CSS BASICS

Introduction:

- CSS provides the styling capabilities.
- Structure alone is not enough; we want the content to be pleasing to the users.
- We can style, color, and do designing in any way we want.

Anatomy of a CSS Rule:

- CSS works by associating rules with HTML elements which govern how they should be displayed.
- The CSS rule contains a selector, after which we have curly brackets and within them is our CSS declaration.
- The declaration contains a property and a value.
- p {color: blue;}
- Stylesheets are files that have multiple CSS rules defined within it.
- Every browser comes with default styles it applies on HTML elements; we overwrite some.

Elements, Classes and ID Selectors:

- CSS selectors are used to determine which set of elements the CSS declarations be applied to.

- **ELEMENT SELECTOR:**
  - We specify only the element name and then declare our CSS.
  - It only styles the elements we target, and doesn't affect other.
  - E.g: **p { color: blue; }**

- **CLASS SELECTOR:**
  - We specify this with a dot and the class name.
  - Targets all that element which have attribute **"class"** with that name.
  - E.g: **.blue { color: blue; }**

- **ID SELECTOR:**
  - We specify this with pound sign and the id name.
  - Targets all the elements with attribute **"id"** with that name.
  - Can only work with 1 element as id is unique for each element.
  - E.g: **#name { color: blue; }**

- CSS allows use to group several selectors into on CSS rule.
- Separate the selectors with comma.
- E.g: **div, .blue, #id1 { color: blue; }**

Combining Selectors:

- This technique allows us to precisely target the elements we want to style.
- There are several ways of doing this:

- **1. ELEMENT + CLASS:**
    - E.g: **p.big { font-size: 20px }, styles every "p" with class "big".**
    - No space between the selectors.
    - This technique is used when we have **"class"** with multiple elements and we don't want to style all of them.

- **2. CHILD SELECTOR:**
    - Syntax: **selector1 > selector2 { … }**
    - It is read from right to left. Selector2 is the direct child and it is targeted.
    - E.g: **article > p { color: blue; }**
    - <article><p>….</p></article> = **Affected.**
    - <article><div><p>…</p></div></article> = **Unaffected.**
    - <p>…</p> = **Unaffected.**

- **3. DESENDENT SELECTOR:**
    - Syntax: **selector1 selector2 {…}**
    - It is read right to left. Every selector2 that is in selector1 at any level is affected.
    - E.g: **article > p { color: blue; }**
    - <article><p>….</p></article> = **Affected.**
    - <article><div><p>…</p></div></article> = **Affected.**
    - <p>…</p> = **Unaffected.**

- These combinations aren't limited to element selector only, we can also have combinations of class, id selectors too! We can have any combination with any type of selector.

Pseudo-Class Selectors:

- The ability to target elements (to style them) based on user interaction with the page.
- For e.g., we want the styling to change when use hovers over something.
- Syntax: **selector:pseudo-class {…}**
- There are many pseudo-class selectors such as **:link, :visited, :hover, :active, :nth-child(num)**
- States of Links:
    - **:link =** the style to use when there is no action done with the link. Initial stable state.
    - **:visited =** the style to use when link is visited at least one (opened at least once).
    - **Above two are usually used together to style the link behavior.**
    - **:hover =** style to use when we hover over the link. (Can also use this with normal tags)
    - **:active =** style the state when user has clicked the link with mouse but hasn't let go of the click.
    - **Above two are usually used together to style this bevavior.**

- **:nth-child(num):**
    - It allows to target a particular element in a list. If we want to target the 3rd element in a list we would use: **li:nth-child(3) {…}**
    - We can also use this to for e.g., style all the odd divs: **div:nth-child(odd**

- We can also combine pseudo selectors in any of the ways we learned above with any type of selector.

CSS RULES AND CONFLICT RESOLUTION

STYLE PLACEMENT:

- Our placement of styles affects which style declarations override which ones.
- We can place our styling in three places:
  - In the <head> tag, we can place styling using <style> tag.
  - Directly with the element using **"style"** attribute.
  - Place all the styles in an external stylesheet.

- We specify external styles using:
  - **<link> tag. We specify two attributes:**
  - **rel = 'stylesheet', which tells the browser that it is a stylesheet.**
  - **href = we provide link of the external CSS.**

- Using external styles, we can have reusability and consistency in all HTML pages.
- Head styles are used to override external CSS.

CONFLICT RESOLUTION:

- There can be a conflict when we define styling in multiple ways, we have to decide which rules would be given priority.

Cascading Algorithm:

- It is an algorithm defining how to combine properties values originating from multiple different sources.
- Cascade combines the importance, origin, specificity and source order of the applicable style declarations to determine which declarations should be applied.

1. ORIGIN PRECEDENCE:
   - When two declarations are in conflict i.e., they specify the same property for the same target. Origin precedence comes with a rule that **"LAST DECLARATION WINS".**
   - When different declarations don't conflict i.e., they specify different property for the same target. The rules is **"DECLARATIONS ARE MERGED".**

2. INHERITANCE:
   - Basic idea is that we have document object model tree. There is <body> having some element within it, and those elements can also have element within in.
   - If we specify styles for a parent tag, all children of that parent will inherit those styles.
   - It does not work in reverse i.e., if we apply to a child, the same styles won't be applied to the parent. Can apply common styles in **<body>** as they are applied to the whole page.

3. SPECIFICITY:
    - The rule is **MOST SPECIFIC SELECTOR COMBINATION WINS.**
    - We can calculate the specificity of a selector by a **score**, the highest would indicate that it is the most specific combination.
    - Arrange types of things that affect score from left to right (highest to lowest):
        - **style = "…"   ID   class, pseudo-class, attribute   #no of elements**
        - Style is the most specific as it is specified with the tag/element.
        - The least is the number of elements used in the selector combination.

- We can override all the rules using **!important** when defining styles.
- For e.g., **p { color: green !important; }**
- Avoid using this though.

STYLING TEXT:

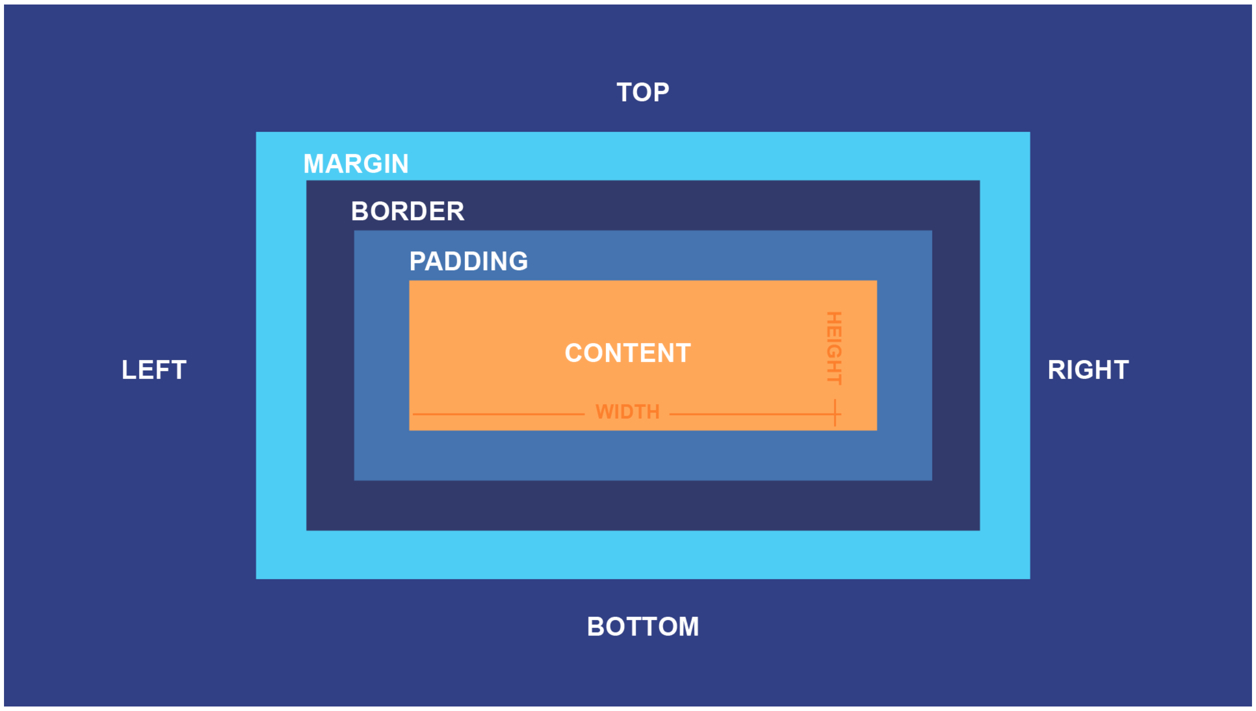
- **font-family:**
    - We specify more than 1 font families like: 'Times New Roman', Times, sarif.
    - In the very end, we select the default font to be selected (either sarif or san-sarif).
    - We specify more than once because it might be that some font in our desktop may not work for some other desktop if it isn't on their computer.
- **color:**
    - Use hex value.
- **font-style:**
    - If we want italic or normal, etc.
- **font-weight:**
    - Weight of boldness of font, can specify keyword **"bold"** or a numeric value.
- **font-size:**
    - We use pixels (px) which is absolute unit of measurement of size.
    - Pixels are relative to the viewing device.
- **text-transform:**
    - Control how text looks: uppercase, lowercase, etc.
- **text-align:**
    - Allows to be center, left, right, justify (within its block element).

- **Relative Font Sizing:**
    - There are 2 units for this: **percent, ems.**
    - **Percent:**
        - font-size = 120% means that we want to take the default size and increase it by 120%
        - Usually used in **<body>** to increase same size for all the HTML elements.
    - **EM:**
        - Unit of measurement equivalent to width of the letter 'm' in the font we use.
        - It is relative to whatever the current font-size is.
        - font-size = 2em, takes the default size and gives the twice size of it.

BOX MODEL AND LAYOUT

BOX MODEL:

- Every HTML element is like a box.
- The box has additional properties which we can change a/c to the layout we want.



- Margin = extends outwards the whole box.
  - margins = top right bottom left
  - Or we can simply give 1 value and it is set for all the sides.

- Padding = extends outwards for the content.
  - padding = top right bottom left
  - Or we can simply give 1 value and it is set for all the sides.

- ### SETTING WIDTH OF THE BOX:
  - if we set **width = 300px,** our box would not be 300px, because the total width for the box would be all the margins, padding, border and width of content added, and it'd not be as we specified.
  - This happens because by default we have **box-sizing: content-box,** which sets the width for the content only and not the whole box!
  - We can change this using **box-sizing: border-box,** now if we specify width, the width is broken for the content in such a way that the total width of the box remains what we specified using **width**.

- ## USING BORDER-BOX FOR ALL ELEMENTS:
  - Normally, we'd use this property in **<body>** and expect it to be applied to all the elements in the page, but this doesn't happen. Because this property can't be inherited.
  - We can use another type of selector **(* star selector)** which applies any styles we want to all the elements in our HTML page.
  - E.g., *** { box-sizing: border-box }**

- ## CUMMULATIVE & COLLAPSING MARGINS:
  - When two elements are placed side by side, the space between them is the sum of margins of their sides.
    - E.g., left margin of box 2 is 50px and right margin of box 1 is 40px, the cumulative margin between them would be 90px.
    - Horizontal margin rule.

  - When two elements are placed in such a way that one is on top of the other, then the larger margin is selected between them.
    - E.g., bottom margin of element 1 is 20px, and top margin of element 2 is 30px, then the margin between both elements would be selected as 30px.
    - Vertical margin rule.

- ## DEALING WITH DEFAULT STYLING:
  - By default, there are some margins, padding, etc., applied to elements by the browser.
  - To deal with them or reset them, we can use the **star selector** and set those as 0.
  - E.g., Setting margin 0 for all elements: *** { margin: 0 }**

- ## SETTING HEIGHT OF THE ELEMENT:
  - When we change the width, the content is shifted and the height is also adjusted.
  - When more content is added which is greater than the size of the box, it is spilled outside the box if we had specified a **height** which was insufficient for the whole content.
  - And if there is another element below it, it is spilled on that element.
  - To deal with this issue, we can use **overflow** property:
    - By default, it is set to **overflow: visible.** (Causes to spill)
    - We can clip it using **overflow: hidden.** (Clips the content where box ends).
    - Can put scroll bars using **overflow: auto.** (Scroll bars are there if needed).
    - We can put scroll bars whether they are needed or not using **overflow: scroll.**

BACKGROUND PROPERTY:

- There are a number of properties we can use to do stuff with background.
- **background-color =** sets color of the background of the element.
- **background-image: url('path') =** we can set an image as our background by giving a link in url().
- **background-repeat =** we can have repeat, repeat-x, repeat-y, no-repeat for the image we used.
- **background-position: horizontal value vertical value.**
    - We give two values (top right) to place the image.
    - If we specify only one, the other is defaulted to **center.**

- We can use background color and image at the same time too.
- We can just use **background** attribute and define all the styles in it too.
    - **background: url('…') no-repeat right center blue**
- This property is useful for adjusting image size/resolution based on screen size.

POSITIONING ELEMENTS BY FLOATING:

- We can float elements using **float** property and possible values are: **right, left, top, bottom, both.**
- When we float an element, the browser takes that element out of the regular document flow.
- Also, for floated elements, the elements don't ever collapse, they don't follow that rule.
- If we have floated elements in our "div" or inside any other element, they won't behave like they are supposed to, those elements would then collapse if we float an element to some direction.
    - To fix this, we must tell the browser to resume regular flow of the document.
    - We do that by using: **clear: left, right, both;**
    - Must be used with the element we want to be regular.
    - This property is not restricted to non-floated elements only, we can also use for floated elements, and then float them again.
    - **clear: left;** means that the element doesn't allow any element to be floated to the left of it.
    - **clear: both;** we use this when an element has floated elements to both its left and right. We use this to ensure that the element would gets it own line and follow regular flow.
    - When there is insufficient space for floating elements to occupy the same line as normal elements, they are pushed to the next line.

- Can use this to create columns.

RELATIVE AND ABSOLUTE POSITIONING:

1. Alternative Element Positioning:

- Floating elements are an example of this type of positioning.
- **"ALTERNATIVE POSITIONING" to the normal flow of document.**

2. Static Positioning:

- Another way of saying **"NORMAL DOCUMENT FLOW".**
- A default setting used for all elements.

3. Relative Positioning:

- Element is positioned relative to its position in normal document flow.
- If offsets are applied to move the element, there will be offsets from the original document flow position of that element.

- CSS (**offset)** properties are: **top, bottom, left, right.**
  - These are like edges of the element box.
  - Set relative position using: **position: relative**

- Elements are not taken out of normal document flow. Meaning, the element we move is sitting in its original position even thought its visually off somewhere else.
- The element's original spot is preserved.

4. Absolute Positioning:

- All offsets (**top, bottom, left, right**) are relative to the position of the nearest ancestor which has positioning set on it other than **static.**
- Some parent, grandparent… must have its position set to something other than **static** in order for absolute positioning to work.
- By default, only **HTML element/tag** has its position as **relative.**
- Element is taken out of normal document flow.

- Specify position as: **position: absolute.**
  - When we apply this, the element is taken out of normal flow.
  - It remains in the place where it was without any offsets. (Until we change)

- If the outer element has offsets applied on it (changing its position), we don't need to worry about all the **absolute** elements within that container element. As we offset that container, all other elements inside it are offset automatically.

INTRO TO RESPONSIVE DESIGN

MEDIA QUERIES:

- They allow us to group styles together and target them to different devices based on features like width, height, orientation.
- Difference between viewing a website on a desktop and cell phone is the screen size, CSS allows us to have different layouts for different screen sizes.
- The most common way of adjusting styling for different sizes is to provide/define different styles for different screen sizes.

Syntax:

- **@media (max-width: 767px or any media-feature) { styles… }**
- The media feature we specify resolves in either TRUE or FALSE.
- We have features like **width, height, orientation, screen…** but the most common ones are **max-width** and **min-width.**
- Most common way of targeting devices is by width.

Logical Operators:

- Can group together media features using **AND OPERATION (and), OR OPERATION (comma ,).**
- E.g., **@media (min-width: 768px) and (max-width: 991px) { … }**
- Most common operator we use is **and.**

Common Approach:

- First, we define base styles which are applied to all the screen sizes no matter what.
- We can then define particular rules or delete/remove previous ones by defining media queries to target different screen sizes.
- While defining ranges, make sure to not overlap ranges for 2 media queries.
    - E.g., **query1 = max-width: 1200px,** **query2 = min-width: 1200px**
    - This is wrong and for **query2** we must write **1199px.**

RESPONSIVE DESIGN:

- Sites designed to adopt its layout to the viewing environment by using fluid, proportion-based grids (specify width using percent), flexible images and CSS3 media queries.
- Site's layout should adopt to the size of the device.
- Content verbosity or its visual delivery may change (e.g., phone number displayed prominently in mobiles), some parts for the website may also be hidden for mobile.

12-Column Grid Responsive Layout:

- Almost every responsive framework uses this. Common layout for responsive sites.
- Reason of 12? Because we have factors of 12 i.e., 1, 2, 3, 4, 6, 12. Meaning we can sub-divide our page into sections that are evenly split and nicely laid out.
- When we approach these layouts, we know that browser is 100% (width), which lets us calculate what one column in this grid is i.e., 100% / 12 == 8.33%.
- We can divide our columns width using this percentage, and can have different ones for different sized devices.
- We can also have grid within a column itself; can also consider 1 column as a grid and have sub-sections within in (tag within tag).
- We can have nested grids.

Trick to make a paragraph (or any tag) centered inside div is to specify width of that tag and then set margin left and right as auto.

Approach:

- We define media-queries a/c to the standard sizes of devices.
- We specify 12 classes with each query (col-lg-1 for large size, col-md-1 for medium size)
- Now, we define width of all of these classes according to the number of columns we want (see point 3 above) **FORMULA: (100%/12) * (columns we want)**
- Then, we can use these classes, both of them (col-lg-1, col-md-1) at the same time with any tag and our media-query would ensure that at one time, only 1 class would be TRUE/ACTIVE.

One Mobile Feature to Consider:

- There is one feature on all phones that they try to zoom out on websites that don't tell them to do anything different.
- Meaning, they try to fit the entire content into the viewport of the phone and ignore the concept of responsive website.
- We have to explicitly tell the devices that the website is responsive so that they don't zoom out.
- We do this by using another meta tag:
    - **<meta name = 'viewport' content = 'width-device-width, initial-scale = 1'>**
    - With this the browser will now consider the device width as the width of the viewport and it won't scale anything up or down (scale = 1).

TWITTER BOOTSTRAP

INTRODUCTION:

- It is the most popular HTML, CSS, JS framework for developing responsive, mobile first projects on the web.
- It mostly contains CSS, pre-defines a lot of CSS classes.
- CSS needs to target a certain HTML structure in a complex HTML document with more than 1 element. Because of this, bootstrap often times requires us to have particular HTML structure in HTML document.
  - It asks to add an additional "div" here and there and give it a particular class so that the CSS bootstrap defined would be applied to it.

- It has a cool JS framework based on **JQuery API's** and plug-in architecture and that lets us put in different plug-ins that help in enhancing the website.

- This framework is also called **MOBILE FIRST FRAMEWORK,** meaning that we deal with how user interface is laid down on mobiles first. There are two approaches to this meaning:
  - **PUREST APPROACH:** which says to code your mobile version first, it helps in content strategy; how to lay it out, what it should be, etc.
  - **NOT 100% PUREST:** idea is that we plan for mobile in our styles but can start coding with desktop version first if.
  - **TRUTH IS:** we can code any version first, both approaches are valid.

- Most people agree to code **MOBILE VERSION FIRST**, start from there and then go to desktop.

Disadvantages of Bootstrap:

- It is too big or too bloated.
- There are a lot of features we might not even use.
- We can make our own framework which is specific and small. However, creating our own framework is challenging too.
- We may or may not be able to write it as efficiently as Twitter Bootstrap.

SETTING IT UP IN HTML FILE:

- We first have some additional folders (css, fonts, js) when we download Bootstrap.
- **getbootstrap.com,** can download from here.
- In css folder, we will have: **bootstrap.css, bootstrap.min.css** (removed extra spaces in this).
- In fonts folder, we will have files needed for basic **Bootstrap Functionality.**
- In js folder, **bootstrap.js, bootstrap.min.js** (minified version, spaces removed, etc.…)
- For JS to work, we also have to download a version of **JQuery.**
    - **jquery.com,** download **jquery.2x** and save it in JS folder.

- **HTML DOCUMENT:**
    1. Meta tag for charset.
    2. Bootstrap requirement of meta tag, which tells IE to go use its latest version:
       <meta http-equiv = 'X-UA-Compatible' content = 'IE=edge'>

    3. Meta tag for **viewport.**
    4. Title tag.
    5. First link tag: connects to **bootstrap.min.css** (CSS framework of bootstrap)
    6. Second link tag**:** connects to **css/style.css** (our own defined style). This link must be after the bootstrap link, because we will override some of bootstrap styles.
    7. At the very end of **<body>** tag, right before we close it, we declare some JS files, first is the JQuery file, second is the bootstrap JS file, third is our defined JS:
        a. **<script src = "js/jquery file"></script>**
        b. **<script src = "js/bootstrap.min.js"></script>**
        c. **<script src = "js/script.js"></script>**

BOOTSTRAP GRID SYSTEM:

- One of the central components of bootstrap that easily allows us to create responsive layouts in the grid system.

```
<div class = 'container>
        <div class = 'row'>
                <div class = 'col-md-4'> COL 1 </div>
        </div>
</div>
```

Nothing special about using div, we can use any element/tag in place of them and define grid using this structure.

- Bootstrap grid always has to be inside of **container** wrapper. Two choices for this:

    1. **class = 'container'**
        - has fixed width that is still responsive based on browser width. Meaning, it has one width for one size, different width for another size.

    2. **class = 'container-fluid'**
        - this class stretches the layout to be of full width of the browser and provide consistent padding around the grid and other content.

- **class = 'row'**
    - Creates horizontal groups of columns. Meaning, that the columns collapse and interact with each other as a group but independently from columns in another row.
    - This class also create a negative margin to counter-act the padding the **container class** sets up.
    - This is done because each column has its own padding because we want to visually separate columns. If there was no negative margin applied to the row, the padding of the **container** would be in addition to the padding of the edge column. And the content of the column would not align nicely with the rest of the column outside the grid.

- Having consistently aligned edges of different content structures within the page is a basic design principle which BOOTSTRAP implements for us.

- **Column class definition**
    - Every bootstrap column class is defined using the template **COL-SIZE-SPAM**
    - **SIZE:** width range, identifier. MD = medium, LG = large (greater than 1200px)
    - **SPAN:** column values ranging from 1 to 12.
        - If we specify columns in our row that exceed 12, then they will do what other floats too i.e., wrap/jump to the next row.
        - If we specify bunch of columns and the element that pushes the total sum of spans above 12, it'd wrap to the next line.

- To get sizing used in bootstrap:
    - Go to getbootstrap.com
    - Navigate to Grid System
    - Scroll down and see the different sizes and their names

- The size COL-XS will never collapse, it always stays horizontal no matter what the screen size is.
    - Meaning, if we want to guarantee that there is always a particular layout to be followed, we can specify it in COL-XS and it will stay no matter what the size of the screen is.