8020 LESSONS
BRINGS YOU

80/20 CSS

Learn 20% CSS which will give you 80% of the result.

PROJECT BASED

BUILD 3 REAL-WORLD RESPONSIVE WEB PAGES.

BY SWASTIK YADAV

TABLE OF **CONTENTS**

8020 PRINCIPLE

8020 principle states that 80% of the results come from 20% of input. This means to achieve an 80% result you need to learn only 20% CSS.

In the real world developers mostly do the same kind of work (uses only 20% of the CSS) on a day-to-day basis.

And this book is all about that. This book will teach you the 20% CSS that you need to know in order to get things done.

BEFORE WE BEGIN

Hello, I am Swastik,

A full-stack software engineer. Thanks for purchasing the book, I really appreciate your investment.

This book is the result of my handwritten notes, diagrams, and a lot of sleepless nights. This is the CSS book I wish existed when I was learning CSS.

After reading the entire book, if you feel that you didn't find any value in it, just send me an email (within 15 days of purchase) at **8020lessons@gmail.com**. And I will refund your money. No questions asked.

Now let's begin!

INTRODUCTION AND ENVIRONMENT SETUP

INTRO TO HTML / CSS. THE ROLE OF HTML AND CSS IN BUILDING REAL-WORLD WEBPAGES.

ENVIRONMENT SETUP FOR WEB DEVELOPMENT

As web developers, we need to set up our environment before we can start building web pages. We need a code editor to write code, and a web browser to check our work.

CODE EDITOR

Code editors are used for writing and structuring our code. Here are a few popular code editors.

- Brackets
- Atom
- Sublime text
- Visual Studio Code (Recommended)

I recommend you to use VS Code as it is used by the majority of professional developers. For your system-specific installation guide visit the official website of these code editors.

WEB BROWSER

A web browser is simply an application to access the world wide web (The Internet). You might already be using a browser to read this lesson.

The two most widely used web browsers by developers are:

- Google Chrome
- Mozilla Firefox

These two are the most modern browsers with a lot of features for web developers.

KNOW YOUR HTML - COMMON TERMS

Before anything else, it is important to understand the syntax and common terms used in HTML. In this lesson, you will learn what is HTML and what are the 3 building blocks of HTML.

WHAT IS HTML?

HTML stands for "HyperText Markup Language". It is the structure of a web page, it gives meaning to the content of a webpage. All the texts, forms, and multi-media on a webpage are defined by HTML with tags like heading, input, image, video, etc.

HTML COMMON TERMS

There are 3 terminologies that are the building blocks of HTML.

8020lessons.in

1 ELEMENT

The element defines the type of content. Whether it is a paragraph, a heading, an image, or a form. For example, we define paragraphs with the **p** element, headings with multiple levels of **h** element, links/anchor with the **a** element, and images with **img** element.

2 TAGS

A tag is denoted by a less-than (<) and a greater-than (>) sign wrapped around an Element. Most elements have an opening tag (<>) and a closing tag (</>). The opening tag denotes the start of the Element and the Closing tag denotes the end of the Element

Some elements like image and input don't need a closing tag. These are referred to as self-closing tags.

3 ATTRIBUTES

An attribute provides additional value to the element. The most commonly used attributes are **class, id, src, and href**. For example in an image tag source of the image is defined with the **src attribute**. And in an anchor element link to a different page is defined with a **href attribute**.

HTML DOCUMENT TYPE STRUCTURE

HTML documents are files with .html file extension. Here is an example of HTML file structure.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
  </head>
  <body>
    <p>Here goes all the visible content on the webpage.</p>
  </body>
</html>
```

- **<!DOCTYPE html>** informs the web browser that we are using the latest version of HTML.
- **<html>** element is the beginning and </html> is the end of the structure.
- **<head>** element contains information about the page. Like title and metadata. Content of the <head> element is not visible on the webpage.
- **<meta charset="utf-8">** specifies the character encoding of the page.
- Only content between the opening (**<body>**) and closing (**</body>**) body tag is visible on the webpage.

KNOW YOUR CSS - COMMON TERMS

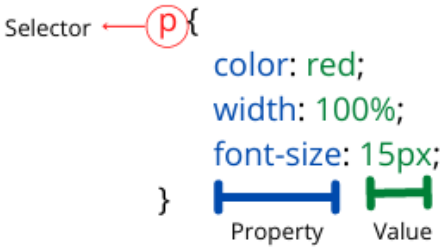
Let's get familiar with common terms used in CSS. In this lesson, you will learn what is CSS and what are the most commonly used terms in CSS.

WHAT IS CSS?

CSS stands for "Cascading Style Sheet". It handles the presentation and appearance of HTML elements on a webpage using fonts, colors, and positioning. Anything related to how an element should look and feel must be handled by CSS.

CSS COMMON TERMS

There are 3 terminologies that are the building blocks of CSS. When you need to apply styles on an HTML element think of it as a 3 step process.



8020lessons.in

1. Select the element you want to style.
2. Specify which property of that element you want to change.
3. Specify the value of that property.

SELECTORS

Selectors specify which HTML element or elements need to be styled. You may want to increase the font size of all paragraph elements or you may just want to change the color of a specific paragraph element. Selectors help you select the element you want to target.

Most common types of selectors.

1 **Type Selector:** With type selectors, you can directly target the elements with their names.

```
p {
  color: red;
  width: 80%;
  font-size: 16px;
}
```

2 **Class Selector:** With class selectors, you can target elements by their class attributes. Use class selectors when you need to apply the same stylings to multiple elements. In the CSS file, we denote the class with a (.) followed by the class attribute name.

```
<head>
    <style>
        .para {
          color: red;
          width: 80%;
          font-size: 16px;
        }
    </style>
</head>
<body>
    <p class="para">This is a paragraph element 1</p>
    <p class="para">This is a paragraph element 2</p>
</body>
```

3 **ID Selector:** With ID selectors you can target an element by its ID attribute. There can not be multiple ID attributes with the same name. Use ID selector to apply unique styling to an element. In the CSS file, the ID attribute is denoted by (#) followed by the ID name.

```
<head>
    <style>
        #red-background {
          background: red;
        }
    </style>
</head>
<body>
    <p id="red-background">This is a paragraph element</p>
</body>
```

PROPERTIES

This determines what property of the selected element needs to be changed. You may want to change the color, font size, background, height, or weight of the paragraph element.

Color, font size, background, weight, and height are just a few properties to name.

VALUES

This defines the value you want to set for the property of an element. For the color property, you can set the value to red, green, blue, or any other color.

Say you want to change the color of a paragraph element to red. Here paragraph element is the selector, color is the property, and red is the value of that property.

CONNECTING HTML AND CSS

How do you get your HTML and CSS to talk to each other? By referencing CSS in your HTML.

HTML and CSS are separate technologies and they should remain that way. You should not write CSS in HTML and vice versa. The best practice is to reference the CSS file in the head of the HTML file with a link element.

```
<html>
    <head>
        <link rel="stylesheet" href="style.css">
    </head>
    <body> ... </body>
</html>
```

CROSS-BROWSER COMPATIBILITY WITH CSS RESETS.

Every web browser has its default settings for different elements. So, to ensure cross-browser compatibility (ensuring that your website looks the same on all browsers) CSS resets are widely used. It resets all the default browser settings to zero so that you can define your default styling for different elements.

CSS reset code needs to be at the top of the CSS file due to cascading reasons. We will learn more about cascading.

Eric Meyer's reset is one of the most popular CSS resets. Copy the provided CSS code from Eric Meyer's reset and paste (along with the credit to Eric Meyer) it at the top of your CSS file.

SEMANTIC HTML FUNDAMENTALS

SEMANTIC HTML IS SO IMPORTANT THAT WE WILL COVER IT RIGHT FROM THE BEGINNING INSTEAD OF INTRODUCING IT SOMEWHERE AT THE END.

SEMANTIC HTML ELEMENTS FUNDAMENTAL

Semantic elements are a very important part of HTML5, which defines the purpose of an element and it makes your code readable for both humans and machines. Prior to HTML5 developers used class attribute on div and span elements to specify the purpose of an element.

For example, here is how you would add a footer to a page prior to HTML5.

```
<body>
  <div class="footer">
    © visionmad.com 2021
  </div>
</body>
```

And here is how you should do it now with semantic HTML5.

```
<body>
  <footer>
    © visionmad.com 2021
  </footer>
</body>
```

WHY SEMANTIC ELEMENTS?

There are several benefits of using tags that have semantic meaning. I have listed two major benefits below.

1. Easier to read and understand: It adds value to an element by clearly defining what each element is about.
2. Accessibility: It enables computers, screen readers, search engines, and other devices to read and understand the content on a web page.

TEXT BASED SEMANTIC ELEMENTS.

Text-based semantic elements provide meaning and purpose to texts on your web page.

Headings

Headings are used to defining multi-level titles of a topic. Headings are of 6 different levels from **<h1> to <h6>**. <h1> being the most important title and <h6> being the least important title.

```
<body>
  <h1>Heading level 1</h1>
  <h2>Heading level 2</h2>
  <h3>Heading level 3</h3>
  <h4>Heading level 4</h4>
  <h5>Heading level 5</h5>
  <h6>Heading level 6</h6>
</body>
```

Here is how it looks on the web page.

Paragraphs

As the name suggests, this is to add some paragraphs that mostly support your headings. The last sentence that you read was a paragraph element.

```
<body>
  <p>
    As the name suggests, this is to add some paragraphs that mostly supports
    your headings.
  </p>
</body>
```

Strong

A strong element is used to make the text **bold**. Here is an example.

```
<body>
  <p>
    Strong element is used to make a text <strong>bold</strong>.
  </p>
</body>
```

Emphasis

An emphasis element is used to make the text *italic*. Here is an example.

```
<body>
  <p>
    Emphasis element is used to make a text <em>italic</em>.
  </p>
</body>
```

STRUCTURE BASED SEMANTIC ELEMENTS.

These semantics denotes the structural intension of an element and adds structural value and meaning to it. Some of the most used and common structure based semantic elements are **<header>, <nav>, <article>, <section>, <aside>, and <footer>**.

8020lessons.in

The point to be noted here is that these elements behave exactly like a ** **. They do add a structural purpose to an element but you still have to rely on CSS to change their position and styling. For example, the **<aside>** tag won't put the element on side of the page by default, that needs to be done by CSS.

Header

The **<header>** element is used at the top of the page. It may contain a logo or a navigation element.

```
<header> ... </header>
```

Navigation

The **<nav>** element is used for navigation purposes. It contains a collection of links to other pages of the same website.

```
<nav> ... </nav>
```

Article

The **<article>** element is used as an independent, self-contained content on the page which can be reused on other parts of the page as well.

You will see **<article>** and **<section>** together in next example.

Section

The **<section>** element is used to group related content together. A section is a thematic grouping of content.

```
<section>
  <p>Top Professional Profiles</p>
  <section>
    <p>Entrepreneurship</p>
    <article>Elon Musk</article>
    <article>Mukesh Ambani</article>
    <article>Ratan Tata</article>
  <section>
  <section>
    <p>SuperHeros</p>
    <article>SuperMan</article>
    <article>Wonder Woman</article>
    <article>BatMan</article>
  <section>
</section>
```

Aside

The <aside> element is used to put content off to the left or right side of the page. For example a sidebar.

Again, it won't put the content on right or left of the page by default. It is just to define the purpose of the content. Positioning needs to be done with CSS.

```
<aside> ... </aside>
```

Footer

The **<footer>** element is used at the bottom of the page to provide some additional links and information about the page or overall website.

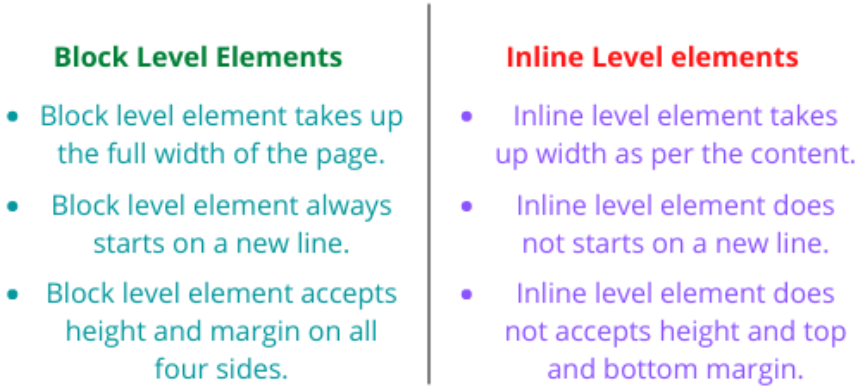
```
<footer></footer>
```

BLOCK VS INLINE LEVEL ELEMENT.

This lesson will introduce you to the Block and Inline level element. Let's understand it with two non-semantic general elements in HTML - ** ** and ****. But first, let's define Block and Inline elements in HTML.

WHAT IS BLOCK AND INLINE LEVEL ELEMENT?

Web browsers treat every element as a box. And there are two types of boxes - Block and Inline. This is called the box model which you will study in further lessons of this book.



Top and bottom margins do not affect inline elements because inline elements flow with content on the page. You can set left and right margins on an inline element but not top or bottom because it would disrupt the flow of content.

The 3rd point in the above image is not discussed in this lesson. We will discuss it in the box model lesson.

IMPORTANT BLOCK LEVEL ELEMENTS

-
- <header>
- <nav>
- <main>
- <p>
- <h1>-<h6>
- <article>
- <section>
- <aside>
- <form>
- <footer>

IMPORTANT INLINE LEVEL ELEMENTS

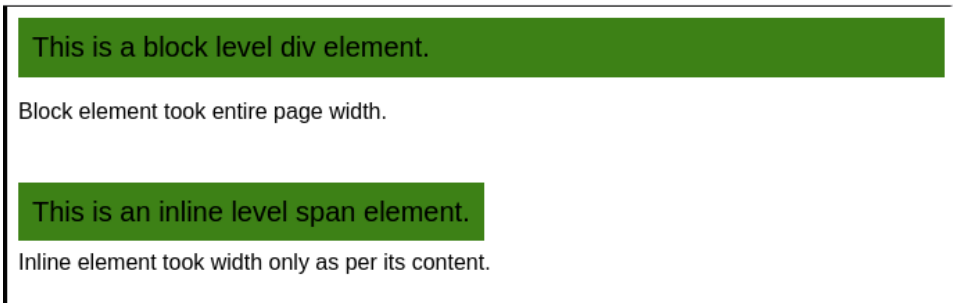
-
- <a>
- <button>
- <input>
-
-
- <img>

DIVS AND SPANS

Div and Span are two non-semantic general elements in HTML, used for grouping and styling purposes. **Div is a block-level element** and **Span is an inline-level element**.

```
<head>
  <style>
    .block-div {
      background: green;
    }
    .inline-span {
      background: green;
    }
  </style>
</head>

<body>
  <div class="block-div">
    This is a block level div element.
  </div>
  <span class="inline-span">
    This is an inline level span element.
  </span>
</body>
```

Here is the result of the above code snippet. Notice how div took entire page width whereas span took width as per its content.



INLINE ELEMENT CAN NOT WRAP BLOCK ELEMENT

Block elements can wrap an Inline element, but an Inline element cannot wrap a Block element.

This is right.

```
<p>
  Block element like p can wrap
  <span>
    inline element like span.
  </span>
</p>
```

This is wrong

```
<span>
  Inline element like span can not wrap
  <p>
    block element like p.
  </p>
</span>
```

There is only one exception to this rule. Anchor **(<a>)** which is an inline element is allowed to wrap block elements. Because you may want to add a hyperlink to a block element.

WORKING WITH HYPERLINKS

The hyperlink is one of the core concepts of the web. It is used to create links from one web page to another web page. The linked web page can be a page from the same website or it can be a page from a different website.

We specify the URL of the linked page in **href attribute**. The href stands for hyperlink reference.

```
<a href="https://twitter.com/8020lessons">8020 Lessons Twitter Account</a>
```

RELATIVE AND ABSOLUTE PATHS

Link to a page on the same website uses a relative path and a link to a page on a different website uses an absolute path.

The relative path does not include the domain name because you are linking to a page under the current domain. It just includes the directory and file names to be changed. See the example below.

```
<a href="/curriculum/html-css">HTML CSS Curriculum page</a>
```

The absolute path includes the domain name along with the directories and file name because now you are linking to a website or domain outside the current one.

```
<a href="https://en.wikipedia.org/wiki/HTML">Wikipedia HTML page</a>
```

CREATING AN EMAIL LINK

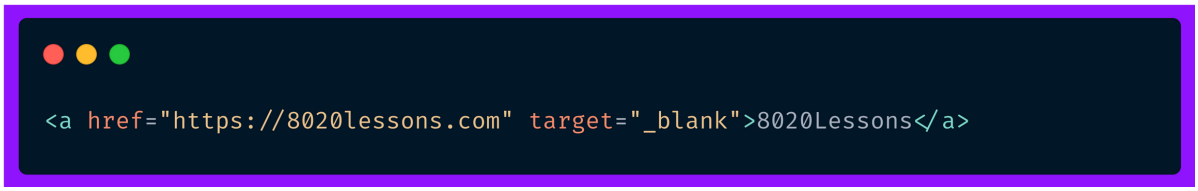
An email link opens the default mailing app of the system so that your user can easily send you emails. To create an email link href attribute value needs to start with mailto: followed by the email address to which email should be sent.

A simple example.

```
<a href="mailto:swastik@8020lessons.com">Send feedback about this book.</a>
```

OPEN LINK IN A NEW TAB

Links can be opened in a new browser window by using the **target attribute**. This is also good when you don't want your user to leave your site.



CASCADING STYLE SHEET FUNDAMENTALS

CSS STANDS FOR CASCADING STYLE SHEETS. WITHIN CSS, ALL STYLES CASCADE FROM THE TOP OF A STYLE SHEET TO THE BOTTOM, ALLOWING DIFFERENT STYLES TO BE ADDED OR OVERWRITTEN AS THE STYLE SHEET PROGRESSES.

CALCULATING CSS CASCADE SPECIFICITY

This lesson will teach you exactly how CSS decides to render styles by using the cascade and specificity.

THE CASCADE

As CSS stands for **cascading style sheet**. Cascade is something arranged or occurring in a series or a succession of stages. In CSS all styles are applied from the top of the file to the bottom of the file.

For example, if you set the background of a div twice the color from later will be applied.

```
div {
  background: green;
}
div {
  background: blue;
}
```

The blue color is applied in the background of the div because it appears later in the cascade file.

CALCULATING SELECTORS SPECIFICITY

The specificity of CSS selectors is defined by their point value. Type selector has the lowest specificity with point value **0-0-1**, Class selector has medium specificity with point value **0-1-0**, and Id selector has the highest specificity with point value **1-0-0**.

When a styling conflict occurs, the selector with higher specificity wins regardless of its place in the cascade file.

```
<head>
    <style>
        #div-id {
          background: red;
        }
        .div-class {
          background: green;
        }
        div {
          background: blue;
        }
    </style>
</head>
<body>
    <div class="div-class" id="div-id">I am a div</div>
</body>
```

The resulting div background is red.

As per cascade background should be blue, but here specificity of ID took precedence regardless of its position in the cascade file because it has the highest specificity with point value 1-0-0.

COMBINED SELECTORS SPECIFICITY

You can combine different selectors to achieve specific styling needs. First, let's discuss what is combined selectors.

Combination of selectors.

Say for example, you have a **<section> element** which contains multiple **<p> elements**. One of the **<p> tag** have a class name of **profile_text**. With combined selectors you can target **all <p> elements of that <section>** or **just the <p> element with the class name of profile_text**.

```
<head>
    <style>
        .profile_card p {
          color: gray;
        }
        .profile_card p.profile_text {
          color: aqua;
        }
    </style>
</head>
<body>
    <section class="profile_card">
      <p> ... </p>
      <p> ... </p>
      <p class="profile_text">Web Developer</p>
    </section>
</body>
```

In the above CSS snippet, there are two examples of combined selectors.

Calculating combined specificity

First one with a **class (profilecard)** and a **type (p) selector**. This will style all **<p> element** under profilecard **<section>**

Example 1

```
.profile_card  p {
    color: gray;
}
```

.profile_card (class selector)	0	1	0
p (type selector)	0	0	1
Resulting point value	0	1	1

8020lessons.in

The resulting point value is **0-1-1**.

Second one with **two class selectors (profilecard, profiletext)** and **one type selector (p)**. This will style only **<p> element** with **profile_text** class name under <section> element.

Example 2

```
.profile_card  p.profile_text {
    color: gray;
}
```

.profile_card (class selector)	0	1	0
p (type selector)	0	0	1
.profile_text (class selector)	0	1	0
Resulting point value	0	2	1

8020lessons.in

The resulting point value is **0-2-1**.

That's how you combine selectors and calculate the specificity of combined selectors.

CSS COLOR PROPERTY AND LENGTH UNIT

In this lesson, you will learn about different ways of using colors and different length units in CSS.

COLORS IN CSS

Colors are an extremely important part of web design and development. They represent subliminal messages of the brand and website. You have 4 ways of using colors in your CSS.

- **Keyword (Direct color name)**: Keyword color values are the direct name of the color you want to use. For example red, green, and blue.
- **Hexadecimal**: Hexadecimal consists of a # followed by six characters. These characters are numbers from **0 to 9** and letters from **a to f**. The first two characters represent the **shade of red**, the next two represent the **shade of green**, and the last two represents the **shade of blue**. You can mix various shades of red, green, and blue to create new colors, to be specific this permutation combination creates over 16 million color options.
- **RGB**: RGB works similar to hexadecimal with various shades of red, green, and blue. RGB takes 3 number characters from **0 to 255**. 0 is the lightest shade and 255 is the darkest shade. Here is how Red, Green, and Blue are defined in RGB.

  Red rgb(255, 0, 0)

  Green rgb(0, 255, 0)

  Blue rgb(0, 0, 255)

- HSL: **HSL stands for Hue, Saturation, and Lightness**. It takes three values just like RGB. The first value is a plain number from **0 to 360** which represents the degree of color on the color wheel. The second value is saturation in percentage from **0% to 100%**. It defines how saturated the color is. The third value is lightness also in percentage from **0% to 100%**. It defines how light or dark the color is. Here is an example of Yellow color with HSL.

  Yellow hsl(60, 100%, 50%)

Alright! Enough talk about colors. Going forward you will learn more about colors on your own. Now let's talk about length units in CSS.

LENGTHS UNIT

Length units in CSS are commonly used for width, height, and font-size. Length unit comes in two different forms absolute and relative length.

Absolute Length unit

Pixel is the most popular absolute unit of measurement and is represented by px unit notation. To put things in perspective, you should know that a pixel is 1 / 96th of an inch. Pixels are mostly used to set the font size of texts and the width of elements. Here is an example setting font-size of p element to 20px and width and height of div to 50px;

```css
p {
    font-size: 20px;
}
div {
    width: 50px;
    height: 50px;
}
```

RELATIVE LENGTH UNITS

Relative length units are a bit more complex as they rely on device sizes and other units.

Relative Length unit

1 Percentage: on an element works relative to the parent of that element. If you want the width of a div to be half of its parent set it to 50%.

```css
div {
  width: 50%;
}
```

2 Em: Em is relative to the element's font size, **1em is equal to the font size of the element**. For example if font-size is 20px and width is set to 5em. The resulting width is (20 * 5) 100px. And when font-size is not specified for the element, em becomes relative to the closest parent with font-size specified.

```css
.hero-section {
  font-size: 20px;
  width: 5em;
}
```

3 Rem: Rem is relative root font-size, which is defined in **<html>** type selector.

```css
html {
  font-size: 16px;
}

h1 {
  font-size: 2rem;
}
```

THE BOX MODEL

THE BOX MODEL IS ONE OF THE MOST CONFUSING CONCEPTS OF HTML AND CSS. BUT IT WON'T BE AFTER THIS LESSON. THE BOX MODEL IS HOW ELEMENTS ARE DISPLAYED ON THE WEB PAGE, HERE YOU WILL ALSO LEARN ABOUT THE INLIE-BLOCK ELEMENT.

CSS DISPLAY PROPERTY AND INLINE-BLOCK VALUE

So far you learned about two types of HTML elements, **block-level**, and **inline-level** elements. Now is the time to get familiar with the 3rd type called an **inline-block element**. Before diving into the box model, you need to understand the CSS display property and options for its value.

CSS DISPLAY OPTIONS

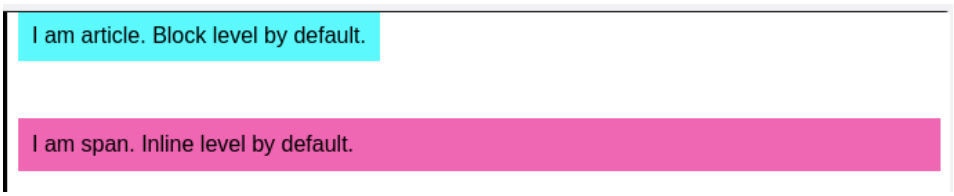
Display property determines exactly how an element is displayed on the web page. The most common options for the value of display property are **block, inline, inline-block,** and **none**.

Block and Inline value

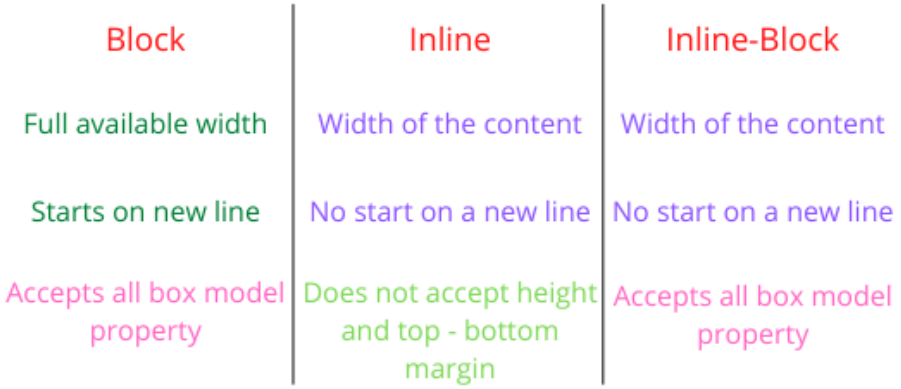
Every HTML element has a default display property. Block-level elements have the display set to block and inline-level element to inline by default. But you can overwrite the default display property. See the example below.

```
<head>
    <style>
        .block-element {
          display: inline; /* Overwritten */
          background: aqua;
        }
        .inline-element {
          display: block; /* Overwritten */
          background: hotpink;
        }
    </style>
</head>
<body>
    <article class="block-element">Article: Block level by default</article>
    <span class="inline-element">Span: Inline level by default</span>
</body>
```

See the result below. A block element (<article>) now behaves like an inline element taking width which is only required by the content. And inline element () now behaves like a block element taking the entire available width.



INTRODUCING INLINE-BLOCK VALUE



We will get to know what is margin in a while. Till then think of it as a way to give spaces around an element.

- Block-level element accepts height and margin on all four sides. There is no catch here.
- The inline-level element does not accept height and top-bottom margin. Height property, top, and bottom margin property do not work on inline-level elements.

That's where **inline-block comes into play**. it provides the best of both values (block and inline). With display property set to inline-block, the element has all qualities of an inline element but now it also accepts height and top-bottom margin.

NONE VALUE

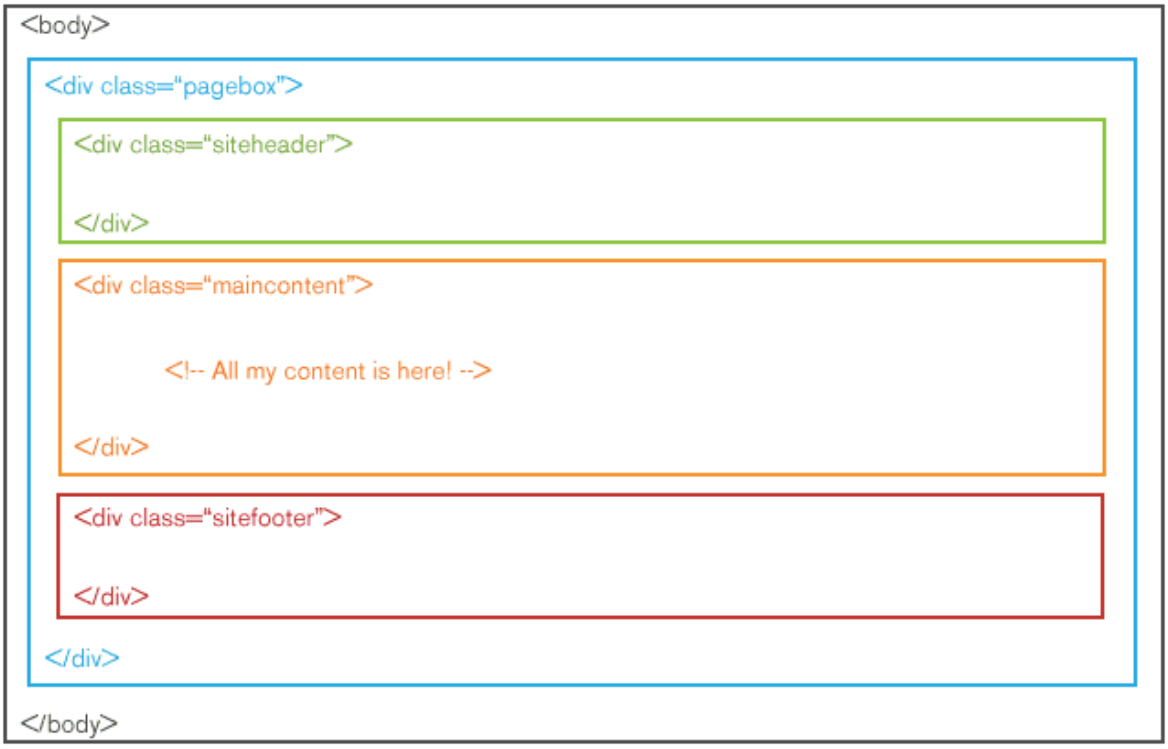
Setting the display property value to none will hide the element and the web page will appear as if the element was never there.

```
div {
  display: none;
}
```

That's how display property and its value works to show an element on the web page,

WHAT IS THE BOX MODEL

According to the box model, **every element on the web page is treated as a rectangular box**. And that rectangular box can be manipulated using different box model properties. Some popular box model properties are **width, height, margin, border, and padding**.



THE BOX MODEL PROPERTIES

The size of an element begins with the width and height properties, and then any margin, padding, or border property values are added from there. By default, the box model is additive in nature.

- MARGIN

Margin is used to provide spaces around an element. The margin falls outside of the border and is transparent.

```css
div {
  margin: 50px; /* Applies margin on all four side. */
}
.div1 {
  margin-top: 20px; /* Applies margin on top of the element. */
  /* Similarly margin-right, margin-bottom, and margin-left can be used. */
}

/* There is also a short hand to define spacing on all four side in one line.
Define top, right, bottom, and left spacing in the same order. */
.div2 {
  margin: 20px 15px 25px 10px; /* order: top, right, bottom, left */
}
```

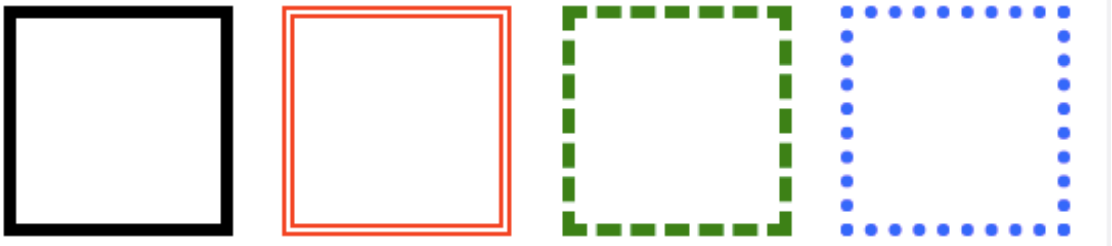
- BORDER

Borders provide an outline around an element and fall between margin and padding. The border property requires three values **width, style, and color**. Border styles are of various kinds, most commonly used are **solid, double, dashed, and dotted**.

```
<head>
    <style>
        .box {
          width: 50px;
          height: 50px;
          margin: 10px;
          display: inline-block;
        }
        .box1 {
          border: 6px solid black;
        }
        .box2 {
          border: 6px double red;
        }
        .box3 {
          border: 6px dashed green;
        }
        .box4 {
          border: 6px dotted blue;
        }
    </style>
</head>
<body>
  <div class="box box1"></div>
  <div class="box box2"></div>
  <div class="box box3"></div>
  <div class="box box4"></div>
</body>
```

Here I have used multiple classes. The box class will target all the div and box1, box2, box3, and box4 are to target individual divs.

Result:



Border Radius

Border radius is pretty interesting. You can manipulate the shape of the div or any other element from recatngular box to a circle. Giving 50% value to the **border-radius** property will make the element a perfect circle given that width and height of the element is same.

```
div {
  border-radius: 50%;
}
```

- PADDING

Padding is used to add spaces within the element. It falls inside of the border. Working with padding is similar to that of margin in terms of syntax and values.

```
/*20px padding on all four sides*/
.div1 {
  padding: 20px;
}
/*Short hand - top right bottom left*/
.div2 {
  padding: 20px 10px 30px 15px;
}
/*Short hand - 10px for top and bottom. 15px for left and right*/
.div3 {
  padding: 10px 15px;
}
```

VISUALIZING THE BOX MODEL

8020lessons.in

TOTAL WIDTH AND HEIGHT OF THE BOX

The **total width** of the box can be calculated using the following formula.

margin-left + border-left + padding-left + width + padding-right + border-right + margin-right

And, **the total height** of the box can be calculated with the following formula.

margin-top + border-top + padding-top + height + padding-bottom + border-bottom + margin-bottom

THE BOX SIZING

You know that the box model is additive by default. Any margin, border, or padding will be added to its final width and height. But you can change this behavior with the box-sizing property.

For example if a div is 100px wide and it has 10px margin, 10px padding, and 2px border on all sides, the final width becomes 144px.

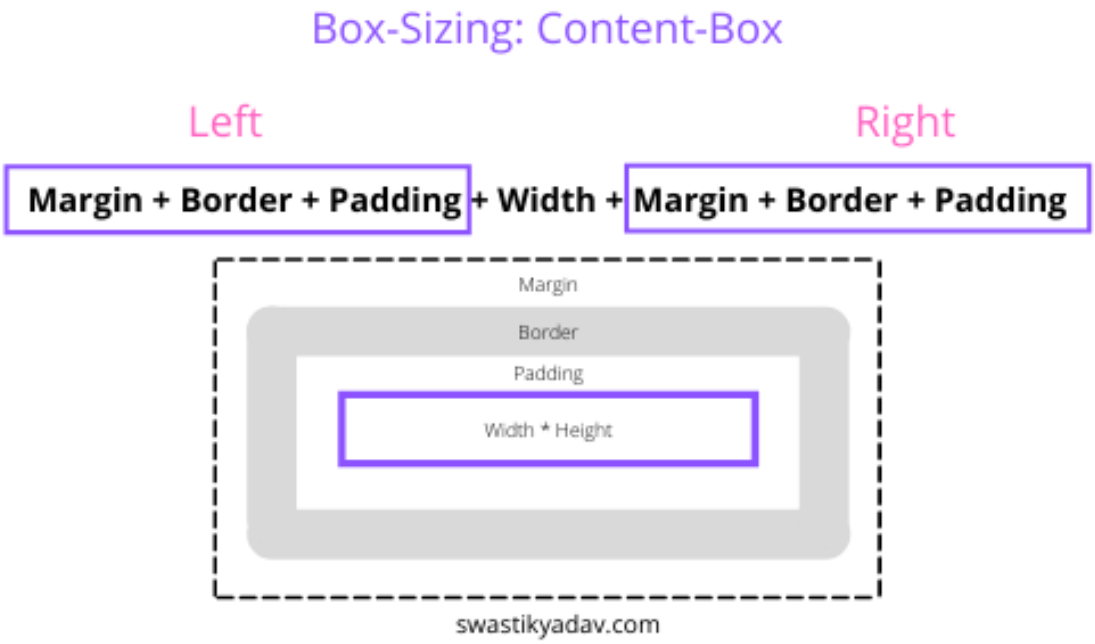
With box-sizing, you can add padding and border without increasing the size of the div (keeping it 100px only). However, any margin will still be added to the final width and height.

The box-sizing property has two major values **content-box** and **border-box**.

- CONTENT BOX

Content Box is the default setting of box-sizing which is being additive in nature. So, there isn't any catch here.

```
/*Default setting*/
div {
  box-sizing: content-box;
}
```



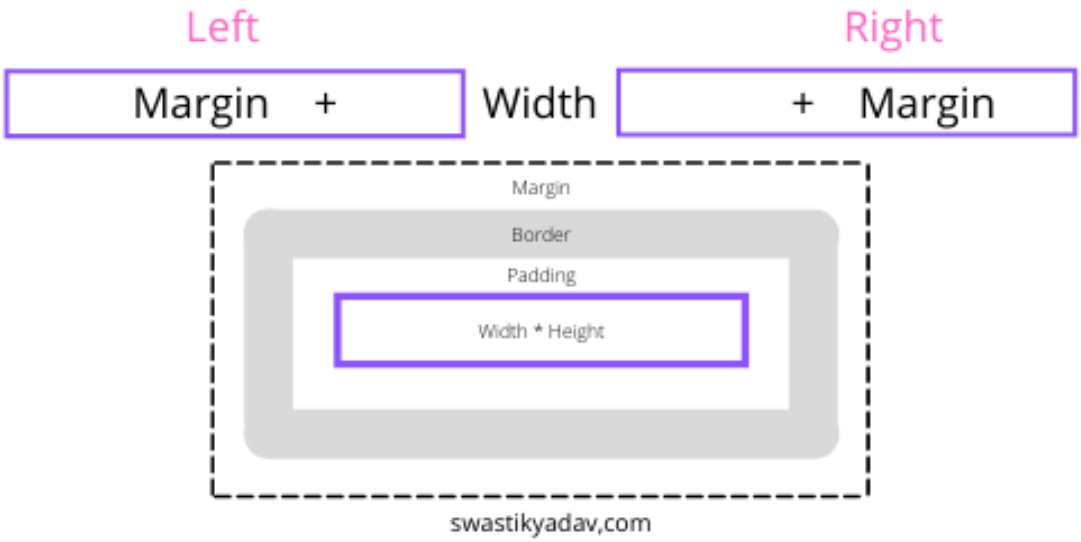
- BORDER BOX

With border-box as the value of the box-sizing property, any padding or border will not be added to the size of the element. However, any margin will still be added to the size of the element.

Border Box makes the content shrink proportionally to the padding and border.

```
/*Setting to border-box*/
div {
  box-sizing: border-box;
}
```

Box-Sizing: Border-Box



CRAFTING AND POSITIONING MODERN LAYOUTS

AS DISCUSSED IN THE LAST LESSON EVERY ELEMENT IN HTML IS A BOX. HENCE IN THIS LESSON, YOU WILL LEARN TO POSITION THESE BOXES IN ANY MANNER YOU WANT. THIS SUPERPOWER GIVES YOU THE ABILITY TO POSITION CONTENT AND ELEMENTS ON A PAGE IN NEARLY ANY IMAGINABLE WAY.

CSS POSITION PROPERTY.

To move HTML elements around the page CSS position property is used. It accepts various values like **static, relative, absolute, and fixed**. These values change the flow of the element on the page.

To be able to position elements, you also need to know about box offset properties which we will discuss in a while. It defines: In which direction and by how many units the box should move.. Let's study each type of positioning one by one.

Position Static

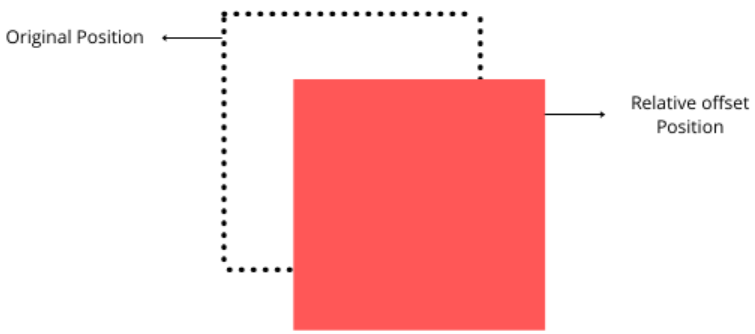
Static is the default value of any element's position property. Static element exists in the normal flow of that page and it doesn't accept any box offset properties.

Position Relative

The relative element appears within the normal flow of the page, but it allows box offset properties. Box offset properties are applied with the element's original position as the origin.

```
div {
  width: 80px;
  height: 80px;
  position: relative;
  top: 20px;
  left: 20px;
}
```

- **left: 20px;** pushes the element 20px to the right from original position.
- **top: 20px;** pushes the element 20px down from its original position.



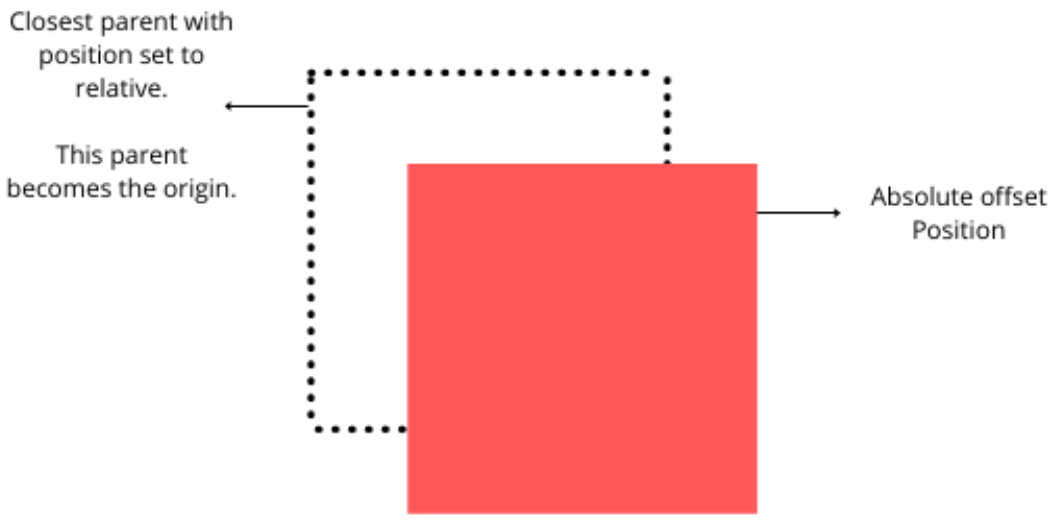
Position Absolute

The absolute element does not appear in the normal flow of the page. Absolutely positioned element moves in relation to the closest relatively positioned parent. If no parent is positioned relatively, then the element moves in relation to the **<body>**.

So, simply put:
For an absolute element, the box offset properties are applied with the **closest relative parent** as the origin. And if there is no relative parent <body> becomes the origin for box offset properties.

```
<head>
    <style>
        .parent {
          position: relative;
        }
        .child {
          position: absolute;
          top: 20px;
          left: 20px;
        }
    </style>
</head>
<body>
    <section class="parent">
      <article class="child">
      </article>
    </section>
</body>
```

- **top: 20px** pushes the child 20px below the parent's top-border.
- **left: 20px** pushes the child 20px to the right of parent's left-border.



Position Fixed

The fixed element does not appear in the normal flow of the page and it works exactly like the absolute positioning. The only two differences are:

1. A fixed element moves in relation to the browser window.
2. A fixed element does not scroll with the page. It remains at the specified box offset values.

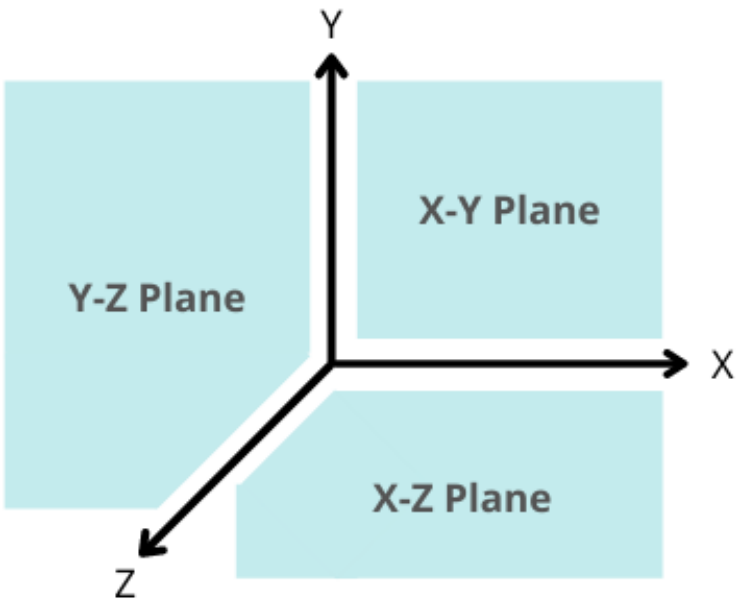
A real-world use-case of fixed positioning is the header of a website which is fixed at **top: 0** and **left: 0**.

```
header {
  width: 100%;
  position: fixed;
  top: 0;
  left: 0;
}
```

Z-INDEX

Earlier when I mentioned that position absolute and position fixed changes the normal flow of the page, what I meant was it puts the element on the z-axis.

Normally HTML elements are positioned on a 2D cartesian plane with x and y-axis, but position absolute and position fixed puts it on the 3D z-axis. That's what Z-Index is.



8020lessons.in

Z-index is how many units you want to move the element on the z-axis.

```css
.blue-box {
  z-index: 5;
}
.red-box {
  z-index: 10;
}
```

The red box has a higher Z-Index value so it appears above the blue box.

MODERN LAYOUTS WITH FLEXBOX

Flexbox also known as Flexible box gives you complete control over the alignment, direction, order, and size of the boxes. It makes it very easy to build modern web page layouts.

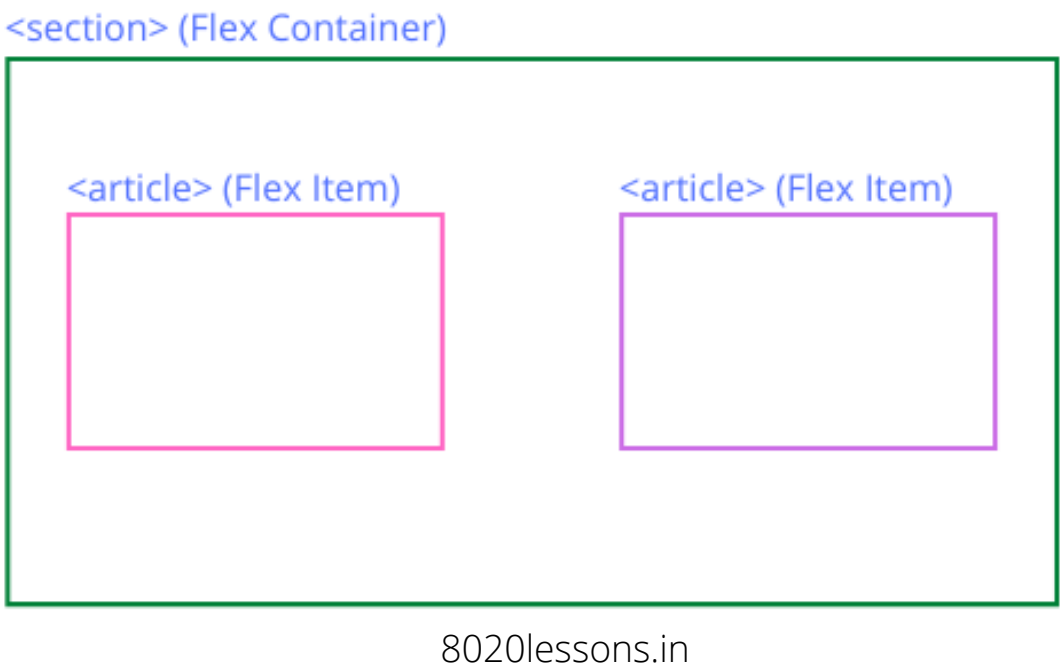
WHAT IS FLEXBOX?

Flex is one of the values for the CSS display property. Flexbox uses **flex containers and flex items** to define the positioning and layout.

1. **Flex Container:** The element with display property set to flex becomes the **Flex Container.**
2. **Flex Items:** All direct child of the flex container becomes the **Flex Items**.

```
<head>
    <style>
      .container {
        display: flex;
      }
    </style>
</head>
<body>
    <section class="container">
      <article>Article 1</article>
      <article>Article 2</article>
    </section>
</body>
```

In the above example, <section> with class container is the **flex container** and it's direct <article> childrens are the **flex items**.



Display flex tells the browser that the container element should be rendered with flexbox instead of the default box model. The default behavior of display flex is to align all the flex items in a row.

FLEX CONTAINER PROPERTIES

Justify Content: Horizontal alignment of items.

Justify content property defines the horizontal alignment of its items. Justify content has the following values.

- **flex-start**: Align items at row-start
- **flex-end**: Align items at the row-end
- **center**: Align items at row center
- **space-between**: Space between flex items
- **space-around**: Space around flex items

Align Items: Vertical alignment of items.

Align items property defines the vertical alignment of its items. **Values for align-items are similar to that of justify-content.**

.The difference being Align-Items aligns the items vertically and Justify-Content aligns the items horizontally.

Flex-Wrap: Create a grid layout.

When flex-items row width exceeds the width of flex-container, flex-items flow off the edge of the container. This behavior can be changed with the flex-wrap property. Flex-Wrap have only two values:

- **no-wrap:** this is the default value. Flex items flow off the page.
- **wrap:** this will wrap the flex-items on the next line and creates a grid**.**

```css
.container {
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
}
```

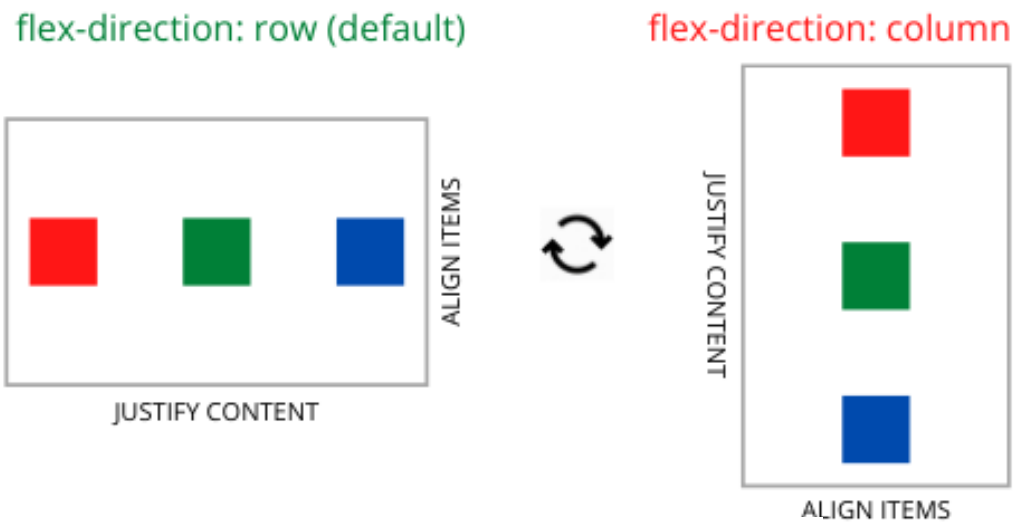
If you want to make a grid with flex-wrap, make sure to give flex-items a min-width, else flex-items will shrink to fit the container.

Flex Direction: Row or Column.

By default, flex containers align their flex items in a row. You can change the flex container direction with the flex-direction property.

It takes four values:

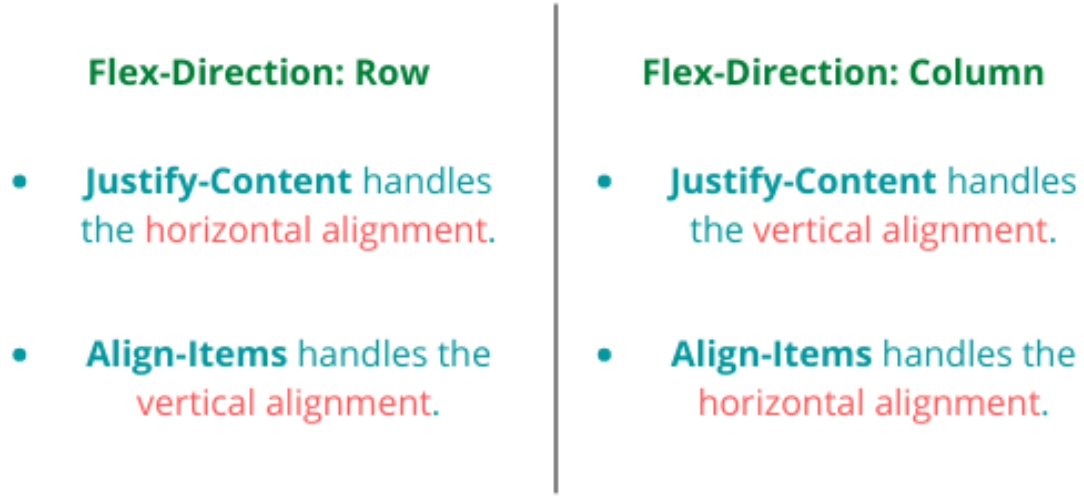
- row
- row-reverse: Reverses the row
- column
- column-reverse: Reverses the column



8020lessons.in

By default, justify-content is used for the horizontal, and align-items for the vertical alignment. But when flex-direction is changed to column, justify-content and align-items switch their roles.

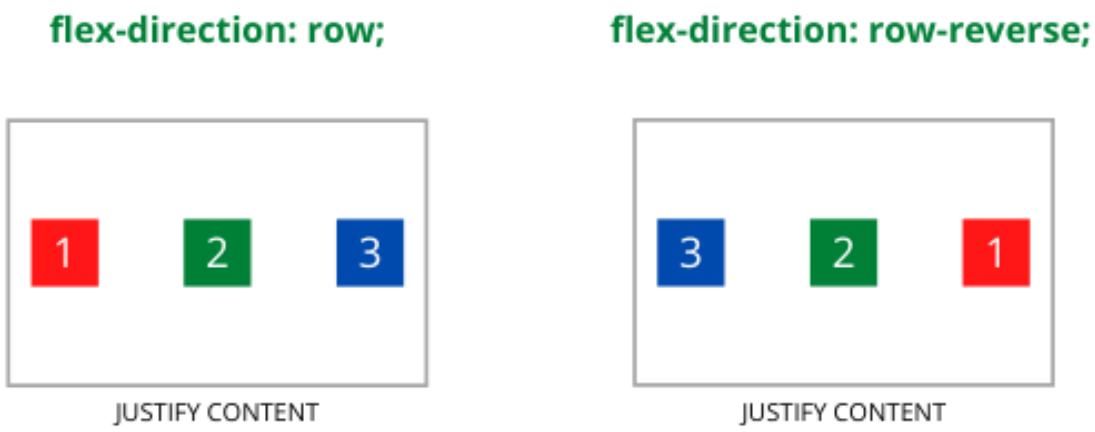
With flex-direction set to column, justify-content is used for the vertical, and align-items for the horizontal alignment.



8020lessons.in

Row-Reverse and Column-Reverse.

Row-Reverse and Column-Reverse just reverse the direction of row and column respectively. There is no catch here.



8020lessons.in

An important use case.

Flex Direction is very useful to create responsive web designs. With flex-direction and media query, you can define flex-container to be horizontal on desktop screens and vertical on mobile screens.

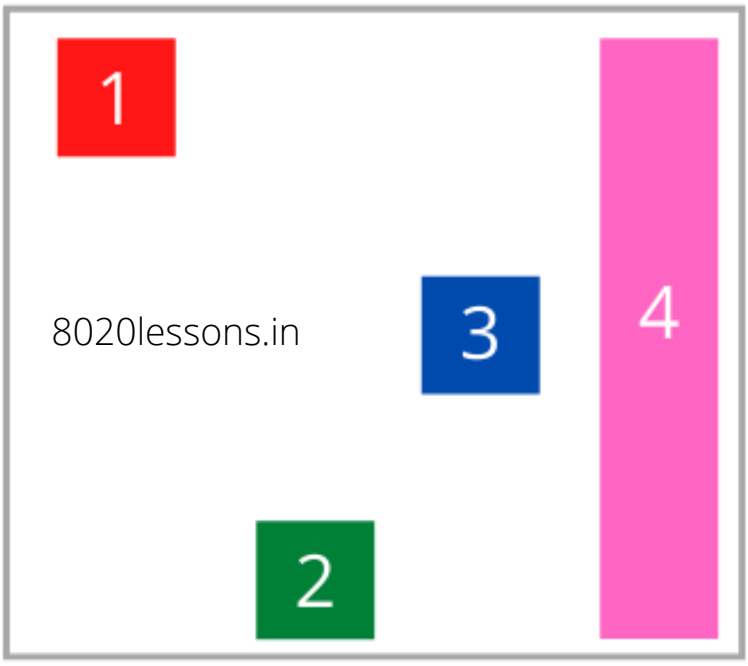
FLEX ITEMS PROPERTY: ALIGN SELF

Align-Self - Align individual flex-items vertically.

Align-Self is similar to align-items, the only difference is that **align-items is applied on flex-container** and **align-self is applied on individual flex-item**.

Its values are also the same as align-items: **center, flex-start, flex-end, stretch, baseline**.

```
.container {
   display: flex;
}

.box-2 {
   align-self: flex-end;
}

.box-3 {
   align-self: center;
}

.box-4 {
   align-self: stretch;
}
```



ENHANCE UI WITH WEB TYPOGRAPHY

TYPOGRAPHY IS THE APPEARANCE OF ALL THE TEXT ON YOUR WEB PAGE. IT IS THE ARTISTIC IMPRESSION OF HOW THE TEXT LOOKS, FEELS, AND READS. IN TECHNICAL TERMS, IT IS THE SIZE, COLOR, WEIGHT, AND STYLE OF THE TEXT.

TYPEFACE VS FONT

A typeface is what we see, the artistic impression of how the text looks, feels, and reads.

A font is a file that contains a typeface. The typeface can be accessed by the system through a font.

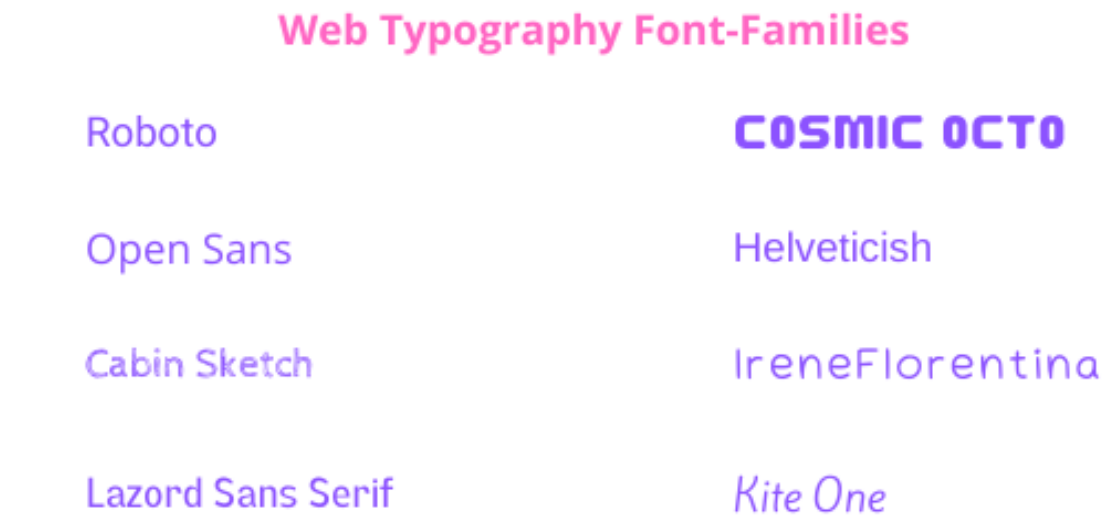
Font Family

Every word on a web page uses a font. The font family declares which font should be used to display the text. The value of font-family property contains multiple font names.

The first font is the default font choice, if the default font is not available the next font name is used as a fallback font.

```css
body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
}
```

Here are a few font-families examples.

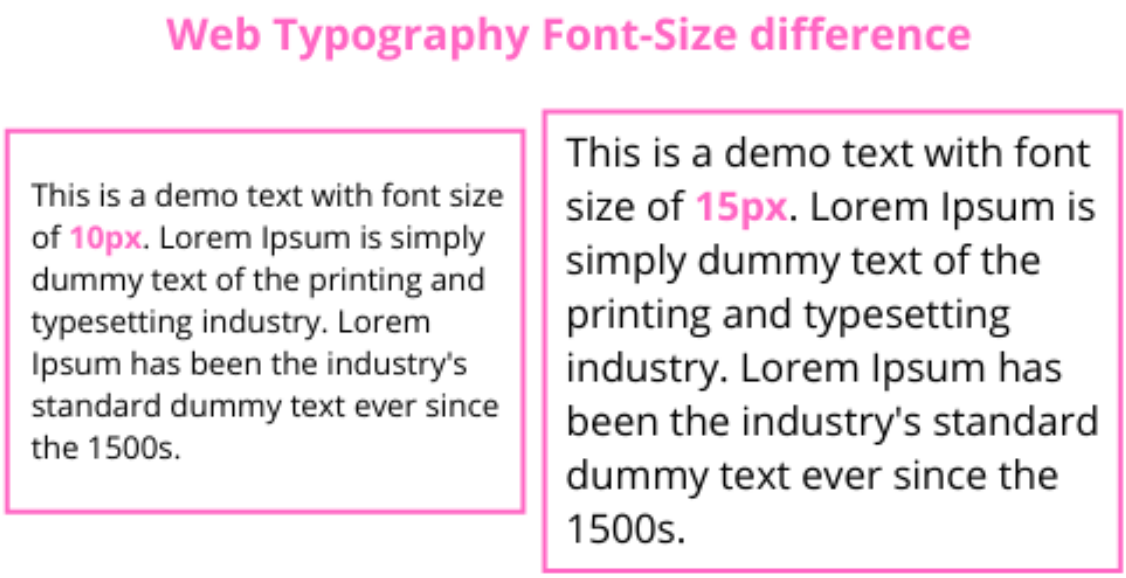


FONT-SIZE

Font-Size property is used to control the size of the text on the web page using common length values like pixels, em units, and percentages.

```
body {
  font-size: 15px;
}
```

See the example below.



The font-size of 15px is much easier to read then the font-size of 10px.

LINE HEIGHT

Line Height is the distance between two lines of text. The line-height property takes length values.

The best practice is to set the line-height to 1.5 times the font size of the text.

```
body {
  font-size: 22px;
  line-height: 1.5;
}
```

The above snippet will set line-height to 1.5 times of 22px;

Set the line height in the below interactive example to see it live in action.

Interactive line-height example.

TEXT ALIGN

Text Align property aligns the text at a horizontal level. It accepts 5 values: **left, right, center, justify, and inherit**.

The following CSS sets all paragraph text to be center-aligned:

```
p {
  text-align: center;
}
```

LETTER SPACING AND WORD SPACING

Using the letter-spacing property you can specify the space between the letters on a page.

Word Spacing allows specifying the space between the words on a page.

With **none** being the default spacing value for both letter-spacing and word-spacing.

Below is an interactive example. Change the letter-spacing and word-spacing below and see it in action.

Interactive letter-spacing and word-spacing example

WEB SAFE FONTS

There are few default fonts on every device. You can safely use those fonts as they are pre-installed on every possible device, Hence these fonts are known as Web-Safe Fonts.

Here is the list of Web-Safe Fonts.

- Arial
- Garamond
- Lucida
- Tahoma
- Trebuchet
- Courier
- Georgia
- Times
- Verdana

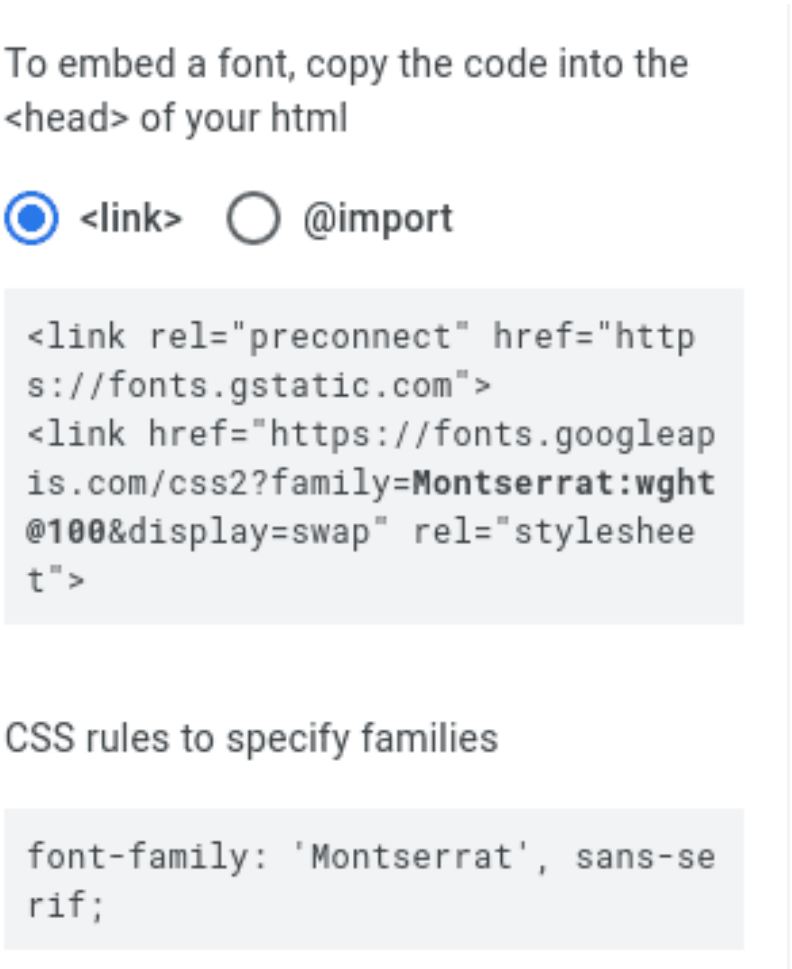
EMBEDDING WEB FONTS

You are not limited to using only web-safe fonts. You can use a lot of different fonts. For example, let's see how you can use a Google Font.

Go to Google Fonts and select the font you want to use.

There are two ways to use a google font one is through **HTML link tag** and the other is through **CSS @import**.

1 Select link and you will get an HTML link tag to copy and paste in your HTML head. Once pasted you can simply specify that font in your CSS font-family property.



OR

2 Select @import and you will get CSS @import code to copy and paste on top of your CSS file. Once pasted you can simply specify that font in your CSS font-family property.

That's how you enhance the UI and make the user's reading experience a lot better.

BACKGROUNDS AND LISTS

APPLY SOLID COLORS, AN IMAGE, A GRADIENT, OR A COMBINATION OF THESE ON BACKGROUNDS. CREATE AND STYLE LISTS ON YOUR WEB PAGE.

TYPEFACE VS FONT

In this lesson, you will learn to apply different types of backgrounds like colors, images, and gradients to the web page. A well-applied background contributes to the overall appearance of your website.

Background Colors

CSS Background property is used to add a background (color) to an element. Background color value can be defined as Keywords, HEX color code, RGB, or HSL.

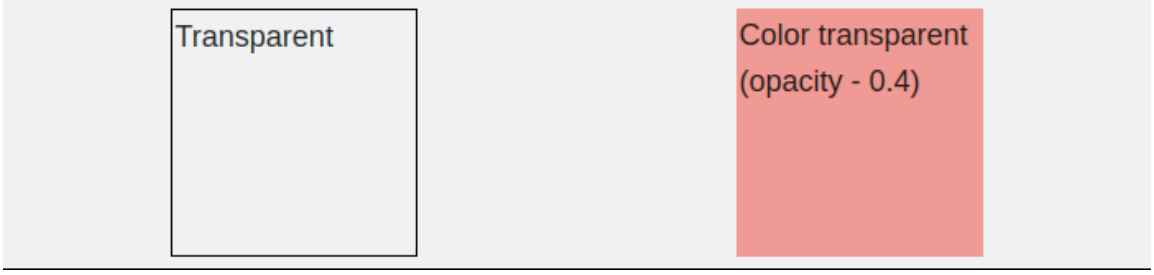
```
div {
  background: #ff0000;
}
```

Transparent Background

You can set an element's background is **transparent**. But if you want it to be color transparent, you can use alpha value in rgba.

```
div {
  background: rgba(255, 0, 0, 0.4);
}
```

The last value in rgba (a) is alpha which defines the opacity. Opacity 0 means completely transparent and 1 means no transparency at all.



BACKGROUND IMAGES

You can also set an image to an element's background with background property and specifiying URL in its value.

```css
section {
  background: url("image-path");
  background-repeat: no-repeat;
  background-size: cover;
  background-position: center;
}
```

Background Repeat

By default background image is repeated horizontally as well as vertically to fit the element size. This is handled by setting the **background-repeat** property to **no-repeat**.

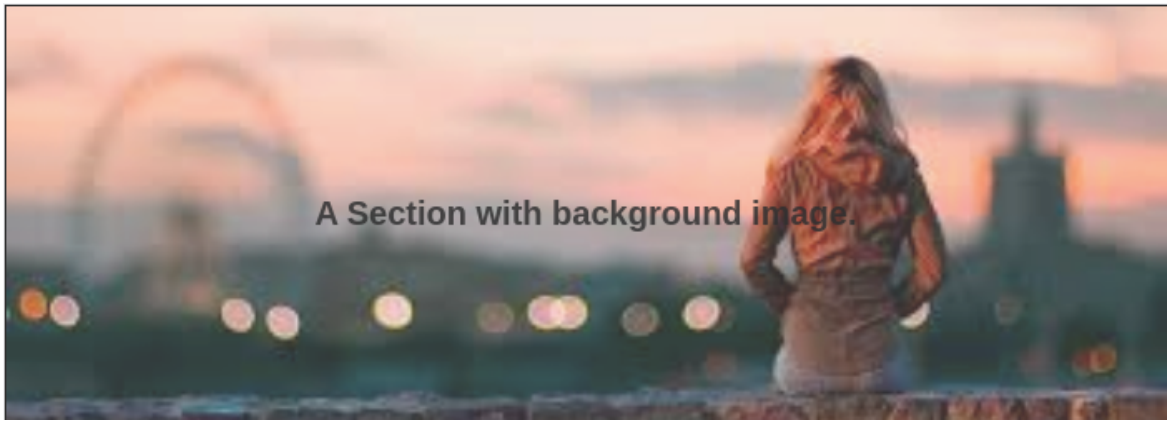
Background Size

Setting the **background-size** property to **cover** makes the image fit the element's size.

Background Position

Background position specifies the position of the image within the element. the background-position property takes two values a horizontal offset and a vertical offset. Its value can be set to top, bottom, left, right, and center, or pixels, and percentages.

Here is an example of adding a background image to a section element with background properties as the above snippet.



BACKGROUND GRADIENTS

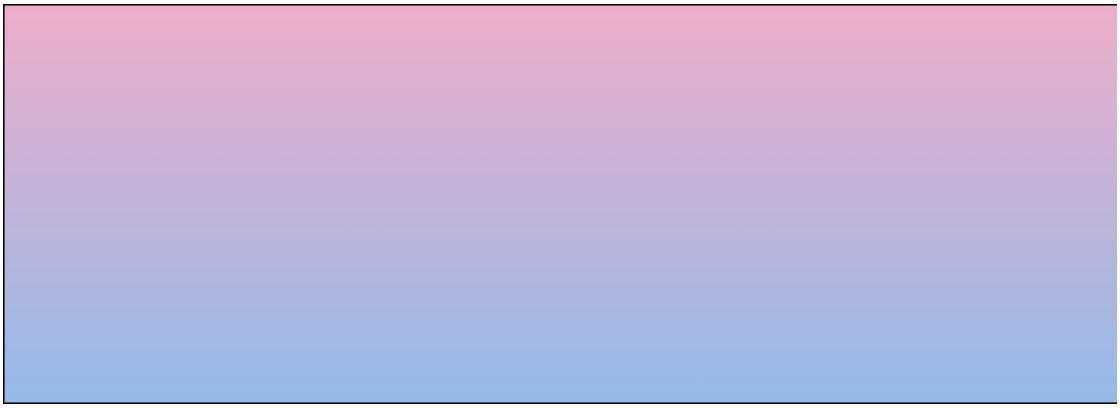
The color gradient is the smooth transition from one color to another. It is treated as an image in a CSS background.

Linear Gradient Background

Linear gradient takes two color values, the first one is the beginning color value and the second one is the ending color value.

```
div {
  background: linear-gradient(#eeaeca, #94bbe9);
}
```

RESULT (Gradient direction top to bottom)



By default, the gradient colors move from **top to bottom**. You can specify the direction of gradient like the example below. To start the gradient from left and end at right use the value **right** in the gradient. And to start gradient at top-left and end at right-bottom use the value **right bottom**.

```
div {
  background: linear-gradient(to right, #eeaeca, #94bbe9);
}
```

RESULT (Gradient direction left to right)

Radial Gradient Background

Radial gradient works from the inside (first color value) to the outside (second color value) of an element creating a circular color transition.

```
div {
  background: radial-gradient(#eeaeca, #94bbe9);
}
```

RESULT (pink color starts from the center and ends with blue color at the edges)

So, that's how you apply colors, images, and gradients to the background of an element. It can be time-consuming and confusing to create the perfect gradient you need. You can also use gradient generator tools like this one.

CREATING AND STYLING HTML LISTS

Lists are a very important part of HTML which lets you display data in a list format. Think of a to-do list or navigational links, they are all HTML lists. In this lesson, you will learn how to create and style HTML lists.

Creating Lists

HTML mainly gives you two types of lists to work with: **unordered** and **ordered** lists.

An **unordered list** is the list of items where the order of items does not matter. For example a shopping list. For unordered list ** element** is used and each list is wrapped in ** (list item) element**.

```
<ul class="shopping-list">
  <li>Milk</li>
  <li>Bread</li>
  <li>Eggs</li>
  <li>Chocolates</li>
</ul>
```

The ordered list is similar to the unordered list, the only difference being that here the order of items does matter. For example a set of instructions to follow. For ordered list ** element** is used with ** element** for each item.

```
<ol class="dc-movie-order">
  <li>The Dark Knight</li>
  <li>Man Of Steel</li>
  <li>Wonder Woman</li>
  <li>Justice League - Snyder Cut</li>
</ol>
```

STYLING HTML LISTS

Now you know how to create all sorts of lists. So, let's take a look at how you can style any HTML list.

List Style Type

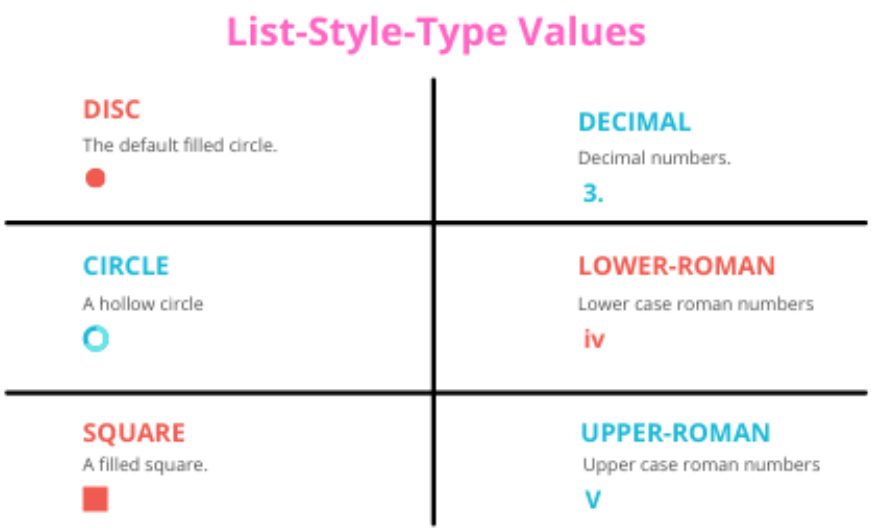
With list-style-type property, you can style the list item marker. By default, the list item marker for the unordered list is a filled circle (disc). Here is an example of changing the default marker to a square.

```
<head>
    <style>
        .square-marker {
          list-style-type: square;
        }
    </style>
</head>
<body>
    <ul class="square-marker">
      <li>List item 1</li>
      <li>List item 2</li>
      <li>List item 3</li>
    </ul>
</body>
```

RESULT:

- List item 1
- List item 2
- List item 3

Here is the list of major values for the list-style-type property.

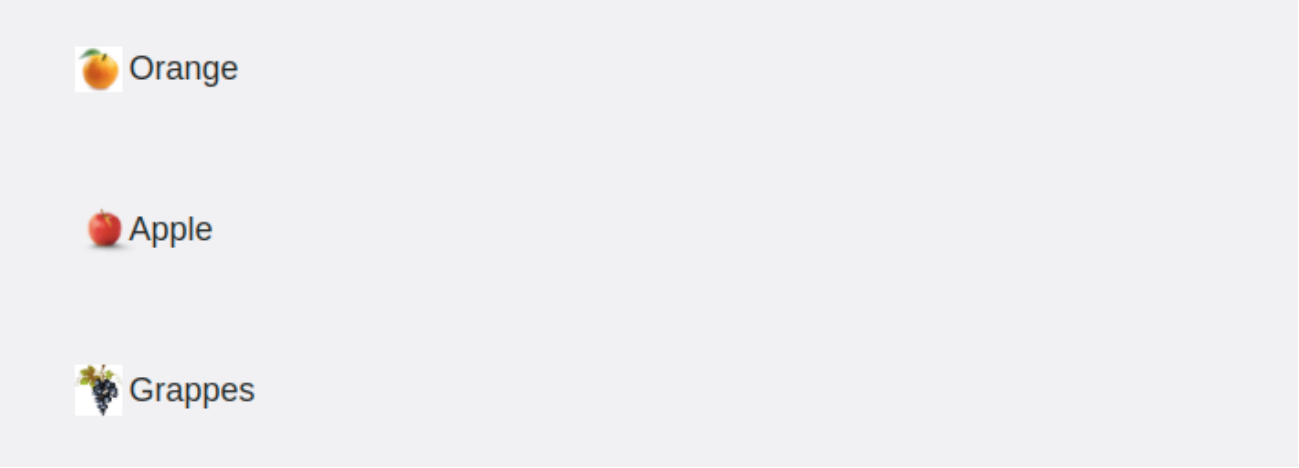


Using Image as List item marker

Yes, it is possible. You can also set the list item marker as an image. Set the list-item-type for li as none, and set the image URL in the background property.

```
<head>
    <style>
        .orange {
          background: url("image-path");
          background-position: 0 50%;
          background-repeat: no-repeat;
          padding: 10px;
        }
        /* Similarly set for apple and grappes */
    </style>
</head>
<body>
    <ul>
      <li class="orange">Orange</li>
      <li class="apple">Apple</li>
      <li class="grappes">Grappes</li>
    </ul>
</body>
```

RESULT:



This was all about creating and styling all sorts of HTML lists.

ORGANIZE DATA WITH HTML TABLES

HTML TABLES ALLOW US TO ORGANIZE TABULAR DATA IN A STRUCTURED MANNER THAT IS EASIER TO READ AND DIGEST FOR THE USERS. IN THIS LESSON, WE WILL LEARN TO ORGANIZE AND STYLE INFORMATIONAL DATA IN A TABULAR FORMAT WITH HTML TABLES.

THE TABLE MARKUP

Tables present data in rows and columns format. HTML provides various elements to create a table. Starting with a **<table> element**. HTML tables can be defined in 3 parts.

1. Table Head (**<thead>**)
2. Table Body (**<tbody>**)
3. Table Footer (**<tfoot>**)

Here is an image showing all 3 parts of the table.

8020lessons.in

Table Rows

Each of those 3 parts wraps table row (**<tr>**) elements. Which defines content in each row.

Table Data

Table data defines content in each cell of the row. Two elements <th> and <td> are used. Both elements work the same way, the only difference is in semantics.

- **<th>** is used for the content in the cell of rows of table head.
- **<td>** is used for the content in the cell of rows of the table body.

Table Head

The table head defines the heading for each column.

```
<table> <!-- Table -->
  <thead> <!-- Table's Head -->
    <tr> <!-- Table's Head's Row -->
      <th>Book</th>
      <th>Author</th>
      <th>Price</th> <!-- Table's Head's Row's Cell-3 -->
      <th>Rating</th>
    </tr>
  </thead>
</table>
```

Table Body

Table body is where the main content lies. It fills the data in each cell.

```
<table>
  <thead>
    ...
  </thead>
  <tbody> <!-- Table body -->
    <tr> <!-- Table body's row-1 -->
      <td>$100 dollar startup</td>
      <td>Chris Guillebeau</td>
      <td>$29</td> <!-- Table body row-1 cell-3 -->
      <td>9/10</td>
    </tr>
    <tr> <!-- Table body's row2 -->
      <td>The lean startup</td>
      <td>Eric Ries</td>
      <td>$15</td> <!-- Table body row-2 cell-3 -->
      <td>8/10</td>
    </tr>
    <tr> <!-- Table body's row3 -->
      <td>Zero to One</td>
      <td>Peter Thiel</td>
      <td>$25</td> <!-- Table body row-3 cell-3 -->
      <td>7/10</td>
    </tr>
  </tbody>
</table>
```

Under table body (<tbody>) use table row (<tr>) as many times as you want to create a new row.

Table Footer

The table footer contains data that outlines the content of the table.

```
<table>
  <thead> ... </thead>
  <tbody> ... </tbody>
  <tfoot>
    <tr>
      <td>Price Total</td>
      <td></td>
      <td></td>
      <td>$69</td>
    </tr>
  </tfoot>
</table>
```

RESULT: Here is the result of the complete table markup.

Book	Author	Price	Rating
$100 dollar startup	Chris Guillebeau	$29	9/10
The lean startup	Eric Ries	$15	8/10
Zero to One	Peter Thiel	$25	7/10
Price Total			$69

I know this does not look much appealing. But wait till we style this table with CSS.

STYLING THE TABLE

The above table doesn't look much appealing and that's our next job. You can add borders, table striping, and text alignment within the table.

Table Borders

Border can be added to all elements of the table. You should add borders as per your design needs.

Border-Collapse

The default border of the table is stacked with the border of rows, making the table border thicker than the border width provided. To overcome this behavior set the value of border collapse to collapse. The default value of border-collapse is **separate**.

```css
table {
  border-collapse: collapse;
}
th,
td {
  border: 1px solid black;
  padding: 10px 15px;
}
```

Book	Author	Price	Rating
$100 dollar startup	Chris Guillebeau	$29	9/10
The lean startup	Eric Ries	$15	8/10
Zero to One	Peter Thiel	$25	7/10
Price Total			$69

Border-Spacing

Border spacing provides a specified amount of spacing between the borders.

```css
table {
  border-collapse: separate;
  border-spacing: 4px;
}
table,
th,
td {
  border: 1px solid black;
}
th,
td {
  padding: 10px 15px;
}
```

Table example with border spacing.

Book	Author	Price	Rating
$100 dollar startup	Chris Guillebeau	$29	9/10
The lean startup	Eric Ries	$15	8/10
Zero to One	Peter Thiel	$25	7/10
Price Total			$69

As per your requirement you can experiment with adding borders to different table elements.

Table Striping

You can add background color to alternative rows. This is called table striping.

Set background of even row to a color. Use **:nth-child()** selector to set background of alternative rows.

```
tbody tr:nth-child(2) {
  background: #red;
}
```

Above snippet will set background of 2nd row to red.

```
tbody tr:nth-child(odd) {
  background: aqua;
}
```

The above snippet will set the background of every alternative row to #f0f0f2.

Here is an example of table striping.

Book	Author	Price	Rating
$100 dollar startup	Chris Guillebeau	$29	9/10
The lean startup	Eric Ries	$15	8/10
Zero to One	Peter Thiel	$25	7/10
Price Total			$69

With this lesson, we came to the end of our **8020CSS course**. And remember consistent deliberate practice is the key to master any skill.

In the next few lessons, we will discuss the theoretical stuff like **best practices**, **philosophy**, **how CSS works under the hood**, **media query**, **how to research**, **how to approach a new project,** e.t.c.

RESPONSIVE WEB PAGES WITH MEDIA QUERY

MEDIA QUERY IS A CSS TECHNIQUE TO MAKE THE WEB PAGE RESPONSIVE AND LOOK GOOD ON DEVICES OF ALL SIZES. IT INCLUDES A BLOCK OF CSS PROPERTIES ONLY IF A CERTAIN CONDITION IS TRUE OR A BREAKPOINT MATCHES.

WHAT IS MEDIA QUERY?

Media query is a CSS technique to make the web page responsive and look good on devices of all sizes. It includes a block of CSS properties only if a certain condition is true or a breakpoint matches.

Device BreakPoints

There are so many screen sizes that it is impossible to write a media query for all of them. So, we follow 5 breakpoints that cover all major screens. You can insert as many breakpoints as you want, there is no restriction here.

- Extra Large Devices: **Above 1200px**
- Large Devices: **Above 992px**
- Medium Devices: **Above 768px**
- Small Devices: **Above 600px**
- Extra Small Devices: **Below 600px**

Let's look at an example.

```css
article {
  width: 60%; /* Above 768px article width is 60% */
  font-size: 22px;
}

@media all and (max-width: 768px) { /* Below 768px article width is 100% */
  article {
    width: 100%;
    font-size: 18px;
  }
}
```

This is "Desktop First Design" **Above 768px article width is 60%** and **Below 768px article width is 100%**.

Hide elements for smaller screen

It is a very common practice to show elements for a larger screen and hide them for a smaller screen.

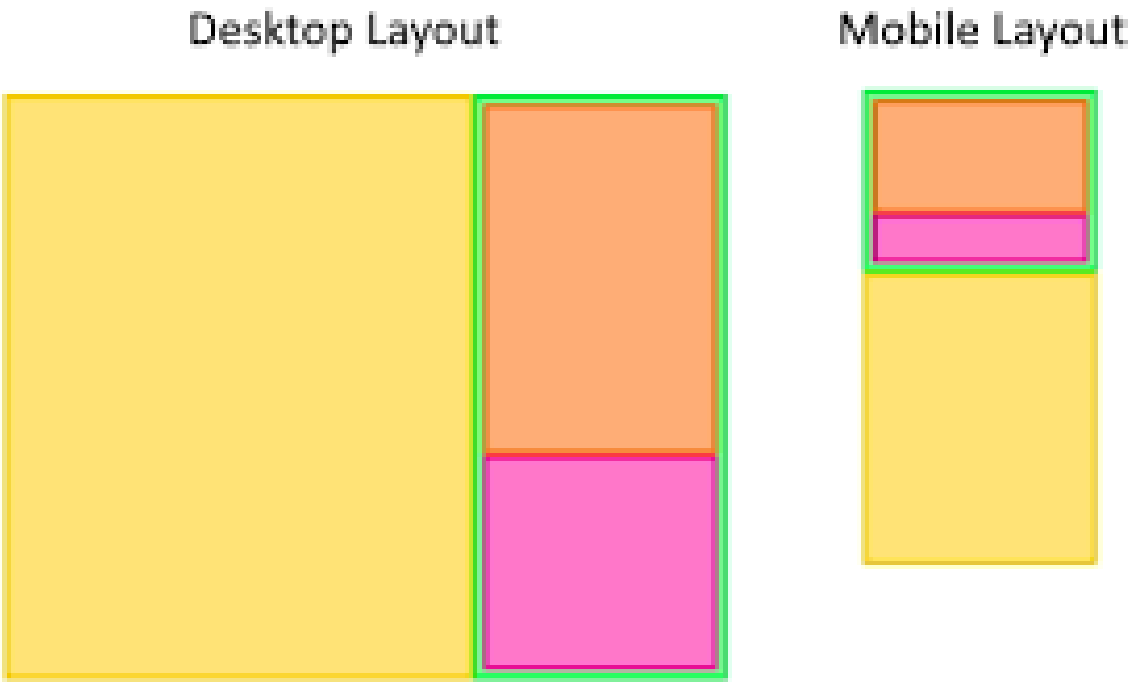
Here is an example of hiding an image for a screen size below 600px.

```css
.image {
  width: 60%; /* Above 768px article width is 60% */
}

@media all and (max-width: 600px) { /* Below 768px article width is 100% */
  .image {
    display: none;
  }
}
```

Change flex-direction to column for smaller screens.

This is also a common technique to change flex-direction to column for a smaller screen. For Desktop show flex items side by side and for mobile show them in a column.

```css
.container {
  display: flex;
  flex-direction: row;
}

@media all and (max-width: 7680px) { /* Below 768px flex direction is column
*/
  .container {
    flex-direction: column;
  }
}
```



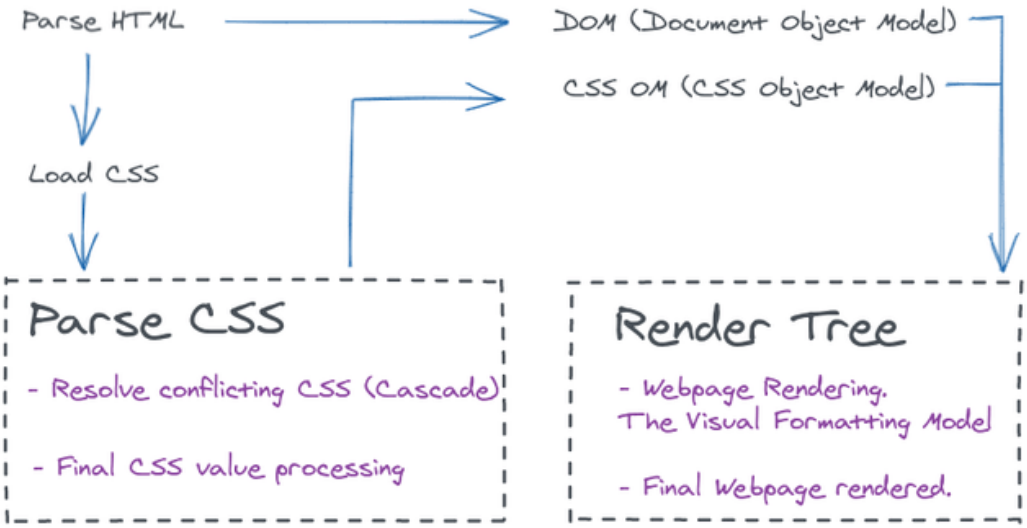
HOW CSS IS PARSED ON THE BROWSER?

- HOW BROWSER PARSES THE CSS?
- HOW CSS RENDERS THE STYLE?
- HOW THE LAYOUT OF THE PAGE IS DECIDED?

CSS IS PARSED ON THE BROWSER IN 3 STEPS.

1. Resolve conflicting CSS declarations.
2. Process final computed CSS values.
3. Visual Formatting Model.

Here is a complete flow of these 3 steps.



Let's understand each step.

1 Resolve Conflicting CSS declarations.

- Different stylesheets are combined together.
- Resolving conflicts when more than one rule is applied to a certain element.

For example: Which color to apply when defined twice for the same element.
Here Cascade & Specificity is at play.

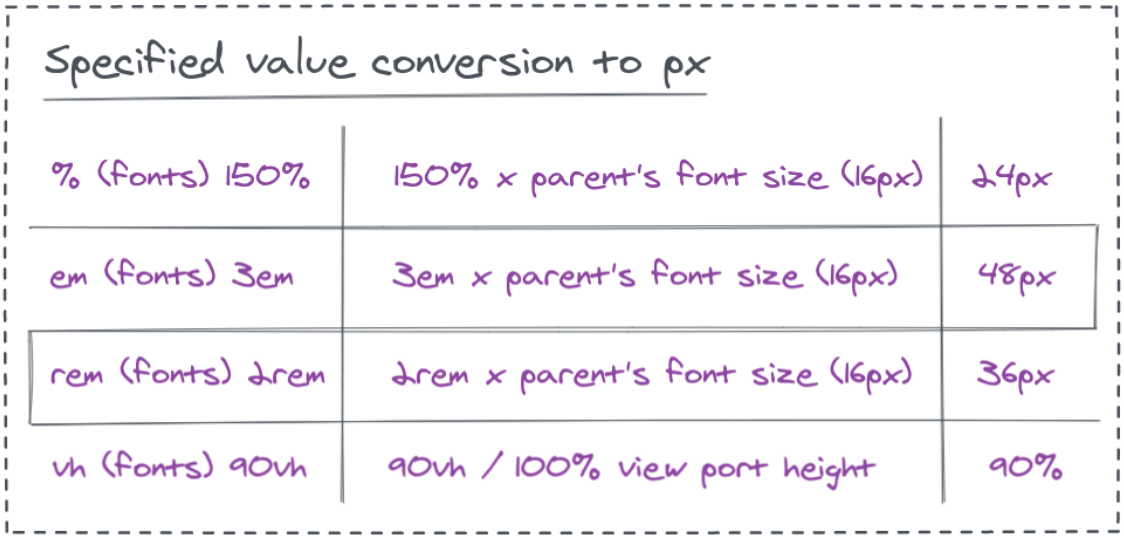
```css
.my-class {
  color: red;
  text-align: center;
  font-size: 18px;
  color: blue;
}
```

The blue color will be applied due to Cascade & Specificity.

2 Processing Values

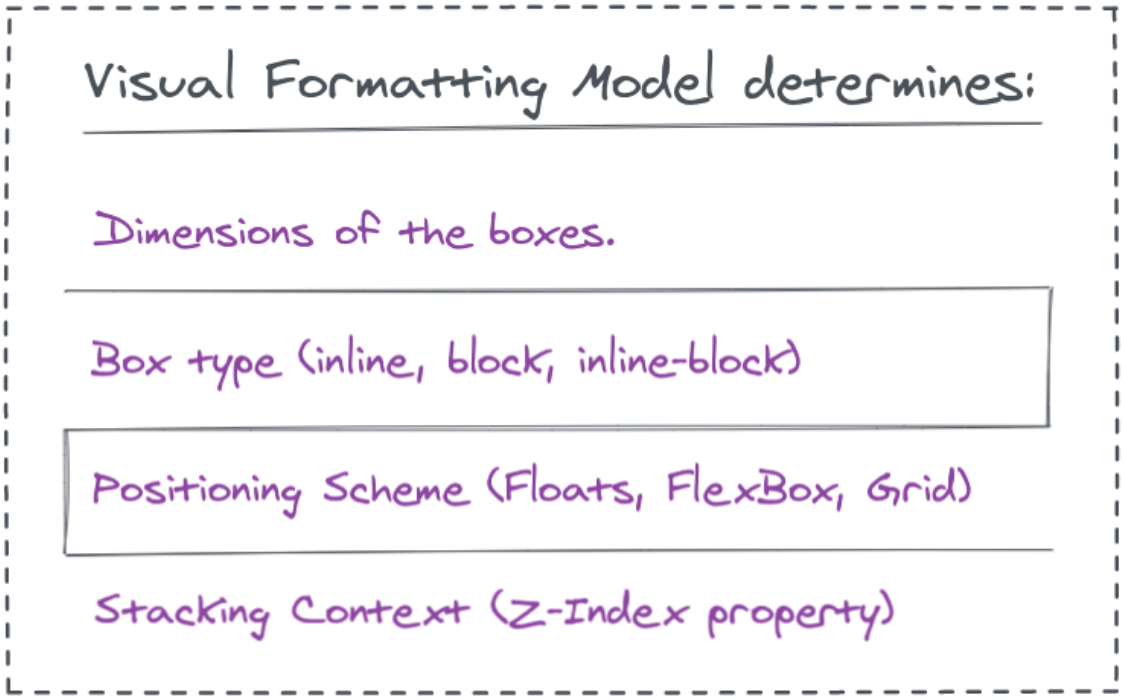
All relative units **(%, em, rem, vh, vw)** are finally converted to the absolute unit, pixels **(px)**.

At this phase, all CSS values become computed values not the specified values.



3 The visual formatting model.

This is the algorithm that calculates boxes & determines the layout of these boxes, for each element in the render tree, to determine the page's final layout.



Theoretically, that's the basics of how CSS is parsed on the browser.

CSS BEST PRACTICES

- CONVERT PX TO REM - AN EFFECTIVE WORKFLOW
- WRITE MAINTAINABLE AND SCALEABLE CSS WITH BEM

CONVERT PIXELS TO REM - AN EFFECTIVE WORKFLOW

Beginner CSS developers often use the pixel unit everywhere and that's fine there is nothing wrong with it.

But in this lesson, I will try to convince you to use **rem instead of px**. You will see what are the benefits of converting **px to rem**.

Working with pixels

Let's say you built a webpage where you used pixels for everything from font size to padding, margin, and all distances. Now to make this page responsive you will define media queries for all those sizes.

- Extra Large Devices: **Above 1200px**
- Large Devices: **Above 992px**
- Medium Devices: **Above 768px**
- Small Devices: **Above 600px**
- Extra Small Devices: **Below 600px**

For Instance:

CSS

```css
body {
  padding: 30px;
}
.heading-primary {
  margin-bottom: 60px;
}
.text-primary {
  font-size: 60px;
  letter-spacing: 35px;
}
.text-secondary {
  font-size: 20px;
  letter-spacing: 17px;
}
.btn {
  padding: 15px 40px;
  border-radius: 100px;
  font-size: 16px;
}
```

Media Query

```css
@media all and (max-width: 600px)
{
  body {
    padding: 15px;
  }

  .heading-primary {
    margin-bottom: 30px;
  }

  .text-primary {
    font-size: 30px;
    letter-spacing: 17px;
  }

  .text-secondary {
    font-size: 15px;
    letter-spacing: 8px;
  }

  .btn {
    padding: 7px 20px;
    border-radius: 100px;
    font-size: 10px;
  }
}
```

We wrote a media query for each font size, padding, margin, and distance. Well, that's fine, But I can show you a better way to make all size and length-related stuff adaptable and responsive with just one line of the media query.

Working with rem

The rem unit is always related to the **root font size**. And the root font size is set in the HTML selector.

For example, the default root font size of most browser is 16px. So, 1rem is 16px, 2rem is 32px, 3rem is 48px, and so on and so forth.

So if we now set that root font size, we can and very easily change all the other measurements on our page to rem.

To keep our px to rem calculations easy, I will set the root font size to 10px. So, now 10px is 1rem, 20px is 2rem, 35px is 3.5rem.

rem = px / 10

Let's change all px units to rem.

Now to make the page responsive you don't have to define media queries for all size and length-related stuff. Just define the root font size in the media query and boom, you have an adaptable and responsive page out of the box.

CSS with all unit set to rem

```css
html {
  font-size: 10px;
}

body {
  padding: 3rem;
}

.heading-primary {
  margin-bottom: 6rem;
}

.text-primary {
  font-size: 6rem;
  letter-spacing: 3.5rem;
}

.text-secondary {
  font-size: 2rem;
  letter-spacing: 1.7rem;
}

.btn {
  padding: 1.5rem 4rem;
  border-radius: 10rem;
  font-size: 1.6rem;
}
```

Media Query

```css
@media all and (max-width: 600px) {
  html {
    font-size: 6px;
  }
}
```

Compare this media query with the previous one.

That's it. All sizes and lengths will reduce accordingly below 600px because all your sizes and lengths depend on the root font size.

DEFINE ROOT FONT SIZE IN %

It's a bad practice to define the root font size in px. You should always define the root font size in %. Let me explain why.

Sometimes readers (users) might change the default font size of the browser for his / her reading convenience. So, if the reader increases the default font size to 20px, it won't change the root font-size defined by you which is 10px.

That's why you should define the root font size in percentage. So, that it changes when the reader changes the browser's default font size.

So, the root font size of 100% will be the browser's default font size of 16px. But you want it to be 10px (to make your calculation easier). Let's do the math.

16px = 100%
*(16px / 16px) * 10px = (100 / 16) * 10*
10px = 62.5%

All right then, Let's change the root font size from 10px to 62.5%.

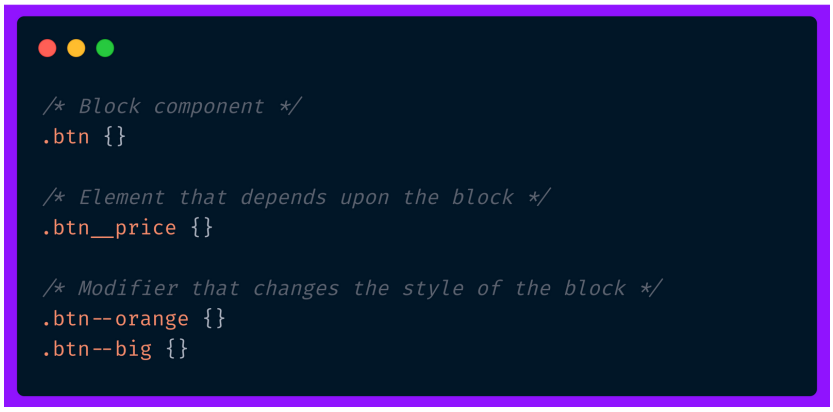
```
html {
  font-size: 62.5%;
  /* Default 100% is 16px. All rem will adapt accordingly
if user changes browser font size. */
}
```

WRITE MAINTAINABLE AND SCALEABLE CSS WITH BEM

The Block, Element, Modifier methodology (commonly referred to as BEM) is a popular naming convention for classes in HTML and CSS.

It helps developers better understand the relationship between HTML and CSS in a given project.

Here's an example of what a CSS developer writing in the BEM style might write:



```
/* Block component */
.btn {}

/* Element that depends upon the block */
.btn__price {}

/* Modifier that changes the style of the block */
.btn--orange {}
.btn--big {}
```

BEM (Block Element Modifier)

- Block: Standalone component that is meaningful on its own. (Parent Component)
- Element: Part of a block that has no standalone meaning. An element of the block component (Child Component).
- Modifier: A different version of a block or element. A modifier that changes the style of the block (Different version of Parent Component)

In the above code snippet:
btn class is the block.
btn__price is the element. price has no standalone meaning, it depends on the block. (double underscore is used for element class.)
btn--orange is the modifier. It modifies the block's color to orange. A different version of the block. (double dash is used for modifier class.)

BEM helps us get rid of the nested class names.

Without BEM: Nested Class Names

```
<head>
  <style>
    .profile { ... }
    .profile .heading { ... }
    .profile .img { ... }
    .profile .link { ... }
    .profile .link .btn { ... }
  </style>
</head>
<body>
  <section class="profile">
    <h1 class="heading">Name: Swastik</h1>
    <img src="/image-path" class="img" />
    <a href="/path" class="link">
      <button class="btn">Profile Link</button>
    </a>
  </section>
</body>
```

With BEM: No Nesting of Class Names

```
<head>
  <style>
    .profile { ... }
    .profile__heading { ... }
    .profile__img { ... }
    .profile__link { ... }
    .profile__link__btn { ... }
    .profile--bg { ... } /*Change profile background color.*/
  </style>
</head>
<body>
  <section class="profile profile--bg">
    <h1 class="profile__heading">Name: Swastik</h1>
    <img src="/image-path" class="profile__img" />
    <a href="/path" class="profile__link">
      <button class="profile__link__btn">Profile Link</button>
    </a>
  </section>
</body>
```

To wrap things up I think it's fair to say that even though BEM won't solve all our problems it is extraordinarily useful for constructing scalable and maintainable interfaces where everyone on the team needs agreements, promises, and binding social contracts between developers so that our codebase can adapt over time.

TECHNICAL SOPHISTICATION

HOW TO DO RESEARCH AND FIGURE OUT SOLUTIONS ON YOUR OWN.

TECHNICAL SOPHISTICATION

As learnenough.com describes Technical Sophistication:

The ability to figure out technical problems on your own. Technical sophistication includes concrete skills like command lines, text editors, and coding, as well as fuzzier skills like Googling the error message and knowing when to just reboot the darn thing.

As a developer, you may (you will surely) stuck somewhere and don't know how to proceed further. That's where your research skills come in.

Programming is as much about reading as it is about writing the code. I also believe in the philosophy of **Learn as you go.**

- Google about your problem, open all ten results in new tabs and read all of them thoroughly one by one.
- Found a promising solution? tweak it as per your needs and implement it.

I use Google every day while programming, Developers with years of experience do the same. So, there is nothing wrong with it as long as you get the job done with it.

No book could ever teach you everything about CSS. You get the basics from this book, **now go build epic shit** and **learn as you go**.

DON'T HESITATE TO SEEK HELP

Mastering the skill of technical sophistication takes time. Just keep doing it and you will keep getting good at it. Ask for help when you just can't figure out things.

Beginners often hesitate to ask for help, but you shouldn't every experienced and senior developer has been there. So, they are more than happy to help.

I am always ready to help. Send me your doubt at **swastiky28@gmail.com** and I will get in touch with you ASAP.

PROJECTS

A REAL WORLD RESPONSIVE WEBPAGE PROJECT

WORK IN PROGRESS!

I am still working on the 3 responsive real-world CSS projects.

I want it to be the best project-based lecture out there. That's why I am not rushing through it. I am taking my time.

The lecture will be ready soon.

Benefits

- **3 Responsive CSS projects video lecture**
- **Exactly how it is done in the real world - No dummy sites**
- **We will follow all the best practices we discussed in this book.**
- **This will teach you how to get things done with CSS.**

You will get the video course access for free as I promised that with this E-Book

For now, watch me build a completely responsive web page from scratch in the below two videos.

Invoice Web Page Project:

part 1: https://youtu.be/PHHFCh96IO4
part 2: https://youtu.be/0RmeXmBHPho

THANK YOU!