto one line or can wrap onto multiple lines .

**<hr> -**It is used to define a thematic break in an HTML page

- Anchor tags can be used to jump over the specific **Internal section** within the page , by using specific **‘id’** in both target element as well as in the **‘href’** of anchor tag.

**<a href=”#footer”>Jumpt to footer</a>**

**<footer id=”footer”>Footer content</footer>**

- CSS variables look like this : **--main-color : #4587sd ;**

**- :root** is a pseudo-class-selector . This is used to hold **global css variables** so that they can available in any selector

- **line-height** property decides the height of gap b/t two lines of texts

**Version Control(GIT and Github)**

**- It is** also called source control .

- Version control is the practice of tracking and managing changes to software code .

- Version control systems are software tools that helps software teams manage changes to source code over time

**Some Linux Terminal commands**

**mkdir** foldername – creates a directory

**touch** filename – creates a file

**vim** filename - open the file into VIM terminal text editor

**:x** - to save and come out of the VIM

**ls** - to check the directory elements

**rm** filename - deletes the file

**GIT commands**

**git status** - checks the status of the working repo

**git init** - initializes new empty repo

**git add .** - it adds the untracked/unstaged changes to the staging area

**git commit -m “message”** - it saves the changes made with a message . It doesn’t save changes in remote servers but only in the local repo of GIT .

**git log** - it gives the history of your previous commits

**git checkout** first7digits - used to go to(or checkout) specific commits

**git checkout** branchName - to switch between branches

**git checkout -b** newBranch - creates new branch

**git branch** - shows the available branches

- **Branch** in GIT is another line of development , which can be modified without affecting the main part of the repo

**git push origin master** - to save/make changes to the remote server like GITHUB

**Bootstrap**

- It is a CSS framework used to develop responsive and mobile-friendly websites , includes prebuilt classes ready to use .

- CDN(Content Delivery Network) - By CDN one can directly use or include any library into the website without actually downloading it into the website’s folder .

- Bootstrap works on a grid system of 12

- In bootstrap , using things uder container is necessary

- <div class=”**container**”> - this regular/standard container comes with extra margin around it

- <div class=”**container-fluid**”> - this specific container has no margin and takes up the 100% width of the viewport

- all the contents would be in the **columns(<div class=”col”>)** which are the immediate child of parent **row(<div class=”row”>)** class .

**offset-md-2 -** This class add empty space to the left of 2-columns wide .

**SASS(Syntactically Awesome StyleSheets)**

- It is a CSS extension

- It is a CSS pre-processor

- Its syntax is easie , cleaner than CSS

- SASSY CSS(SCSS) is more CSS and less SASS

- FEATURES OF SASS (that are absent in css) : variables , nested rules , mixins , imports , inheritance , built-in functions etc

- We need to set a **sass watcher** through terminal in order to compile and convert sass file into css file every time it changes(because browser only interpret css as **sass is only a preprocessor**)

command : **sass --watch file.sass:file.css**

- **Mixins** allow us to define styles that can be re-used throughout your stylesheet . Mixins can also take arguments and assign them to the properties