HASHIM PUNASIYA

Module 3 – Mernstack – CSS and CSS3

Question 1: What is a CSS selector? Provide examples of element, class, and ID selectors.

A **CSS selector** is a pattern used to select and style specific HTML elements in a web page. It tells the browser which elements to apply the specified CSS rules to.

Examples of CSS Selectors

➔ Element Selector

Targets all instances of a specific HTML element.

```
p{
    color: blue;
}
```

This example will style all `<p>` (paragraph) elements with blue text.

➔ Class Selector

Targets elements with a specific `class` attribute. Classes are reusable across multiple elements.

```
.highlight {
    background-color: yellow;
}
```

This example will style any element with the *class="highlight"* attribute by giving it a yellow background.

➔ ID Selector

Targets an element with a specific `id` attribute. IDs are unique and should be used only once per page.

```
#header {
    font-size: 24px;
}
```

This example will style the element with id="header" by setting its font size to 24px.

Example usage in HTML

<h1 id="header">This is the page header.</h1>

These selectors are the foundation of CSS and can be combined or used with more advanced techniques for precise styling.

Question : Explain the concept of CSS specificity. How do conflicts between multiple styles get resolved?

What is CSS Specificity?

CSS specificity is a set of rules that determines which CSS styles are applied when there are conflicting styles for the same element. It is essentially a ranking system that prioritizes some rules over others.

Each type of CSS selector has a different **specificity weight**. The higher the specificity, the more precedence a rule has.

Specificity Weighting

Specificity is calculated based on the following criteria (in order of importance):

1. **Inline styles** (added directly to an element using the `style` attribute) have the highest specificity.
   - Example: `<p style="color: red;">`
2. **ID selectors** (`#id`) are more specific than classes or element selectors.
   - Example: `#header`
3. **Class selectors**, **attribute selectors**, and **pseudo-classes** (e.g., `.class`, `[type="text"]`, `:hover`) have moderate specificity.
   - Example: `.highlight`
4. **Element selectors** and **pseudo-elements** (e.g., `p`, `h1`, `::before`) have the lowest specificity.
   - Example: `p`

Specificity Hierarchy

CSS calculates specificity using a point system:

- Inline styles: **1,0,0,0**
- ID selectors: **0,1,0,0**
- Class, attribute, and pseudo-class selectors: **0,0,1,0**
- Element and pseudo-element selectors: **0,0,0,1**

Resolving Conflicts

When multiple rules target the same element:

Higher specificity wins: A more specific selector overrides a less specific one.

```
#title {

   color: blue;
}

.highlight {

    color: red;

}

h1 {

    color: green;

}
```

For <h1 id="title" class="highlight">, the text color will be **blue** because #title has the highest specificity.

➔ **If specificity is the same**, the rule that appears later in the stylesheet (or is closer to the element in the cascade) takes precedence.

```
.highlight
 {
   color: red;
}

.highlight {
   color: green;
}
```

The text will be **green**, as the second .highlight rule is applied last.

➔ **Inline styles override all external stylesheets**:

<p id="text" style="color: yellow;">This text is yellow.</p>

Even if an external stylesheet sets #text { color: blue; }, the text will remain yellow due to the inline style.

➔ **Use of !important :**

A rule with !important will override all other rules, regardless of specificity, except another !important rule with higher specificity.

```
p {
   color: green !important;
}

.highlight {
   color: blue;
}
```

The paragraph will be **green**, as !important overrides everything else.

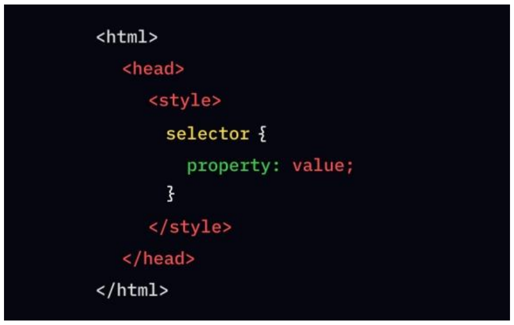
Question What is the difference between internal, external, and inline CSS?

CSS can be applied to HTML using three main methods: **internal CSS**, **external CSS**, and **inline CSS**. Each method has its own use case and characteristics.

1. Internal CSS

Internal CSS is defined within a `<style>` tag in the `<head>` section of an HTML document. It is used when styles are only needed for a single page.

Example



```
<html>
  <head>
    <style>
      selector {
        property: value;
      }
    </style>
  </head>
</html>
```

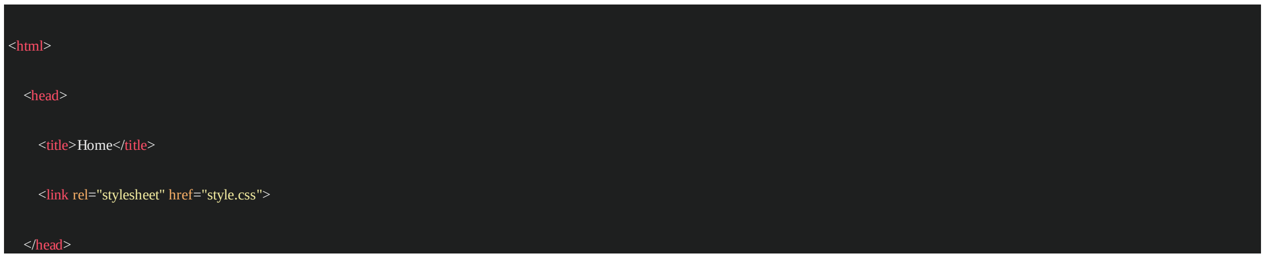
Characteristics:

- **Scope**: Styles are applied only to the specific HTML file in which they are defined.
- **Advantages**:
  - Keeps styles separate from the content structure.
  - Useful for quick, single-page styling.
- **Disadvantages**:
  - Not reusable across multiple pages.
  - Can clutter the HTML file if many styles are included.

2. External CSS

External CSS is written in a separate `.css` file and linked to the HTML document using the `<link>` tag. It is the most commonly used method for large projects.

Example in Html



```
<html>
  <head>
    <title>Home</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

Now in EXTERNAL CSS

```
h1{
   color: blue;
}
p{
   color: green;
}
```

Characteristics:

- **Scope**: Styles can be shared across multiple HTML files.
- **Advantages**:
  - Promotes reusability and maintainability.
  - Keeps the HTML file cleaner and more readable.
  - Easy to update styles across multiple pages by modifying a single file.
- **Disadvantages**:
  - Requires an additional HTTP request to fetch the CSS file (though this can be mitigated by caching).
  - Not ideal for small projects with limited styling needs.

3. Inline CSS

Inline CSS is applied directly to an HTML element using the `style` attribute. It is the most specific method of applying styles.

Example

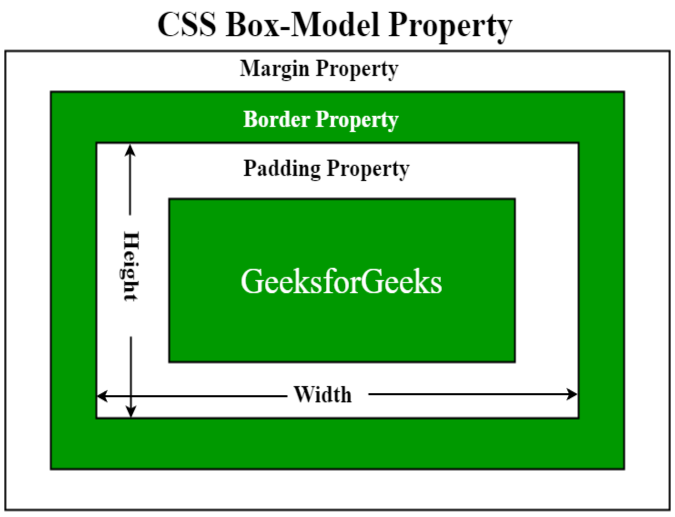
```
<!DOCTYPE html>
<html>
<body>
  <h1 style="color: navy; background-color: lightblue;">Welcome to My Page</h1>
</body>
</html>
```

Characteristics:

- **Scope**: Styles apply only to the specific element where they are written.
- **Advantages**:
  - Quick and easy for small, specific changes.
  - Does not require a separate CSS file or `<style>` block.
- **Disadvantages**:
  - Does not separate content and presentation, making the code harder to maintain.
  - Increases file size and redundancy if applied to multiple elements.
  - Overrides external and internal CSS unless `!important` is used.

CSS BOX MODEL

Question : Explain the CSS box model and its components (content, padding, border, margin). How does each affect the size of an element?



The CSS box model is a way to think of HTML elements as rectangular boxes with four main parts: content, padding, border, and margin. Each part affects the size of the element in different ways\

Content

- The area where the content of the element is displayed, such as text, images, or videos
- The size of the content box is determined by the amount of content and its size

Padding

- The space between the content and the border
- Padding can be applied to all sides of the box or to specific sides
- Increasing the padding makes the content look cleaner and easier to read

Border

- The area that surrounds the padding and content
- The border can be applied to all sides of the box or to specific sides
- The thickness of the border affects the height and width of the element

Margin

- The space between the border of the element and other elements
- The margin is transparent and shows the background color of the element
- Increasing the margin makes the space between elements larg

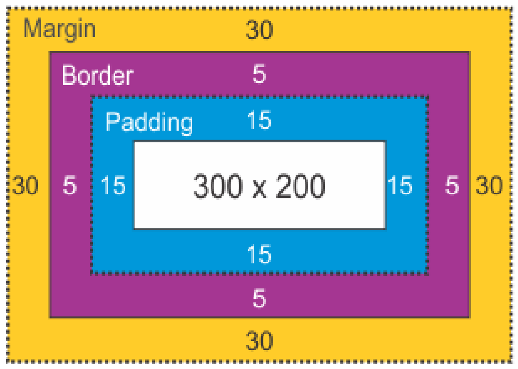
How do these parts affect the size of an element

- The default width and height of an element only applies to the content box
- The border and padding are added to the width and height to determine the size of the box on the screen
- The margin affects the total space the box takes up on the page, but it's not included in the box's actual size

Question : What is the difference between border-box and content-box box-sizing in CSS? Which is the default?

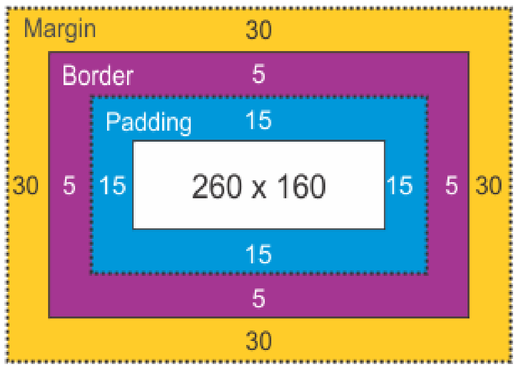
border-box and content-box are the two different values of box-sizing. content-box: This is the default value of box-sizing. The dimension of element only includes 'height' and 'width' and does not include 'border' and 'padding' given to element. Padding and Border take space outside the element

Box Model is content-box



```
div{
      width: 300px;
      height: 200px;
      padding: 15px;
      border: 5px solid grey;
      margin: 30px;
      -moz-box-sizing: content-box;
      -webkit-box-sizing: content-box;
      box-sizing: content-box;
    }
```

Box Model is border-box



```
div{
      width: 300px;
      height: 200px;
      padding: 15px;
      border: 5px solid grey;
      margin: 30px;
      -moz-box-sizing: border-box;
      -webkit-box-sizing: border-box;
      box-sizing: border-box;
    }
```

CSS Flexbox

Question 1: What is CSS Flexbox, and how is it useful for layout design? Explain the terms flex-container and flex-item.

What is CSS Flexbox?

CSS Flexbox, short for **Flexible Box Layout**, is a layout model in CSS designed to provide a more efficient way to align, distribute, and organize items within a container. It excels in creating responsive designs, allowing elements to adjust dynamically based on the size of their container and the available space. Flexbox is particularly useful for one-dimensional layouts, either along a horizontal or vertical axis.

Key Features of Flexbox:

1. **Flexible Sizing**: Elements can shrink or grow based on container space.
2. **Alignment Options**: Align elements horizontally and vertically without using complex hacks.
3. **Reordering**: Items can be reordered visually without changing the underlying HTML structure.
4. **Responsiveness**: Handles dynamic resizing and rearrangement of content on different screen sizes.

Key Terms in Flexbox:

1. **Flex Container**:
   - The parent element that holds the flex items.
   - Defined by setting the `display` property to `flex` or `inline-flex`.
   - The flex container dictates how its child elements (flex items) behave and are arranged.
2. **Flex Item**:

- The child elements of a flex container.
- These items follow the layout rules defined by the container.
- They can be manipulated individually using properties like `flex-grow`, `flex-shrink`, and `flex-basis`.

Key Properties of Flexbox:

On the Flex Container:

- **`flex-direction`**: Determines the main axis (`row`, `column`, etc.).
- **`justify-content`**: Aligns items along the main axis.
- **`align-items`**: Aligns items along the cross axis.

- **`align-content`**: Controls the space between rows (if there are multiple lines).
- **`flex-wrap`**: Allows items to wrap onto new lines if needed.

On the Flex Items:

- **`order`**: Changes the order of items.
- **`flex-grow`**: Defines how much space an item can take up relative to others.
- **`flex-shrink`**: Specifies how much an item should shrink when space is limited.
- **`flex-basis`**: Sets the default size of an item before resizing.
- **`align-self`**: Aligns a single item independently along the cross axis.

Why is Flexbox Useful for Layout Design?

1. **Simplifies Alignment**: Aligning items vertically or horizontally becomes straightforward.
2. **Space Distribution**: It dynamically distributes available space among items.
3. **Responsive Design**: Handles resizing and wrapping, making layouts adapt to different screen sizes.
4. **Eliminates Float/Clear Fixes**: Replaces older CSS techniques like floats for layouts.
5. **Versatile**: Works well for both small components (like navigation bars) and larger structures (like entire page layouts).

By mastering Flexbox, you can create modern, clean, and efficient layouts with minimal code.

Question 2: Describe the properties justify-content, align-items, and flex- direction used in Flexbox.

Here's a breakdown of the **`justify-content`**, **`align-items`**, and **`flex-direction`** properties in Flexbox:

1. `justify-content`

The `justify-content` property is used to align flex items **along the main axis** (the primary axis defined by the `flex-direction` property). It controls the horizontal alignment for `flex-direction: row` and vertical alignment for `flex-direction: column`.

Common Values:

- **`flex-start`**: Items are aligned at the beginning of the main axis (default value).
- **`flex-end`**: Items are aligned at the end of the main axis.
- **`center`**: Items are centered along the main axis.
- **`space-between`**: Items are spaced out evenly, with the first item at the start and the last item at the end.
- **`space-around`**: Items are evenly spaced, but there's half-size spacing at the start and end.

- **`space-evenly`**: Items are evenly distributed with equal spacing between them and around them.

2. `align-items`

The `align-items` property aligns flex items **along the cross axis** (perpendicular to the main axis). It determines how the items are positioned vertically (for `flex-direction: row`) or horizontally (for `flex-direction: column`).

Common Values:

- **`stretch`**: Items stretch to fill the container along the cross axis (default value).
- **`flex-start`**: Items are aligned at the start of the cross axis.
- **`flex-end`**: Items are aligned at the end of the cross axis.
- **`center`**: Items are centered along the cross axis.
- **`baseline`**: Items are aligned along their text baseline.

3. `flex-direction`

The `flex-direction` property defines the direction of the main axis, determining the flow of flex items in the container.

Common Values:

- **`row`**: Items are placed from left to right (default value).
- **`row-reverse`**: Items are placed from right to left.
- **`column`**: Items are placed from top to bottom.
- **`column-reverse`**: Items are placed from bottom to top.

CSS Grid

Question : Explain CSS Grid and how it differs from Flexbox. When would you use Grid over Flexbox?

What is CSS Grid?

CSS Grid is a **two-dimensional layout system** in CSS that allows you to design web pages by creating layouts in rows and columns simultaneously. It provides a powerful way to divide a webpage into regions or align items within a container.

CSS Grid uses a **grid container** (parent) that defines rows and columns, and **grid items** (children) that are placed within the defined grid structure.

Key Features of CSS Grid:

1. **Two-dimensional Layout**: Works on both rows and columns at the same time, making it ideal for complex layouts.
2. **Explicit Control**: You can define the size, position, and spacing of rows and columns with precision.
3. **Template Areas**: Allows naming grid areas, making layouts easier to manage and visualize.
4. **Responsive Design**: Easily adapts to different screen sizes using fractional units, auto-sizing, and media queries.

What is CSS Grid?

CSS Grid is a **two-dimensional layout system** in CSS that allows you to design web pages by creating layouts in rows and columns simultaneously. It provides a powerful way to divide a webpage into regions or align items within a container.

CSS Grid uses a **grid container** (parent) that defines rows and columns, and **grid items** (children) that are placed within the defined grid structure.

Key Features of CSS Grid:

1. **Two-dimensional Layout**: Works on both rows and columns at the same time, making it ideal for complex layouts.
2. **Explicit Control**: You can define the size, position, and spacing of rows and columns with precision.
3. **Template Areas**: Allows naming grid areas, making layouts easier to manage and visualize.
4. **Responsive Design**: Easily adapts to different screen sizes using fractional units, auto-sizing, and media querie

How CSS Grid Differs from Flexbox

Feature	CSS Grid	Flexbox
Layout Type	Two-dimensional (rows and columns).	One-dimensional (row *or* column).
Usage	Best for entire page layouts or large sections.	Best for aligning smaller components like navigation bars.
Alignment	Rows and columns can be aligned simultaneously.	Aligns items along a single axis at a time.
Explicit Positioning	Provides direct placement of items using grid lines or named areas.	No direct placement; items are arranged in sequence.
Responsiveness	Built-in tools like `grid-template-areas` and fractional units (`fr`).	Relies on properties like `flex-grow` and wrapping for responsiveness.

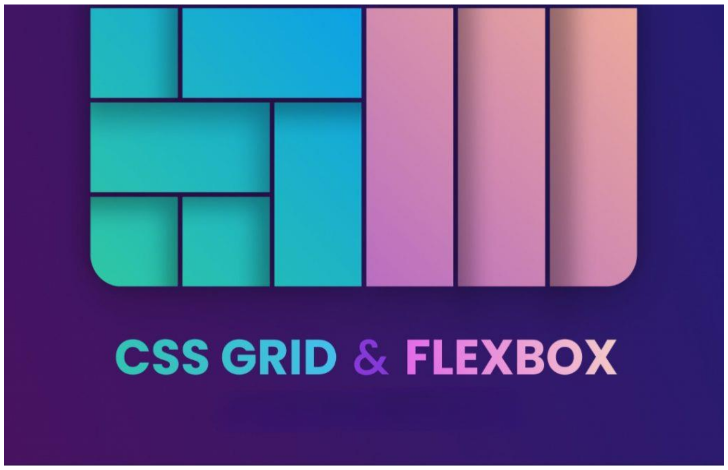
When to Use CSS Grid Over Flexbox

Use CSS Grid when:

1. **You need a two-dimensional layout**:
   - For example, creating a webpage with a header, sidebar, main content, and footer.
2. **Precise placement is required**:
   - For example, positioning items in specific rows and columns.
3. **You're building larger layouts**:
   - Complex layouts like dashboards, gallery grids, or multi-sectioned web pages are easier with Grid.

Use Flexbox when:

1. **You need a one-dimensional layout**:
   - For example, creating a navigation bar, buttons, or aligning items in a single row or column.
2. **Content size is dynamic**:
   - For example, layouts where items need to shrink or grow based on the container's size.
3. **You need to align items quickly**:
   - For example, centering items within a container using `justify-content` or `align-items`.



Question : Describe the grid-template-columns, grid-template-rows, and grid-gap properties. Provide examples of how to use them.

Description of Properties in CSS Grid

1. `grid-template-columns`

The `grid-template-columns` property defines the number, size, and layout of **columns** in a grid container. It specifies the column widths and creates the structure for grid items.

- Values can be:
  - **Fixed Units**: `px`, `em`, `rem`, etc. (e.g., `100px 200px`)
  - **Flexible Units**: `%` or `fr` (fractional units, like `1fr`, to distribute space proportionally).
  - **Auto**: Automatically sizes columns based on content.
  - **Repeat() Function**: Simplifies repetitive column definitions.

Example:

.container {

display: grid;

grid-template-columns: 100px 200px 1fr;

}

- Creates 3 columns:
  - First column: 100px wide.
  - Second column: 200px wide.
  - Third column: Takes up the remaining available space.

Description of Properties in CSS Grid

1. `grid-template-columns`

The `grid-template-columns` property defines the number, size, and layout of **columns** in a grid container. It specifies the column widths and creates the structure for grid items.

- Values can be:
  - **Fixed Units**: `px`, `em`, `rem`, etc. (e.g., `100px 200px`)
  - **Flexible Units**: `%` or `fr` (fractional units, like `1fr`, to distribute space proportionally).
  - **Auto**: Automatically sizes columns based on content.
  - **Repeat() Function**: Simplifies repetitive column definitions.

Example:

```
css
CopyEdit
.container {
    display: grid;
    grid-template-columns: 100px 200px 1fr;
}
```

- Creates 3 columns:
  - First column: 100px wide.
  - Second column: 200px wide.
  - Third column: Takes up the remaining available space.

2. `grid-template-rows`

The `grid-template-rows` property defines the number, size, and layout of **rows** in a grid container. It works similarly to `grid-template-columns` but for rows.

- Values are the same as for `grid-template-columns`.

Example:

```
.container {
   display: grid;
   grid-template-rows: 100px auto 2fr;
}
```

- Creates 3 rows:
  - First row: 100px high.
  - Second row: Automatically sized based on content.
  - Third row: Takes up twice as much space as a single `fr` unit.

3. `grid-gap` (or `gap`)

The `grid-gap` property defines the spacing **between rows and columns** in the grid. It has been replaced by the shorthand `gap` in modern CSS but is still supported in older browsers.

- **Syntax**:
  - `gap: row-gap column-gap;`
  - If a single value is provided, it applies to both row and column gaps.

Example:

```
.container {
   display: grid;
   grid-template-columns: 1fr 1fr;
   grid-template-rows: auto;
   gap: 20px 30px; /* 20px row gap, 30px column gap */
}
```

Adds 20px of space between rows and 30px between columns.