Module 3 – Frontend – CSS and CSS3

CSS Selectors & Styling

Theory Assignment

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

A CSS selector is used to identify and select specific HTML elements on a web page so that styles can be applied to them through CSS rules. Different types of selectors allow targeting elements based on their name, class, or ID.

Element Selector

The element (type) selector targets all HTML elements of a given type by simply using the element's name. For example:

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

This rule applies the color green to every `<p>` (paragraph) tag on the page.

Class Selector

The class selector targets all elements that have a specific class attribute. Syntax uses a period (`.`) followed by the class name:

```
 .center {      text-align: center;

  color: red;

}
```

This will style every element with class `center`.

ID Selector

The ID selector targets a unique element based on its `id` attribute. The syntax uses a hash (`#`) followed by the ID name:

```
#para1 {

  text-align: center;
```

```
  color: red;
}
```

This will style the element with `id="para1"` only.

Question 2: Explain the concept of CSS specificity. How do conflicts between multiple stylesget resolved?

CSS specificity is a measurement used to determine which style rules are applied to an HTML element when multiple rules could apply.

How Specificity Is Calculated

- Inline styles (inside the HTML tag): 1000 points
- ID selectors: 100 points for each
- Class selectors, attribute selectors, and pseudo-classes: 10 points for each
- Element selectors and pseudo-elements: 1 point for each
- Universal selector (*) has no specificity

This is often expressed as a four-part score: `[inline, IDs, classes/attributes/pseudo-classes, elements/pseudo-elements]`. For example, the selector `#content .note h2` would have the score `[0,1,1,1]`, equaling a total specificity of 111.

Resolving Style Conflicts

- The style with the highest specificity score is applied.
- If two rules share the exact specificity, the rule declared last in the CSS will be used.
- Inline styles will always override styles from external or internal CSS, unless those styles use.

Question 3: What is the difference between internal, external, and inline CSS? Discuss the advantages and disadvantages of each approach.

Internal, external, and inline CSS are three main ways to apply styles in web development, each with unique advantages and disadvantages.

Inline CSS

- Definition: Styles are applied directly to an HTML element using the `style` attribute.
- Advantages: Quick to implement for one-off changes; highest specificity, overriding other CSS rules.
- Disadvantages: Hard to maintain and update; makes HTML files larger; poor reusability for multiple elements or pages.

Internal CSS

- Definition: Styles are defined inside a `<style>` tag within the `<head>` of the HTML document.
- Advantages: Useful for one-page websites or when applying styles specific to a single page; easier maintenance than inline.
- Disadvantages: Increases the size of the HTML document; styles cannot be reused across multiple pages; redundancy if styles are copied between pages.

External CSS

- Definition: Styles are placed in a separate `.css` file and linked to HTML documents using the `<link>` tag in the `<head>`.
- Advantages: Enables consistent styling across multiple pages; keeps HTML cleaner and more maintainable; CSS files can be cached for faster loading.
- Disadvantages: Requires an additional HTTP request to load the stylesheet, which may slightly delay rendering; if the external file fails to load, styles won't be applied.

Feature	**Inline CSS**	**Internal CSS**	**External CSS**
Location	In the element's style attribute	`<style>` in document head	Separate `.css` file

Scope	One element	Entire HTML document	Multiple HTML documents
Maintainability	Poor	Moderate	Best
Reusability	None	Limited	High
Performance	Fast for simple changes	Can slow with many styles	Efficient with caching

Lab Assignment

Style the contact form (created in the HTML Forms lab) using external CSS. The following should be implemented:

Change the background color of the form.

Add padding and margins to form fields.

Style the submit button with a hover effect.

Use class selectors for styling common elements and ID selectors for unique elements.

First, ensure your HTML form is correctly structured and linked to an external CSS file (e.g., `style.css`). Notice the use of classes for common elements and an ID for the unique submit button.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Form</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <form class="contact-form">
        <h2>Contact Us</h2>
        <div class="form-group">
            <label for="name">Name:</label>
            <input type="text" id="name" class="form-field" name="name" required>
        </div>

        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" class="form-field" name="email" required>
        </div>
        <div class="form-group">
            <label for="message">Message:</label>
            <textarea id="message" class="form-field" name="message" rows="5"
required></textarea>
```

```
        </div>
        <button type="submit" id="submit-btn">Send Message</button>
    </form>
</body>
</html>
```

1. Style the Form Container

We use a class selector (`.contact-form`) to target the main form element and change its background.

```
/* Styling the Form Container */
.contact-form {
    /* Change the background color of the form */
    background-color: #f4f4f4;

    /* Center the form and limit its width */
    max-width: 500px;
    margin: 50px auto;
    padding: 30px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```

2. Style Form Fields (Inputs and Textarea)

We use a class selector (`.form-field`) to apply common padding and margins to the input and textarea elements.

```
/* Styling Common Form Fields (Inputs and Textarea) */

.form-field {

    width: 100%;

    /* Add padding and margins to form fields */

    padding: 10px;

    margin-top: 5px; /* Margin above the field */

    margin-bottom: 20px; /* Margin below the field */



    box-sizing: border-box; /* Ensures padding is included in the element's total width/height */

    border: 1px solid #ccc;

    border-radius: 4px;

}
```

3. Style the Submit Button

We use an ID selector (`#submit-btn`) for unique styling, and the `:hover` pseudo-class for the required effect.

```
/* Styling the Submit Button (Unique Element using ID) */

#submit-btn {

    /* Base style for the button */

    background-color: #007bff;

    color: white;

    padding: 12px 20px;

    border: none;
```

```
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease; /* Smooth transition for hover effect */
}

/* Style the submit button with a hover effect */
#submit-btn:hover {
    background-color: #0056b3; /* Darker blue on hover */
}
```

CSS Box Model

Theory Assignment

Question 1: 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 fundamental concept that explains how the size and spacing of everyHTML element are calculated and rendered on a webpage. Each element is represented as a rectangular box composed of four components: content, padding, border, and margin.

Components of the CSS Box Model

Content: This is the innermost part of the box where the actual content like text, images, or media is displayed. Its size is defined by CSS properties such as `width` and `height`.

Padding: The padding is the transparent space between the content and the border. It creates space inside the element around the content. Padding increases the total size of the element because it adds space around the content within the border.

Border: The border wraps around the padding and content. It is visible and can have a width, color, and style (solid, dashed, etc.). The border also contributes to the total element size by adding to the dimensions outside the padding.

Margin: The margin is the outermost transparent area that creates space between the border of the element and surrounding elements, helping to separate elements from each other. Margins do not affect the background of the element and are also added to the total space the element occupies externally.

Question 2: What is the difference between border-box and content-box box-sizing inCSS? Which is the default?

The difference between `border-box` and `content-box` values of the CSS `box-sizing` property lies in how the total size of an element is calculated:

- content-box (the default value): The width and height you set apply only to the content area of the element. Padding and border are added *outside* this content

size, increasing the element's total rendered size. So if you set `width: 200px` with padding and border, the visible element becomes wider than 200px because padding and border add to that width.

- border-box: The width and height you set *include* the content, padding, and border. This means padding and border are drawn *inside* the specified size, and the content area shrinks to accommodate them. If you set `width: 200px` with padding and border, the overall element stays 200px wide, with the content area reduced accordingly.

Content-box bases the size only on content (padding and border add on top), while border-box treats the size as the full box including padding and border.

Formulas for a set width:

content-box: total width = content width W + padding + border

border-box: total width = W (content + padding + border combined)

Lab Assignment

Task

Create a profile card layout using the box model. The profile card should include:

A profile picture.

The user’s name and bio.

A button to "Follow" the user.

Additional Requirements:

Add padding and borders to the elements.

Ensure the layout is clean and centered on the page using CSS margins.

Use the box-sizing property to demonstrate both content-box and border-box on different elements.

```
<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>Profile Card Layout</title>

   <link rel="stylesheet" href="style.css">

</head>

<body>

   <div class="profile-card">

      <img class="profile-picture content-box-example" src="profile-placeholder.png" alt="Profile Picture">

         <div class="profile-info">

         <h2 class="user-name border-box-example">Jane Doe</h2>

         <p class="user-bio border-box-example">

            Front-end developer passionate about clean code and responsive design. Currently exploring CSS Grid and Flexbox integration.

         </p>

         <button class="follow-button">Follow</button>

      </div>

   </div>

</body>
```

</html>

This CSS file implements the Box Model properties (margin, border, padding) and demonstrates both `content-box` and `border-box`.

```
/* GLOBAL RESET: Use border-box by default for most elements for easier layout */

* {

    box-sizing: border-box;

    margin: 0;

    padding: 0;

}

/* 1. LAYOUT AND CENTERING */

.profile-card {

    /* Define the card's dimensions */

    width: 350px;

    /* Ensure the layout is clean and centered on the page using CSS margins */

    margin: 50px auto;

    /* Box Model Styling for the main card container */

    background-color: #ffffff;

    border: 1px solid #ddd;

    padding: 20px;

    border-radius: 10px;

    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);

    text-align: center;
```

```
}

/* 2. BOX-SIZING DEMONSTRATIONS */

/* Example of box-sizing: content-box (Default CSS behavior) */

/* The width/height only applies to the content area. Padding and border are added OUTSIDE of the defined width. */

.content-box-example {

    box-sizing: content-box;

    /* This element is now 100px (content) + 10px*2 (padding) + 3px*2 (border) = 126px wide in total */

    width: 100px;

    height: 100px;

}

/* Example of box-sizing: border-box (Modern, easier behavior) */

/* The width/height includes content, padding, AND border. */

.border-box-example {

    box-sizing: border-box;

    /* This element's total width remains the same, the content shrinks to fit the padding and border inside. */

    width: 100%;

}

/* 3. ELEMENT STYLING AND BOX MODEL */

/* Profile Picture Styling */

.profile-picture {
```

```
    /* Dimensions from content-box-example above */
    border-radius: 50%;
    object-fit: cover;
    display: block;
    margin: 0 auto 20px auto; /* Centering the image and adding margin below */
    border: 3px solid #6c5ce7; /* Add borders */
    padding: 10px; /* Add padding */
}
/* User Name Styling */
.user-name {
    font-size: 1.8em;
    color: #333;
    margin-bottom: 5px; /* Margin below the name */
    padding: 10px; /* Add padding (included in width due to border-box) */
    border-bottom: 2px dashed #eee; /* Add borders */
}
/* User Bio Styling */
.user-bio {
    font-size: 0.9em;
    color: #666;
    margin-bottom: 25px; /* Margin below the bio */
    padding: 15px; /* Add padding (included in width due to border-box) */
```

```
    border: 1px solid #e0e0e0; /* Add borders */
}

/* Follow Button Styling */
.follow-button {
    background-color: #6c5ce7;
    color: white;
    font-size: 1em;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    /* Box Model */
    padding: 10px 20px; /* Add padding */
    margin: 5px 0; /* Add margins */
    transition: background-color 0.3s;
}
.follow-button:hover {
    background-color: #4a3e9c;
}
```

CSS Flexbox

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

CSS Flexbox, short for Flexible Box Layout, is a one-dimensional layout model designed to arrange and distribute space among items within a container efficiently. It allows items to flexibly grow, shrink, and be aligned in either a row or a column, making it ideal for responsive and dynamic web layouts.

Flex-container and Flex-item

- Flex-container: The parent element with its CSS `display` property set to `flex` or `inline-flex`. This container establishes the flex formatting context that governs the layout of its children.
- Flex-items: The direct child elements inside a flex-container. These items automatically become flexible and can be sized and aligned based on the flexbox properties applied to the container and the items themselves.

Usefulness for Layout Design

Flexbox simplifies tasks like:

- Aligning items horizontally or vertically without complex floats or positioning.
- Distributing space evenly or with specific gaps among items.
- Handling dynamic or unknown item sizes by allowing them to grow or shrink to fill available space.
- Easily controlling the order, alignment, and spacing along both the main axis (row or column direction) and the cross axis (perpendicular direction).
-

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

The `flex-direction` property defines the main axis of the flex container, establishing the primary direction in which flex items are laid out.

- Applied to: The flex container.
- Purpose: Determines the direction (horizontal or vertical) of the items.

Value	Description	Main Axis	Cross Axis
row (Default)	Items are arranged horizontally, from left to right (in LTR writing mode).	Horizontal	Vertical
row-reverse	Items are arranged horizontally, from right to left.	Horizontal	Vertical
column	Items are arranged vertically, from top to bottom.	Vertical	Horizontal
column-reverse	Items are arranged vertically, from bottom to top.	Vertical	Horizontal

The `justify-content` property defines how the browser distributes space between and around flex items along the main axis. This property is typically used when the total size of the flex items is less than the size of the flex container along the main axis.

- Applied to: The flex container.
- Purpose: Controls the alignment and spacing of items along the main axis.

Value	Description

flex-start(Default)	Items are packed towards the start of the main axis.
flex-end	Items are packed towards the end of the main axis.
center	Items are centered along the main axis.
space-between	Items are evenly distributed; the first item is at the start, and the last item is at the end.
space-around	Items are evenly distributed with equal space around them. Note: the space on the ends is half the space between items.
space-evenly	Items are distributed so that the space between any two adjacent items, and the space before the first and after the last item, is equal.

The `align-items` property defines the default behavior for how flex items are laid out along the cross axis on the current line. It essentially aligns the items perpendicular to the main axis.

- Applied to: The flex container.
- Purpose: Controls the alignment of items along the cross axis.

The crucial link is that `flex-direction` determines which axis is the main axis (controlled by `justify-content`) and which is the cross axis (controlled by `align-items`).

For example, to center an item both horizontally and vertically in a standard row layout:

- Use `flex-direction: row;` (Main axis is horizontal).
- Use `justify-content: center;` (Centers horizontally along the main axis).

- Use `align-items: center;` (Centers vertically along the cross axis).

Lab Assignment

Task

Create a simple webpage layout using Flexbox. The layout should include:

A header.

A sidebar on the left.

A main content area in the center. A footer.

Additional Requirements:

Use Flexbox to position and align the elements.

Applydifferentjustify-contentandalign-itemspropertiestoobservetheireffects.

Ensure the layout is responsive, adjusting for smaller screens.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Flexbox Page Layout</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <div class="page-container">

        <header class="header">

            <h1>Flexbox Page Layout Header</h1>
```

```
    </header>
    <div class="main-body">
      <aside class="sidebar">
        <h2>Sidebar</h2>
        <p>Navigation links or secondary content go here.</p>
        <p>This element is set to a fixed width.</p>
      </aside>

      <main class="main-content">
        <h2>Main Content Area</h2>
        <p>This area will take up the remaining flexible space.</p>
        <p>We'll use Flexbox on the **.main-content** to center its text vertically.</p>
      </main>
    </div>

    <footer class="footer">
      <p>&copy; 2025 Flexbox Layout Demo</p>
    </footer>
  </div>
</body>
</html>
```

1. Base Styles & Global Flex Containers

We use `display: flex` on the `.main-body` to arrange the sidebar and main content horizontally, and on the `.header` and `.footer` to align their content.

```
/* Base Styling and Box Model Setup */

body {

    font-family: Arial, sans-serif;

    background-color: #f4f4f9;

    min-height: 100vh; /* Ensure body takes full viewport height */

    display: flex;

    flex-direction: column;

}

.page-container {

    width: 90%;

    max-width: 1200px;

    margin: 20px auto;

    border: 1px solid #ccc;

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

    flex-grow: 1; /* Allows container to fill the height */

}

/* FLEXBOX APPLICATION 1: HEADER & FOOTER */

/* Demonstrating different justify-content and align-items */

.header, .footer {
```

```
    background-color: #333;
    color: white;
    padding: 20px;
    /* Use Flexbox for alignment */
    display: flex;
    /* Header: Align content to the center of the main axis (horizontal) */
    justify-content: center;
    /* Footer: Align content to the end of the cross axis (vertical - not visible with single line) */
    align-items: flex-end;
}
/* FLEXBOX APPLICATION 2: MAIN BODY (Sidebar + Content) */
.main-body {
    display: flex;
    /* Main body should fill the space between header and footer */
    min-height: 500px;
}
.sidebar {
    /* Sidebar has a fixed width */
    width: 250px;
    background-color: #e9e9e9;
    padding: 20px;
```

```
    border-right: 1px solid #ccc;
    /* Prevent sidebar from shrinking */
    flex-shrink: 0;
}
.main-content {
    /* Main content takes all remaining space */
    flex-grow: 1;
    background-color: #ffffff;
    padding: 20px;
    /* Use Flexbox *INSIDE* the main content to center text */
    display: flex;
    flex-direction: column; /* Stacks content vertically */
    /* Center text along the main axis (vertical) and cross axis (horizontal) */
    justify-content: center;
    align-items: center;
    text-align: center;
}
```

2. Responsiveness with Media Query

This media query adjusts the layout for screens smaller than **768px** by changing the main axis from `row` (horizontal) to `column` (vertical).

```
/* RESPONSIVENESS */
@media (max-width: 768px) {
```

```
    .main-body {
        /* Change the main axis direction for smaller screens */
        flex-direction: column;
    }
    .sidebar {
        /* Sidebar now takes the full width */
        width: 100%;
        border-right: none;
        border-bottom: 1px solid #ccc;
    }
}
```

CSS Grid

Theory Assignment

Question 1: Explain CSS Grid and how it differs from Flexbox. When would you use Grid over Flexbox?

CSS Grid Layout is a two-dimensional layout system for CSS. It lets you organize content into rows and columns simultaneously, providing a robust structure for creating complex, organized, and responsive web page designs.

When you use Grid, you define a Grid Container (by setting `display: grid`) and then specify the layout blueprint using properties like:

- `grid-template-columns` and `grid-template-rows`: Explicitly define the size and number of columns and rows.
- `grid-template-areas`: Allows you to name parts of the grid (e.g., `header`, `sidebar`, `main`) and place items visually in the CSS.
- `grid-column` and `grid-row`: Used on the child items to specify exactly which grid lines or areas they should span.
- `gap` (or `row-gap`/`column-gap`): Creates uniform spacing between grid tracks.

Feature	CSS Grid	CSS Flexbox
Dimensionality	Two-Dimensional (Controls both rows and columns at the same time).	One-Dimensional (Controls elements in either a rowOR a column at one time).
Control	"Layout-First" approach. You define the container's grid structure, and then place the	"Content-Out" approach. The items flow along a single axis, and the size of the content often dictates the spacing and flexibility.

	items into the predefined cells/areas.	
Multi-line	When an item spans multiple rows or columns, it remains aligned with the other rows/columns in the grid.	With flex-wrap: wrap, items wrap to new lines, but each wrapped line is an independent "flex-line." Items in different rows do not automatically align vertically with items in other rows.
Overlap	Can easily place multiple items into the same grid cell for intentional overlap (using z-index to control layering).	Cannot easily achieve clean item overlap without resorting to absolute positioning workarounds.

Use CSS Grid for:

- Full-Page Layouts: Building the major structure of a web page, such as the overall layout for a header, footer, sidebar, and main content area.
- Complex, Two-Dimensional Designs: Any layout where you need items to align perfectly in both rows and columns, like a product catalog, a complex dashboard, or a classic 12-column grid.
- Precise Placement: When you need to define an explicit grid and place items exactly at a specific intersection of column and row lines, or have items span multiple tracks.

Continue to use Flexbox for:

- UI Components / Small-Scale Layouts: Arranging a small group of items within a single component, such as:
    - Items in a navigation bar.
    - Aligning an icon next to text within a button.
    - Centering content perfectly in a single row or column.

- One-Dimensional Alignment: When you only need to control flow, spacing, and alignment along a single line (row or column).

Question 2: Describe the grid-template-columns, grid-template-rows, and grid-gap properties. Provide examples of how to use them.

The properties `grid-template-columns`, `grid-template-rows`, and `gap` (or `grid-gap`) are foundational to defining the structure and spacing of a CSS Grid Layout. They are all applied to the Grid Container.

Property	Purpose
grid-template-columns	Defines the number and width of the columns in the grid.
grid-template-rows	Defines the number and height of the rows in the grid.

Key Units and Keywords

Grid provides powerful units and keywords for defining track sizes:

- Fixed/Absolute Units: `px`, `em`, `rem`, etc. (e.g., `100px`)
- Percentage Units: `%` of the grid container's size. (e.g., `25%`)
- The Flexible `fr` Unit: Represents a fraction of the available free space in the grid container. This is the most common and powerful unit for responsive design.
- `minmax(min, max)`: Sets a minimum and maximum size range for a track (e.g., `minmax(100px, 1fr)`).
- `repeat(count, track-list)`: A shorthand to repeat a pattern of track sizes (e.g., `repeat(3, 1fr)`).

Example: Defining Tracks

Let's define a grid with three columns and two rows:

```
.container {
  display: grid;
  /* Columns:
    1. 200px wide (fixed)
    2. 1 fraction of remaining space (flexible)
    3. 1 fraction of remaining space (flexible)
  */
  grid-template-columns: 200px 1fr 1fr;
  /* Rows:
    1. 100px tall (fixed)
    2. Minimum 50px, Maximum 1 fraction of remaining space
  */
  grid-template-rows: 100px minmax(50px, 1fr);
}
```

The `gap` property (which is a shorthand for `row-gap` and `column-gap`) defines the size of the gutters (spacing) between the rows and columns of the grid.

Property	Purpose
gap (shorthand)	Sets the spacing between both rows and columns.

row-gap	Sets the spacing between the rows.
column-gap	Sets the spacing between the columns.

Example: Setting Gaps

The `gap` property accepts one or two values:

1. One Value: Applies the same gap to both rows and columns.
2. Two Values: The first value applies to the row gap, and the second applies to the column gap.

```
.container {
  display: grid;
  /* Example 1: Uniform Gap
    15px between both rows and columns
  */
  gap: 15px;
  /* Example 2: Separate Gaps (overrides the first example)
    20px between rows (row-gap)
    10px between columns (column-gap)
  */
  gap: 20px 10px;
}
```

Lab Assignment

Task

Create a 3x3 grid of product cards using CSS Grid. Each card should contain:

A product image. A product title. A price.

Additional Requirements:

Usegrid-template-columnstocreatethegridlayout.

Usegrid-gaptoaddspacingbetweenthegriditems.

Apply hover effects to each card for better interactivity.

The HTML defines the main `.grid-container` and nine `.product-card` elements within it.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>CSS Grid Product Catalog</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <h1>Featured Products (3x3 Grid)</h1>

    <div class="grid-container">

        <div class="product-card">

            <img class="product-image" src="placeholder-1.jpg" alt="Product 1">
```

```
    <h3 class="product-title">Premium Gadget</h3>
    <p class="product-price">$49.99</p>
</div>

<div class="product-card">
    <img class="product-image" src="placeholder-2.jpg" alt="Product 2">
    <h3 class="product-title">Ergonomic Mouse</h3>
    <p class="product-price">$25.50</p>
</div>

<div class="product-card">
    <img class="product-image" src="placeholder-3.jpg" alt="Product 3">
    <h3 class="product-title">Smart Watch</h3>
    <p class="product-price">$199.00</p>
</div>

<div class="product-card">
    <img class="product-image" src="placeholder-4.jpg" alt="Product 4">
    <h3 class="product-title">Noise Canceller</h3>
    <p class="product-price">$140.00</p>
</div>
```

```
<div class="product-card">
    <img class="product-image" src="placeholder-5.jpg" alt="Product 5">
    <h3 class="product-title">Travel Backpack</h3>
    <p class="product-price">$75.00</p>
</div>

<div class="product-card">
    <img class="product-image" src="placeholder-6.jpg" alt="Product 6">
    <h3 class="product-title">Wireless Charger</h3>
    <p class="product-price">$30.99</p>
</div>

<div class="product-card">
    <img class="product-image" src="placeholder-7.jpg" alt="Product 7">
    <h3 class="product-title">E-Reader Pro</h3>
    <p class="product-price">$129.50</p>
</div>

<div class="product-card">
    <img class="product-image" src="placeholder-8.jpg" alt="Product 8">
    <h3 class="product-title">4K Monitor</h3>
    <p class="product-price">$350.00</p>
```

```
        </div>

        <div class="product-card">
            <img class="product-image" src="placeholder-9.jpg" alt="Product 9">
            <h3 class="product-title">Mechanical Keyboard</h3>
            <p class="product-price">$95.00</p>
        </div>
    </div>
</body>
</html>
```

This CSS implements the Grid layout and the required hover effect.

```
/* Base Styling and Setup */
body {
    font-family: 'Arial', sans-serif;
    background-color: #f4f7f8;
    margin: 0;
    padding: 20px;
    text-align: center;
}

h1 {
    color: #333;
```

```
    margin-bottom: 30px;
}

/* ---------------------------------- */
/* 1. CSS GRID CONTAINER SETUP */
/* ---------------------------------- */

.grid-container {
    display: grid;

    /* Use grid-template-columns to create the 3x3 layout:
       Three columns, each taking an equal fraction of the available space.
    */
    grid-template-columns: repeat(3, 1fr);

    /* Use grid-gap to add spacing between the grid items */
    gap: 30px;

    /* Center the entire grid container on the page */
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
```

```
}

/* ---------------------------------- */
/* 2. PRODUCT CARD STYLING */
/* ---------------------------------- */

.product-card {
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

    /* Add smooth transition for the hover effect */
    transition: transform 0.3s ease, box-shadow 0.3s ease;

    /* Ensure consistent height for all cards if content varies slightly */
    display: flex;
    flex-direction: column;
    justify-content: space-between;
}

.product-image {
```

```
    width: 100%;
    height: auto;
    max-height: 200px; /* Limit image height */
    object-fit: cover;
    border-radius: 4px;
    margin-bottom: 15px;
}
.product-title {
    font-size: 1.2em;
    color: #333;
    margin-bottom: 10px;
}
.product-price {
    font-size: 1.5em;
    color: #007bff;
    font-weight: bold;
    margin-top: 15px;
}
/* 3. HOVER EFFECT */
/* Apply hover effects to each card for better interactivity */
.product-card:hover {
    /* Lift the card up */
```

```
    transform: translateY(-5px);
    /* Enhance the shadow */
    box-shadow: 0 8px 15px rgba(0, 0, 0, 0.2);
    cursor: pointer;
}
```