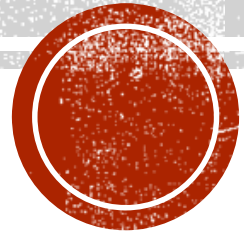
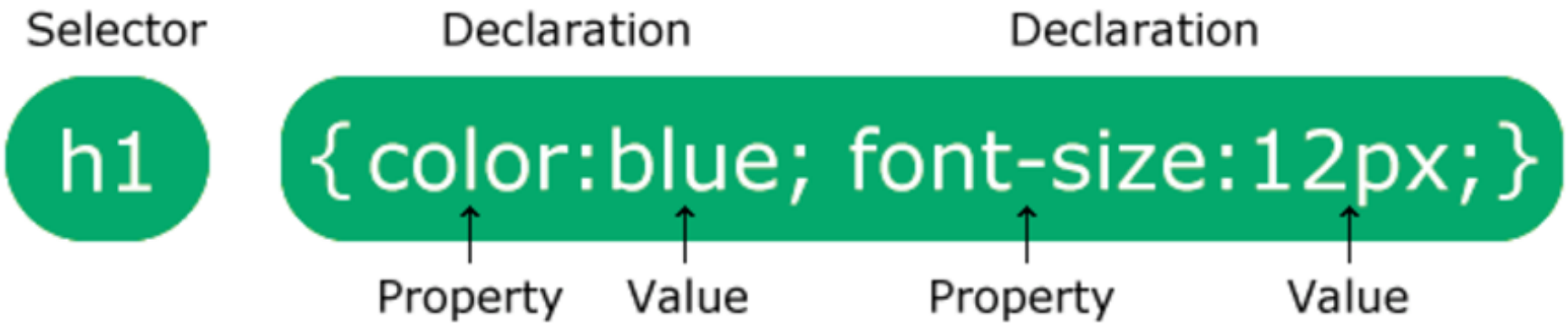
CASCADING STYLE SHEET



CSS Syntax



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

```html
<!DOCTYPE html>
<html>
<head>
<style>
p {
  color: red;
  text-align: center;
}
</style>
</head>
<body>

<p>Hello World!</p>
<p>These paragraphs are styled with CSS.</p>

</body>
</html>
```

Hello World!

These paragraphs are styled with CSS.

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

The CSS element Selector

The element selector selects HTML elements based on the element name.

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<p>Every paragraph will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```

Every paragraph will be affected by the style.

Me too!

And me!

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```

Hello World!

This paragraph is not affected by the style.

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.
To select elements with a specific class, write a period (.) character, followed by the class name.

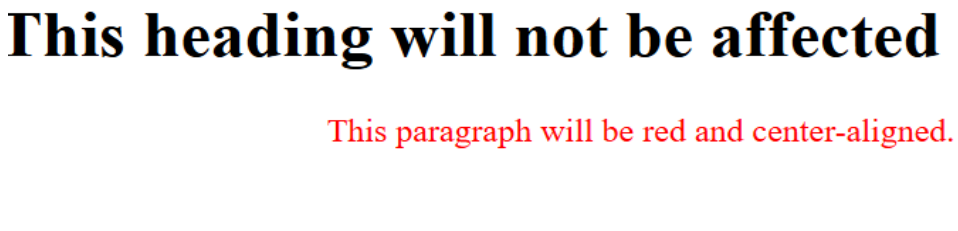
```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>

</body>
</html>
```

Red and center-aligned heading

Red and center-aligned paragraph.

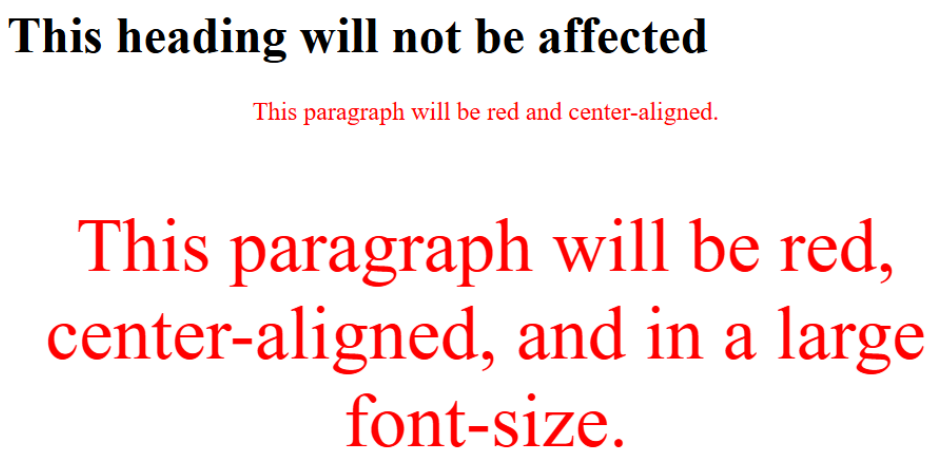
You can also specify that only specific HTML elements should be affected by a class.

```
<!DOCTYPE html>
<html>
<head>
<style>
p.center {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1 class="center">This heading will not be affected</h1>
<p class="center">This paragraph will be red and center-aligned.</p>

</body>
</html>
```



HTML elements can also refer to more than one class.

```
<!DOCTYPE html>
<html>
<head>
<style>
p.center {
  text-align: center;
  color: red;
}

p.large {
  font-size: 300%;
}
</style>
</head>
<body>

<h1 class="center">This heading will not be affected</h1>
<p class="center">This paragraph will be red and center-aligned.</p>
<p class="center large">This paragraph will be red, center-aligned, and
in a large font-size.</p>

</body>
```



The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>

<h1>Hello world!</h1>

<p>Every element on the page will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.
Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
<!DOCTYPE html>
<html>
<head>
<style>
h1, h2, p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1>Hello World!</h1>
<h2>Smaller heading!</h2>
<p>This is a paragraph.</p>

</body>
</html>
```

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers.

A CSS comment is placed inside the `<style>` element, and starts with `/*` and ends with `*/`:

```
<!DOCTYPE html>
<html>
<head>
<style>
/* This is a single-line comment */
p {
  color: red;
}
</style>
</head>
<body>

<p>Hello World!</p>
<p>This paragraph is styled with CSS.</p>
<p>CSS comments are not shown in the output.</p>

</body>
</html>
```

Hello World!

This paragraph is styled with CSS.

CSS comments are not shown in the output.

CSS SPECIFICITY

- What is Specificity?
- If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element.
- Example 1
- In this example, we have used the "p" element as selector, and specified a red color for this element. The text will be red:

```
<html>
<head>
  <style>
    p {color: red;}
  </style>
</head>
<body>

<p>Hello World!</p>

</body>
</html>
```

- Example 2
- In this example, we have added a class selector (named "test"), and specified a green color for this class. The text will now be green (even though we have specified a red color for the element selector "p"). This is because the class selector is given higher priority:

```
<html>
<head>
  <style>
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p class="test">Hello World!</p>

</body>
</html>
```

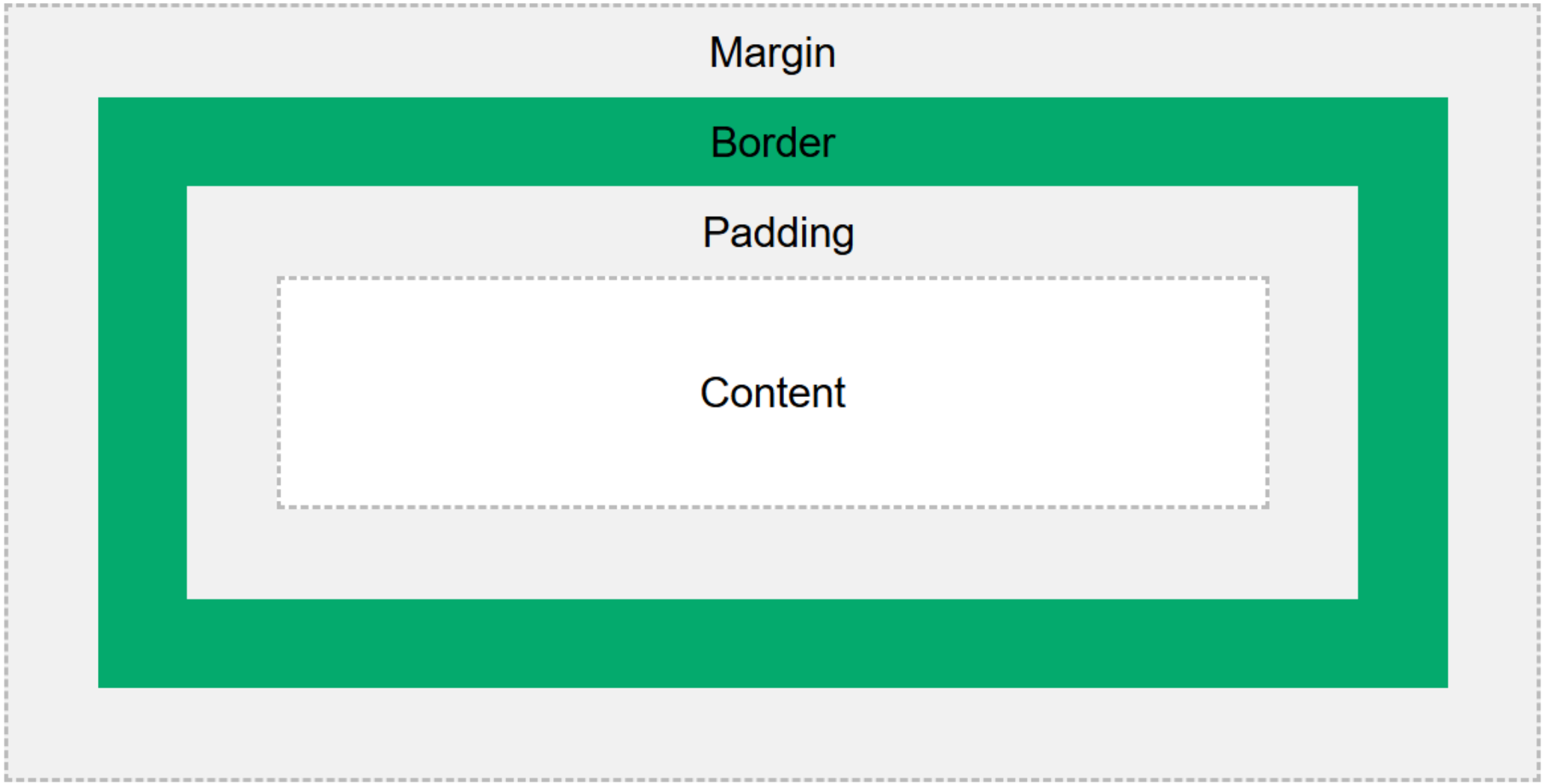
- Example 3
- In this example, we have added the id selector (named "demo"). The text will now be blue, because the id selector is given higher priority:

```
<html>
<head>
  <style>
    #demo {color: blue;}
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p id="demo" class="test">Hello World!</p>

</body>
</html>
```

- Specificity Hierarchy
- Every CSS selector has its place in the specificity hierarchy.
- There are four categories which define the specificity level of a selector:
- **Inline styles** - Example: <h1 style="color: pink;">
- **IDs** - Example: #navbar
- **Classes, pseudo-classes, attribute selectors** - Example: .test, :hover, [href]
- **Elements and pseudo-elements** - Example: h1, ::before

THE CSS BOX MODEL

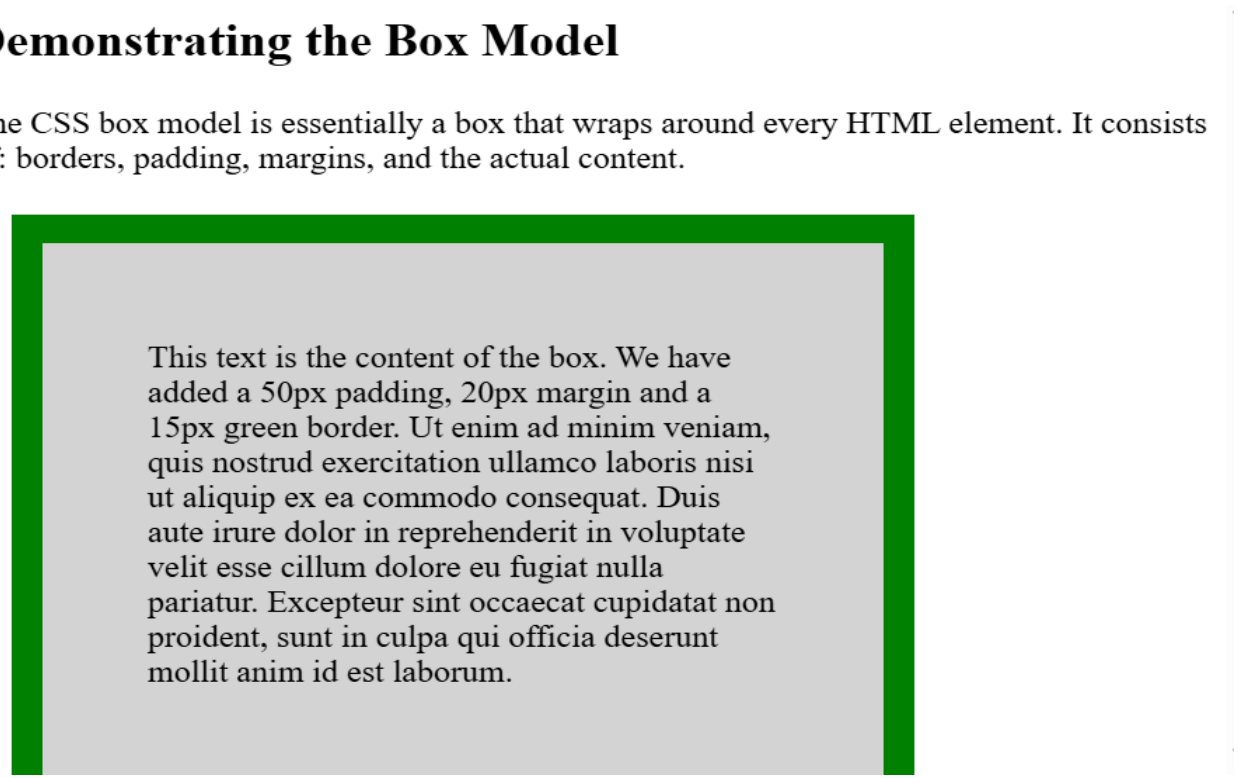
- **What is the CSS Box Model?**
- The CSS Box Model is a layout model that describes how different components of a web element (**content, padding, border, and margin**) are structured and positioned. Each web element generates a rectangular box that encompasses these components, and the Box Model allows developers to control the element's size and spacing effectively.
- **Key Components of the Box Model**
- The CSS Box Model consists of four primary components:


Margin
Border
Padding
Content

- **1. Content Area**
- The content area is where the actual content, such as text, images, or other media, is displayed.
- It is sized using the width and height properties.
- The boundary of the content area is known as the content edge.
- **2. Padding Area**
- The padding surrounds the content area and creates space inside the border.
- It can be adjusted using the padding property (or padding-top, padding-right, padding-bottom, and padding-left for individual sides).
- The padding area increases the overall size of the element without changing the content area.

- **3. Border Area**
- The border wraps around the padding and content, defining the edge of the element.
- It can be styled using properties such as border width border **color, border style, and border-color**.
- The width of the border affects the overall size of the element.
- **4. Margin Area**
- The margin is the outermost space that separates the element from adjacent elements.
- It can be set using the margin property (or margin-top, margin-right, margin-bottom, and margin-left for individual sides).
- Unlike padding and bordborderse margin does not increase the element's total size but affects its placement on the page.

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  background-color: lightgrey;
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
</style>
</head>
<body>

<h2>Demonstrating the Box Model</h2>

<div>This text is the content of the box. We have added a 50px padding, 20px margin and a 15px green border. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</div>

</body>
</html>
```



FLEXBOX AND GRID LAYOUT

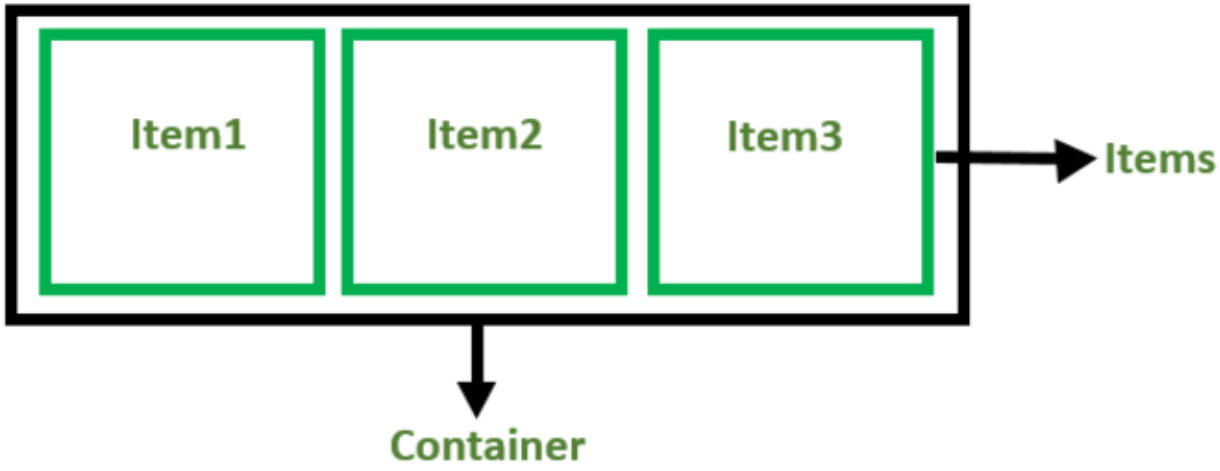
- **CSS Grid Layout**
- CSS Grid Layout, is a two-dimensional grid-based layout system with rows and columns. It makes easier to design web pages without having to use floats and positioning. Like tables, grid layout allow us to align elements into columns and rows.
- To get started, you have to define a container element as a grid with **display: grid**, set the column and row sizes with grid-template-columns and grid-template-rows, and then place its child elements into the grid with grid-column and grid-row.

- **This can be used as a shorthand property for :**
- **grid-template-rows :** Specifies the size of the rows.
- **grid-template-columns :** This specifies the size of the columns.
- **grid-template-areas :** This specifies the grid layout using named items.
- **grid-auto-rows :** This specifies the auto size of the rows.
- **grid-auto-columns :** This specifies the auto size of the columns.
- **grid-auto-flow :** This specifies how to place auto-placed items, and the auto size of the row.

- **Example 2:** This is example for the **grid-template-rows** and **grid-template-columns**.

- **CSS Flexbox**
- The CSS Flexbox offers one-dimensional layout. It is helpful in allocating and aligning the space among items in a container (made of grids). It works with all kinds of display devices and screen sizes. To get started you have to define a container element as a grid with **display: flex;**
- **Features of Flexbox:**
- High flexibility
- Arrangement & alignment of items
- Proper spacing
- Order & sequencing of items
- Bootstrap 4 is built on top of the flex layout

- **Flexbox Components:**
- **Flex Container:** The parent div containing various divisions.
- **Flex Items:** The items inside the container div.



- **Creating a Flexbox:**
- To create a flexbox, set the display property of the container to flex.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Flexbox Tutorial</title>
    <style>
    .flex-container {
        display: flex;
        background-color: #32a852;
    }

    .flex-container div {
        background-color: #c9d1cb;
        margin: 10px;
        padding: 10px;
    }
    </style>
</head>
<body>
    <h2>Welcome </h2>
    <h4> Flexbox</h4>
    <div class="flex-container">
        <div>Item1</div>
        <div>Item2</div>
        <div>Item3</div>
    </div>
</body>
</html>
```

Welcome

Flexbox

Item1 Item2 Item3

- **Aligning and Justifying Content:**
- Flexbox provides several properties for aligning and justifying content within the container:
- **justify-content:** Aligns items along the main axis (e.g., flex-start, flex-end, center, space-between, space-around).
- **align-items:** Aligns items along the cross axis (e.g., stretch, center, flex-start, flex-end).
- **align-content:** Aligns rows within a flex container when there is extra space on the cross axis.

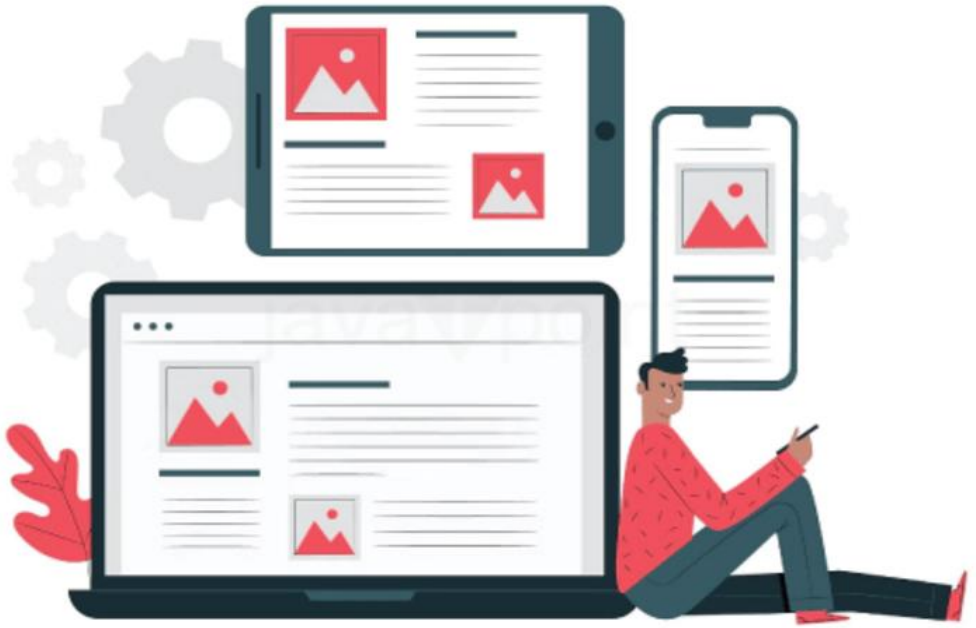
- **Responsive Design with Flexbox:**
- Flexbox excels in creating responsive designs by adjusting items to fit various screen sizes. You can use media queries in combination with Flexbox properties to ensure your layout adapts seamlessly to different devices.
- **Example of Responsive Flexbox:**

Difference Between CSS Grid and Flexbox

Property	Grid	Flexbox
Dimension	Two – Dimensional	One – Dimensional
Features	Can flex combination of items through space-occupying Features	Can push content element to extreme alignment
Support Type	Layout First	Content First
Primary Use Case	Creating complex layouts with rows and columns	Aligning items in a row or column
Performance	Can be less in performance due to very complex grids	Generally faster for simple layouts

WHAT IS CSS RESPONSIVE WEB DESIGN?

- In CSS, making a responsive webpage means that it is a method used to design and code the webpage to respond to all types of screen sizes. It also provides the best viewing express for the user, and it does not depend on the screen size of the mobile; it can fit any size of the mobile phone.



WHY IS RESPONSIVE DESIGN SO IMPORTANT?

- Creating a responsive webpage is very important. These are as follows.
- **1. Device-Independent Rendering**
- It is the main important advantage of creating a responsive webpage. Nowadays, we use so many types of devices with different screen sizes. Also, there is no uniform standard of size, shape, or display parameters across devices. At this point, responsive design comes into work. After creating the responsive webpage, it makes the devices optimal in the particular environment.

- **2. Usage Experience**
- We can increase the user experience by creating a responsive webpage. It is also a key factor that increases the number of web page visitors. If the user finds it very difficult to navigate the webpage, then the user never returns to the website again. We can increase the user experience on the web page by creating a responsive web page.

- Key principles and techniques of CSS Responsive Web Design
- There are some Key principles and techniques of CSS Responsive Web Design. These are as follows.
- **Fluid Grids:** We have to design the web page layout using relative units rather than fixed units to provide a responsive web page design.
- **Flexible Images:** With the help of CSS, we can ensure that we can fit the image inside the container without creating layout issues.
- **Media Queries:** We can apply the CSS rule by providing specific conditions, such as screen width, orientation, and device characteristics, to modify the layout and styling of a webpage.

- **Breakpoints:** We can define the particular width of the screen, which occurs when the layout of the webpage should be changed. We can also design to optimize the different sets of screen sizes.
- **Mobile-First Approach:** First, we have to design the mobile version of the webpage. Then, we have to change the website to fit the web page. It will make a strong foundation for mobile devices.
- **Viewport Meta Tag:** We can control the display of the web page by using the viewport of the meta tag in the HTML. It will also ensure the prevention of unwanted scaling.
- **CSS Flexbox and Grid Layout:** We can create modern CSS layout techniques like grid or flexbox, which are used to create more complex and flexible responsive layouts.

A MOBILE-FIRST APPROACH TO RESPONSIVE WEB DESIGN

- The meaning of the mobile-first approach is to design the first website for the mobile, and then we have to create the desktop version of the web page. There is a variety of reasons which is used to create an effective website.
- It is a very complex thing to create a mobile website. So we have to focus on the design of the mobile.
- It is also very easy to scale the mobile version, which is used to scale down the desktop version.
-

SETTING THE VIEWPORT

- We have to write the viewport tag inside the meta tag. It is used to optimize the variety of the devices.
- **Syntax:**
- **<**meta name="viewport" content="width=device-width, initial-scale=1.0"**>**

EXAMPLE 1

- **Explanation:**
- In the above example, there is a meta tag named "viewport," which ensures that the webpage should ensure that it adjusts its width to match the device's screen width. It also defines the responsive container with a maximum width of 1200px and centers it on the page.

```html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
 <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
    .container {
      width: 90%;
      max-width: 1200px;
      margin: 0 auto;
      padding: 20px;
      background-color: #f0f0f0;
    }
  </style>
</head>
<body>
 <div class="container">
    <h1>Welcome to CCT</h1>
    <p>This is a simple example of responsive web design using CSS and the viewport meta tag.</p>
  </div>
</body>
</html>
```

- Responsive Images
- In responsive images, the image is scaled to fit the size of that fit the screen size. When there is a need to create a responsive image, we have to implement some technique that allows them to scale, resize, or even change based on the characteristics of the user's device.
- How to Make Image Responsive?
- We can create a responsive image with the help of some techniques. They are as follows.
- 1. By using the Width Property
- We can create a responsive image with the help of the width property.
- **Example 2:**

- In the above example, there is a class named ".image-container," which provides the implementation maximum width of the container to 100%, also ensuring that the container adjusts to the width of its parent element. Also, we have set the height property as auto and the width property as width: 100%.

- 2. Using the Max-Width Property
- We can create a responsive webpage with the help of the max-width property. The main advantage of using the max-width property is that we can not enlarge the image more than its original size.
- **Example 3:**
- **Explanation:**
- In the above example, we have created an image in the center position with a maximum width of 800px and a light gray background. Then we have to center the img container, which has a maximum width of 100%. Then, we have to provide the height property as auto. Then, we have to provide the display property as a block. Then, we have to set the margin as 0 auto.

- 3. Responsive Text Size
- When there is a need to create the text size as responsive text, we have to ensure that the content should be legible and user-friendly for different devices. Let's see this with the help of the below code.
- **Example 4:**
- **Explanation:**
- In the above example, we have created text in the center position of the webpage, which has a width of 800px. It has the background of a light gray color. We have set the font side of the web page in the format of a "vw" unit. Then, we have to set the line height property, which increases the web page's readability. Then, we have to set the margin property as 0 auto.

WHAT IS A MEDIA QUERY?

- It is a technique present in CSS used to apply different types of styles to the webpage. It is also the main fundamental part of creating a responsive web page. We can also create the design and layout of the webpage, which is used in creating the web page. We can use media query with the help of @media, which specifies the condition for creating the web page.
- **Syntax:**
- To create the media query, we have to follow the syntax below.
- @media media-type and (media-feature) {
- /* CSS rules to apply when the media query conditions are met */
- }

- The @media rule is used to target media queries and then the media type which is optional depending on the device type and then the condition also called a breakpoint, if the condition is true then the CSS rules are applied to the web page from the parenthesis.

- **Media Types**
- Media Types describe the category of device type. Media type is optional and by default, it is considered as `all` type. Media Types are categorized into four types.
- all - for all device types.
- print - for printers
- screen—for smartphones, tablets, TVs and computer screens
- speech—for screen readers that "read" the page out loud

- **Why to use Media Queries?**
- Media Queries helps the developers to create a consistent and optimized user experience across all devices. Without responsive design, websites may appear unstable and broken on certain screens which results in poor user experience. With a better user experience, it will be easy to use the product.

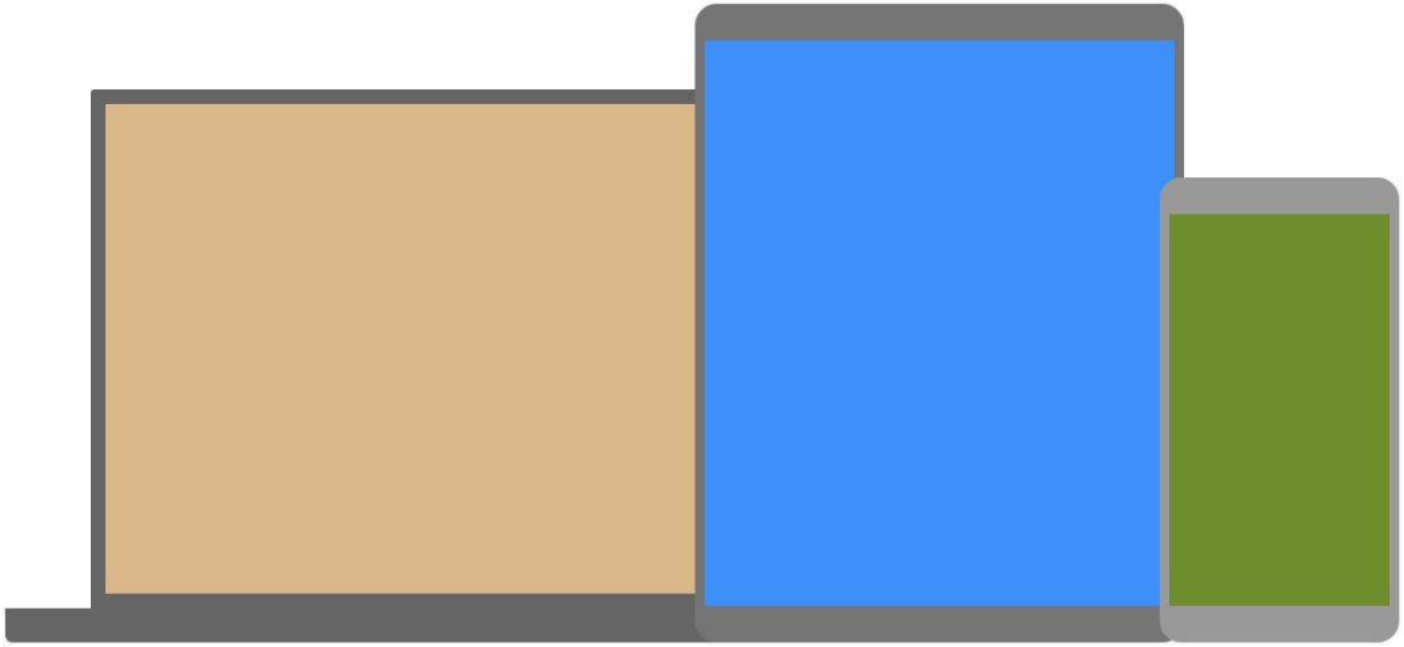
BENEFITS OF RESPONSIVE DESIGN

- **Improves User Experience(UX):** Users can access content seamlessly and efficiently on any device. Whether they are on a desktop computer, tablet, or smartphone, the website adapts to provide an optimal and user-friendly experience. This leads to higher user satisfaction and engagement.
- **Good SEO:** When your website is responsive and provides a consistent experience across devices, it's more likely to rank higher in search engine results. Responsive designs rank higher and increases traffic.
- **Cost-Efficiency:** Developing a single website for all platforms, reduces development and maintenance costs. Maintenance and updates are also simplified when you have one codebase to manage.
- **Increases Engagements**: Users are more likely to stay and interact with a well-designed responsive site. A well-designed responsive site enhances user engagement. When visitors have a positive experience and can easily navigate and interact with your content on their preferred devices, they are more likely to stay on your site.

- **Elements of a Responsive Design**
- **Media Queries:** Media queries are the foundation of responsive design. They allow you to specify conditions based on characteristics like screen width, height, orientation, and resolution. Declared in CSS using @media followed by conditions. you can apply specific styles to different devices or screen sizes.
- **Breakpoints:** Specific screen widths where design changes occur. Breakpoints are specific screen widths (or occasionally heights) at which you make design adjustments. These adjustments can include changes in layout, typography, or the visibility of certain elements.
- **Flexible Layouts:** Use percentage-based or fluid grids and Avoid fixed width and height instead use min-width, max-width, min-height and max-height. CSS Flexbox and CSS Grid are powerful tools for creating flexible layouts.
- **Images**: Scaling images using CSS or using responsive image techniques. Use CSS to set max-width: 100% for images, preventing them from exceeding their container width.
- **Font Sizes:** Adjusting font sizes and better font hierarchy for readability on different screens. Font sizes play a vital role in readability and user experience. Consider using relative units like em or rem instead of fixed pixel values for font sizes.

- **How Does Responsive Design Work?**
- **Define Breakpoints:** Identify key screen widths for design changes mostly mobile (320px—480px), tablets (481px—768px), laptops (769px—1024px), and desktops(1025px—1200px) these breakpoints are industry standard. But you can customize them based on your project's requirements.
- **Write Media Queries:** Use CSS @media to apply styles at breakpoints. They are CSS rules that apply specific styles when certain conditions are met. These conditions are typically based on screen width using the min-width and max-width properties
- **Adjust Layouts**: To create responsive layouts, use CSS features like Flexbox and CSS Grid. These layout techniques provide a flexible and fluid structure that adapts to different screen sizes.
- **Test on Various Devices:** Ensure the design works well across different screens using dev tools. Pay attention to user interactions, load times, and accessibility to ensure a seamless experience across all devices.
- **Refine as Needed:** Continuously optimize and adjust for a seamless user experience. Responsive design is an ongoing process. After testing, gather feedback, and make necessary adjustments to improve the user experience further.

- Let us look at some more examples of using media queries.
- Media queries are a popular technique for delivering a tailored style sheet to different devices. To demonstrate a simple example, we can change the background color for different devices:



```css
/* Set the background color of body to tan */
body {
  background-color: tan;
}

/* On screens that are 992px or less, set the background color to blue */
@media screen and (max-width: 992px) {
  body {
    background-color: blue;
  }
}

/* On screens that are 600px or less, set the background color to olive */
@media screen and (max-width: 600px) {
  body {
    background-color: olive;
  }
}
```

- Media Queries For Menus
- In this example, we use media queries to create a responsive navigation menu, that varies in design on different screen sizes.

Large screens:

Small screens:

```css
/* The navbar container */
.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Navbar links */
.topnav a {
  float: left;
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* On screens that are 600px wide or less, make the menu links stack on top of each other
instead of next to each other */
@media screen and (max-width: 600px) {
  .topnav a {
    float: none;
    width: 100%;
  }
}
```

WHAT IS A CSS PREPROCESSOR?

- CSS preprocessors are scripting languages that extend the default capabilities of CSS. They enable us to use logic in our CSS code, such as variables, nesting, inheritance, mixins, functions, and mathematical operations. CSS preprocessors make it easy to automate repetitive tasks, reduce the number of errors and code bloat, create reusable code snippets, and ensure backward compatibility.
- Each CSS preprocessor has its own syntax that they compile into regular CSS so that browsers can render it on the client side.

- **1. Sass: “Syntactically Awesome Style Sheets”**
- Sass is the most popular and oldest CSS preprocessor, initially released in 2006. Its creators, Natalie Weizenbaum and Hampton Catlin.
- The Sass preprocessor allows us to use variables, if/else statements, for/while/each loops, inheritance, operators, interpolation, mixins, and other dynamic features, then compile the code to plain CSS that web browsers can interpret.

- There are two types of syntax available for SASS:
- SCSS(Sassy CSS): The files using this syntax use *.scss* extension.
- Indented syntax (referred to as just "sass"): older syntax, Files using this syntax use *.sass* extension.

- **Working Steps:**
- Write the SCSS code.
- Compile the SCSS code into CSS code using the command *sass input.scss output.css*. The first filename (input.scss) is the scss file that is to be compiled and the second file name (output.css) is the processed CSS file, to be included/attached in the Html document.
- Include the compiled CSS file in the Html file.
- Now see how to make effective use of the important features of SCSS like variables, nesting, mixins, and operators.
- The main HTML file is named *index.html*
- SCSS file is *styling.scss* and the CSS file is *style.css*
- Command to compile the SCSS file: *sass styling.scss style.css*

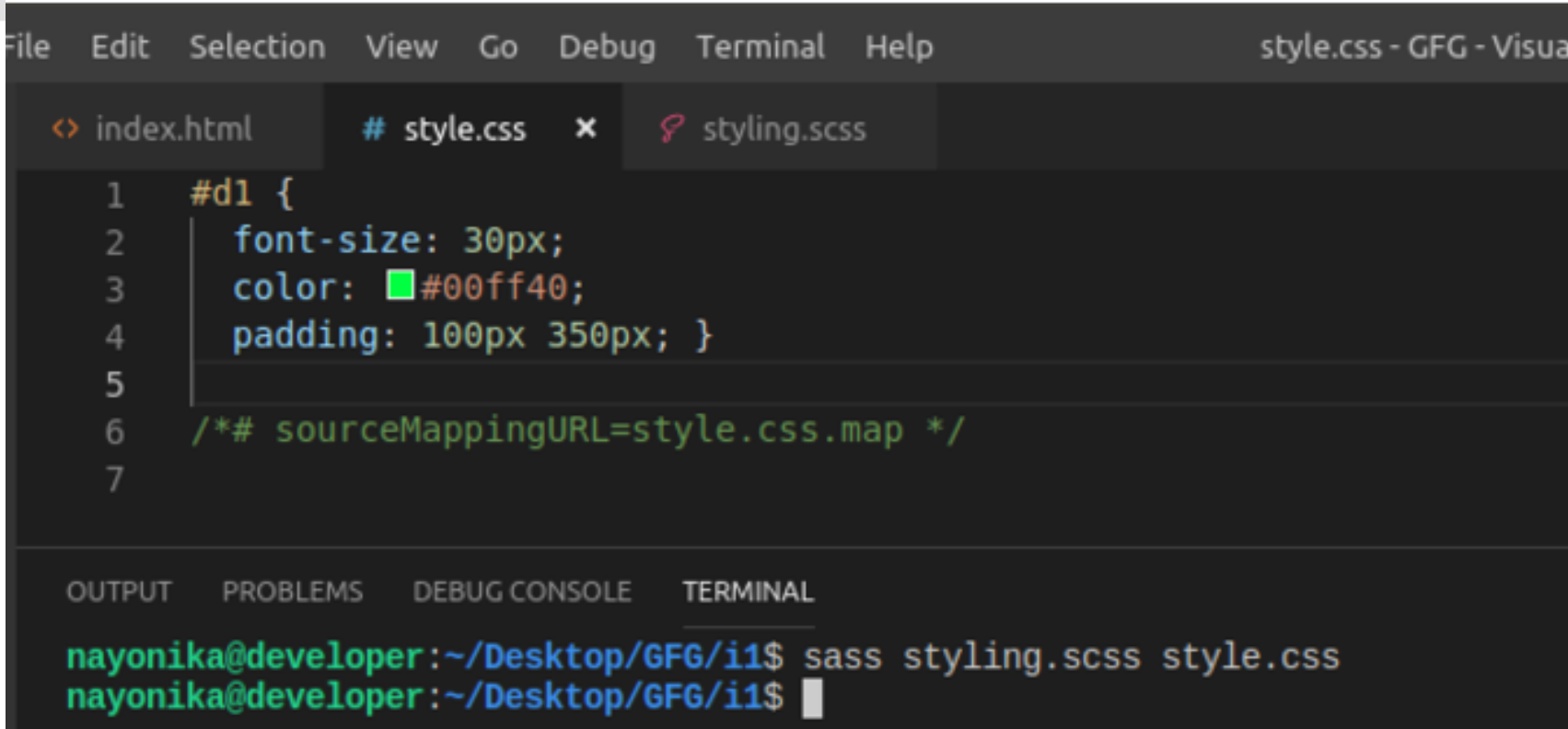
```html
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title>SASS</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div id="d1">Welcome to GeeksforGeeks.
        <ul>
            <li>Algo</li>
            <li>DS</li>
            <li>Languages</li>
            <li>Interviews</li>
            <li>CS subjects</li>
        </ul>
    </div>
</body>
```

Variables: Variables can be used to store CSS values that may be reused. To declare a variable in SASS, the '$' character is used. For eg, $v_name.

```
$fs: 30px;
$bgcolor: #00ff40;
$pd: 100px 350px;
#dl {
 font-size: $fs;
 color: $bgcolor;
 padding: $pd;
}
```

After compiling the CSS code, save it in file by *style.css.*

```
#dl {
 font-size: 30px;
 color: #00ff40;
 padding: 100px 350px;
}
```



- **Nesting**: SASS allows CSS rules to be nested within each other, which follows the same visual hierarchy of HTML. For eg. CSS property can be used to the <li> tag nested inside the div tag.

```
$fs: 30px;
$bgcolor: #00ff40;
#col2: #ff0066e1;
$pd: 100px 350px;
#dl {
 font-size: $fs;
 color: $bgcolor;
 padding: $pd;
 li {
   color: $col2;
 }
```

index.html | styling.scss | style.css

```scss
$fs: 30px;
$bgcolor: #00ff40;
$col2: #ff0066e1;
$pd:100px 350px;
#d1{
    font-size: $fs;
    color: $bgcolor;
    padding: $pd;
    li{
        color:$col2;
    }
}
```

After compiling the CSS code save it file by *style.css.*

```
#dl {
 font-size: 30px;
 color: #00ff40;
 padding: 100px 350px;
}
#dl li {
 color: #ff0066e1;
}
```

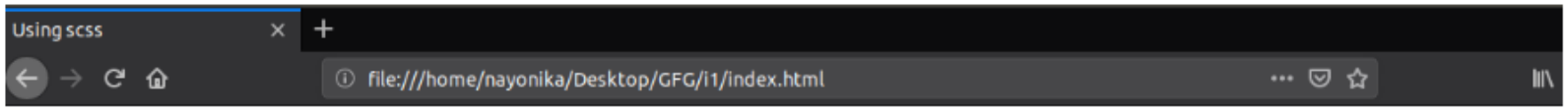
index.html | styling.scss | # style.css ×

```
1  #dl {
2    font-size: 30px;
3    color: #00ff40;
4    padding: 100px 350px; }
5    #dl li {
6      color: #ff0066e1; }
7
8  /*# sourceMappingURL=style.css.map */
9
```

OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL

```
nayonika@developer:~/Desktop/GFG/i1$ sass styling.scss style.css
nayonika@developer:~/Desktop/GFG/i1$
```

Output:



Welcome to GeeksforGeeks.

1. Algo
2. DS
3. Languages
4. Interviews
5. CS subjects

- **Benefits**
- well-established, stable CSS preprocessor
- compilation has become easier with the introduction of Dart Sass
- has a large ecosystem
- advanced dynamic functionalities
- you can choose between two syntaxes
- extensive documentation

Mypage.html

```html
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="mystyle.css">
<body>

<h1>Hello World</h1>

<p>This is a paragraph.</p>

</body>
</html>
```

Mystyle.css

```css
body {
  background-color: lightblue;
  color: darkblue;
  font-size: 18px;
}
```

Mystyle.scss

```scss
/* Define standard variables and values for website */
$bgcolor: lightblue;
$textcolor: darkblue;
$fontsize: 18px;

/* Use the variables */
body {
  background-color: $bgcolor;
  color: $textcolor;
  font-size: $fontsize;
}
```

LESS: "LEANER STYLE SHEETS"

- LESS was released three years after Sass in 2009 by Alexis Sellier. It was influenced by Sass, and so implements many of its features such as mixins, variables, and nesting.
- **Benefits**
- it can be added directly to an HTML page
- flat learning curve as LESS uses the standard CSS syntax (see examples below)
- several dynamic features, including scoped variables, import options, mixins, nested selectors, and others
- extensible via plugins
- detailed documentation

- **LESS** stands for **Leaner Style Sheets.** It is a backward-compatible language extension for CSS. It allows us to use features like variables, nesting, mixins, etc, all in a CSS-compatible syntax. LESS is influenced by SASS and has influenced the newer "SCSS" syntax of SASS. LESS was used in Bootstrap 3 but was replaced by SASS in Bootstrap 4.
- **Pre-Requisites:**
- HTML
- CSS
- **System Requirements:**
- **Operating System:** Cross-platform
- **Browser Support:** IE (Internet Explorer 8+), Firefox, Google Chrome, Safari.

- **File Type:** All LESS files must have the .less file extension.
- **Working:** A web browser does not understand the LESS code itself. That is why you will require a LESS pre-processor to change LESS codes into simple standard CSS code.
- **Working Steps:**
- Write the LESS code in a file.
- Compile the LESS code into CSS code using the command ***lessc style.less style.css*.**
- Include the compiled CSS file in the html file.

style.less

CSS

```
@lt-gray: #ddd;
@background-dark: #512DA8;
@carousel-item-height: 300px;
h1 {
    color:@lt-gray;
    background:@background-dark;
}
```

Syntax: To compile the above less code to CSS code write the following command-

```
lessc style.less style.css
```

The compiled CSS file comes to be:

style.css

CSS

```css
h1 {
  color: #ddd;
  background: #512DA8;
}
```

- **Mixins:** Mixins are a way of including a bunch of properties from one rule-set into another rule set.
- **style.less**

CSS

```
zero-margin {
    margin:0px auto;
    background: white;
}

.row-header {
    margin:zero-margin;
    padding:0px auto;
}

.row-content {
    margin:zero-margin;
    border-bottom: 1px ridge;
    min-height:400px;
    padding: 50px 0px 50px 0px;
}
```

Syntax: To compile the above less code to CSS code write the following command-

```
lessc style.less style.css
```

style.css

CSS

```css
zero-margin {
  margin: 0px auto;
  background: white;
}
.row-header {
  margin: zero-margin;
  padding: 0px auto;
}
.row-content {
  margin: zero-margin;
  border-bottom: 1px ridge;
  min-height: 400px;
  padding: 50px 0px 50px 0px;
}
```

- **Advantages:**
- LESS is cross-browser compatible.
- LESS provides a list of operators making it easy for users to code.
- Maintenance is easy due to the use of variables.
- **Disadvantages:**
- LESS provides fewer frameworks as compared to SASS.
- It can be tricky to those who are new to CSS.