**Grid Layout**

CSS Grid Layout (or CSS Grid) is a two-dimensional layout system that allows you to create complex grid-based layouts with rows and columns for web pages and user interfaces.

It provides a powerful way to design and structure content on a web page, offering precise control over the placement and alignment of elements.

Here are some key features and concepts of CSS Grid:

1. **Two-dimensional layout**: Unlike traditional layout methods like Flexbox, which focus on one dimension (either rows or columns), CSS Grid allows you to define both rows and columns simultaneously, creating a grid-based layout.
2. **Grid container and grid items**: In CSS Grid, you designate an element as a grid container by applying **display: grid;** or **display: inline-grid;** to it. The direct children of the grid container become grid items, which are placed within the grid.
3. **Grid tracks**: Grid tracks are the rows and columns that define the structure of the grid. You can explicitly define the size of tracks using length units (like pixels or percentages) or using flexible units (like **fr** units for distributing available space).
4. **Grid lines**: Grid lines are the horizontal and vertical lines that divide the grid into rows and columns. You can refer to these lines to position grid items within the grid.
5. **Grid areas**: Grid areas are rectangular areas within the grid formed by the intersection of specific rows and columns. You can assign grid items to specific grid areas, allowing for easy placement and alignment.
6. **Grid template**: The grid template defines the structure of the grid, specifying the number and size of rows and columns, as well as the placement of grid items within the grid areas.

CSS Grid provides a more powerful and flexible layout system compared to traditional methods like floats and positioning. It's particularly well-suited for creating complex layouts, including responsive designs and multi-column layouts, and it's widely supported by modern web browsers.

**fr units meaning**

CSS Grid introduced a new measurement unit to the world.

It's actually a new type of flexible unit called Fr unit.

The meaning of Fr is **Fractional Unit**, where 1fr represents a fraction of the available space.

**vh and vw**

vh stands for viewport height.

This unit is based on the height of the viewport.

A value of 1vh is equal to 1% of the viewport height.

A value of 100vh is equal to 100% of the viewport height.

1vw stands for viewport width.

**display: grid**

The **display: grid** property in CSS is used to create a grid layout system, which allows for the precise placement and alignment of elements within a container.

Grid layouts are powerful because they enable complex, two-dimensional layouts that can adapt to different screen sizes, making them ideal for responsive design.

**Basic Concept**

When you apply **display: grid** to a container, it becomes a **grid container**, and its children become **grid items**.

The grid container defines a grid structure using rows and columns, which the grid items are placed within.

**Syntax**

.container {

display: grid;

}

**Key Properties of CSS Grid**

1. **Grid Container Properties**:
   * **grid-template-columns**: Defines the number and size of columns in the grid.
   * **grid-template-rows**: Defines the number and size of rows in the grid.
   * **grid-column-gap** or **column-gap**: Sets the gap (space) between columns.
   * **grid-row-gap** or **row-gap**: Sets the gap (space) between rows.
   * **grid-gap** or **gap**: A shorthand for setting both column-gap and row-gap together.
   * **justify-items**: Aligns grid items along the horizontal axis.
   * **align-items**: Aligns grid items along the vertical axis.
   * **justify-content**: Aligns the grid within the container horizontally.
   * **align-content**: Aligns the grid within the container vertically.
2. **Grid Item Properties**:
   * **grid-column-start / grid-column-end**: Specifies the starting and ending positions of a grid item across columns.
   * **grid-row-start / grid-row-end**: Specifies the starting and ending positions of a grid item across rows.
   * **grid-area**: A shorthand for specifying grid item placement using grid-row-start, grid-column-start, grid-row-end, and grid-column-end.

**Example: Basic Grid Layout**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Grid Layout Example</title>

<style>

.container {

display: grid;

grid-template-columns: 1fr 2fr 1fr; /\* 3 columns with different widths \*/

grid-template-rows: auto auto; /\* 2 rows with automatic height \*/

gap: 10px; /\* Gap between rows and columns \*/

padding: 10px;

background-color: #f9f9f9;

border: 2px solid #333;

}

.item {

background-color: #4CAF50;

color: white;

padding: 20px;

font-size: 1.2em;

text-align: center;

}

.item:nth-child(1) {

grid-column: 1 / 3; /\* Span the first item across two columns \*/

}

.item:nth-child(3) {

grid-row: 1 / 3; /\* Span the third item across two rows \*/

}

</style>

</head>

<body>

<div class="container">

<div class="item">Item 1</div>

<div class="item">Item 2</div>

<div class="item">Item 3</div>

<div class="item">Item 4</div>

<div class="item">Item 5</div>

<div class="item">Item 6</div>

</div>

</body>

</html>

**Explanation**

* **Container (.container)**:
  + display: grid: Turns the container into a grid container.
  + grid-template-columns: 1fr 2fr 1fr;: Defines three columns where the first and third columns are equal width (1fr), and the middle column is twice as wide (2fr).
  + grid-template-rows: auto auto;: Defines two rows with automatic height based on the content.
  + gap: 10px;: Adds a 10px gap between rows and columns.
* **Items (.item)**:
  + Each grid item is placed in the grid according to the order of the HTML structure.
  + The first item spans across two columns using grid-column: 1 / 3;.
  + The third item spans across two rows using grid-row: 1 / 3;.

**Result**

In this example, a grid layout is created with six items. The grid has three columns and two rows:

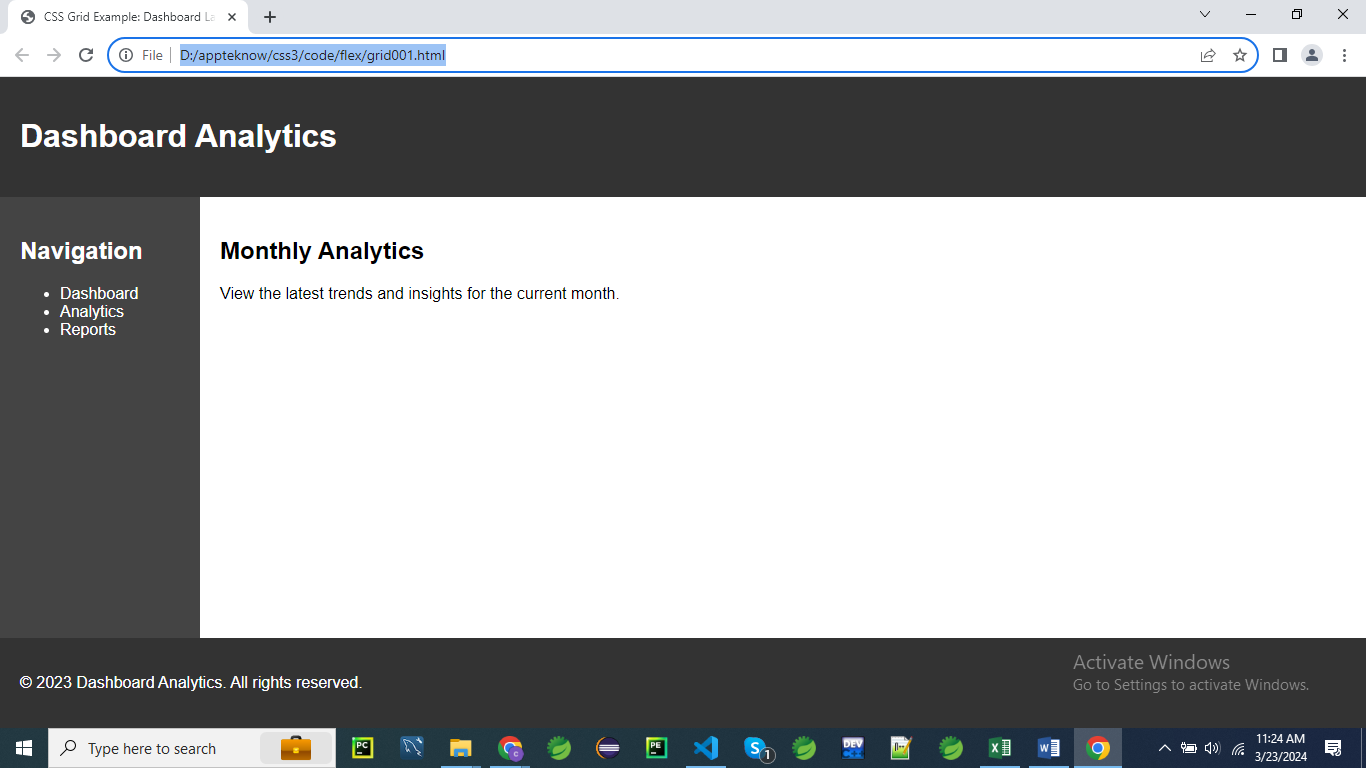
* **Item 1** spans across the first two columns on the first row.
* **Item 3** spans across two rows in the middle column.

**Use Cases for CSS Grid**

* **Complex Layouts**: Grid is ideal for creating complex layouts where precise control over both rows and columns is required.
* **Responsive Design**: Grid allows for easy rearrangement of content at different screen sizes.
* **Layouts with Overlapping Items**: Grid can handle overlapping items, enabling creative layouts like magazine-style pages.

CSS Grid is a versatile layout system that complements Flexbox, giving you the tools to create sophisticated and responsive layouts with relative ease.

**Example 01:**



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>CSS Grid Example: Dashboard Layout</title>

<style>

    /\* CSS styles \*/

    .container {

        display: grid; /\* Activate CSS Grid layout for the container \*/

        grid-template-columns: 200px 1fr; /\* Define two columns: a fixed-width sidebar and a flexible main content area \*/

        grid-template-rows: auto 1fr auto; /\* Define three rows: auto for header and footer height, and 1fr for main content area \*/

        grid-template-areas:

            "header header"

            "sidebar content"

            "footer footer"; /\* Define grid areas for header, sidebar, content, and footer \*/

        min-height: 100vh; /\* Set minimum height to cover the viewport \*/

    }

    header {

        grid-area: header; /\* Place header in the specified grid area \*/

        background-color: #333; /\* Add background color for visualization \*/

        color: #fff; /\* Set text color to white \*/

        padding: 20px; /\* Add padding for spacing \*/

    }

    aside {

        grid-area: sidebar; /\* Place sidebar in the specified grid area \*/

        background-color: #444; /\* Add background color for visualization \*/

        color: #fff; /\* Set text color to white \*/

        padding: 20px; /\* Add padding for spacing \*/

    }

    main {

        grid-area: content; /\* Place main content area in the specified grid area \*/

        padding: 20px; /\* Add padding for spacing \*/

    }

    footer {

        grid-area: footer; /\* Place footer in the specified grid area \*/

        background-color: #333; /\* Add background color for visualization \*/

        color: #fff; /\* Set text color to white \*/

        padding: 20px; /\* Add padding for spacing \*/

    }

    /\* Additional styling for better visualization \*/

    body {

        font-family: Arial, sans-serif; /\* Set a common font for better readability \*/

        margin: 0; /\* Remove default margin \*/

        overflow-x: hidden; /\* Hide horizontal scrollbar \*/

    }

</style>

</head>

<body>

    <!-- HTML structure -->

    <div class="container">

        <header>

            <h1>Dashboard Analytics</h1>

        </header>

        <aside>

            <h2>Navigation</h2>

            <ul>

                <li>Dashboard</li>

                <li>Analytics</li>

                <li>Reports</li>

            </ul>

        </aside>

        <main>

            <h2>Monthly Analytics</h2>

            <p>View the latest trends and insights for the current month.</p>

        </main>

        <footer>

            <p>&copy; 2023 Dashboard Analytics. All rights reserved.</p>

        </footer>

    </div>

</body>

</html>

**CSS Rules Explanation**

**.container:**

**.container {**

**display: grid; /\* Activate CSS Grid layout for the container \*/**

**grid-template-columns: 200px 1fr; /\* Define two columns: a fixed-width sidebar and a flexible main content area \*/**

**grid-template-rows: auto 1fr auto; /\* Define three rows: auto for header and footer height, and 1fr for main content area \*/**

**grid-template-areas:**

**"header header"**

**"sidebar content"**

**"footer footer"; /\* Define grid areas for header, sidebar, content, and footer \*/**

**min-height: 100vh; /\* Set minimum height to cover the viewport \*/**

**}**

1. **display: grid;**
   * Activates the CSS Grid layout for the .container.
   * Enables the use of grid-based layout properties for arranging child elements.
   * Provides a two-dimensional layout system, allowing both rows and columns to be defined.
   * Facilitates complex layouts with precise control over the positioning of items.
   * Makes the container act as a **grid container**, managing the layout of its **direct children**.
2. **grid-template-columns: 200px 1fr;**
   * Defines the width of columns in the grid.
   * The first column is fixed at 200 pixels wide, typically used for the sidebar.
   * The second column takes up the remaining available space (1fr), typically used for the main content area.
   * The fr unit in CSS is a flexible length unit that represents a fraction of the available space in a grid container.
   * Allows for a layout where the sidebar has a consistent width while the content area adjusts to the container’s width.
   * Ensures the sidebar remains at a fixed size while the main content expands or contracts as needed.
3. **grid-template-rows: auto 1fr auto;**
   * Defines the height of rows in the grid.
   * The first and last rows (header and footer) have an automatic height based on their content.
   * The middle row (main content area) takes up the remaining available space (1fr), expanding to fill the space between the header and footer.
   * Provides a flexible layout where the header and footer height adjust to their content, while the main content area adjusts to fill the available space.
4. **grid-template-areas:**
   * Defines named areas in the grid to simplify the placement of items.
   * "header header" specifies that the header spans both columns in the first row.
   * "sidebar content" places the sidebar in the first column and the main content in the second column for the second row.
   * "footer footer" indicates that the footer spans both columns in the third row.
   * Provides a visual map of the layout, making it easier to understand and control the arrangement of grid items.
5. **min-height: 100vh;**
   * Sets the minimum height of the .container to 100% of the viewport height.
   * Ensures the container always covers the full height of the viewport, even if content is minimal.
   * Helps maintain a consistent layout structure regardless of content size.
   * Prevents the container from being shorter than the viewport height, avoiding layout issues.
   * Provides a full-height background for the container.

**header:**

**header {**

**grid-area: header; /\* Place header in the specified grid area \*/**

**background-color: #333; /\* Add background color for visualization \*/**

**color: #fff; /\* Set text color to white \*/**

**padding: 20px; /\* Add padding for spacing \*/**

**}**

1. **grid-area: header;**
   * Specifies that the header element should occupy the grid area named "header."
   * Places the header element according to the layout defined in the grid-template-areas property of the container.
   * Ensures the header spans both columns in the first row of the grid.
   * Simplifies the placement and alignment of the header within the grid layout.
   * Aligns the header element as per the grid template defined.
2. **background-color: #333;**
   * Sets the background color of the header to #333.
   * #333 is a dark gray color in hexadecimal format.
   * Provides a visually distinct background for the header, making it stand out.
   * Helps in visual differentiation from other sections of the page.
   * Ensures a consistent look and feel across different devices and browsers.
3. **color: #fff;**
   * Sets the text color within the header to #fff.
   * #fff is white in hexadecimal format.
   * Creates a high contrast between the dark background and the text, improving readability.
   * Applies to all text within the header element.
   * Enhances the visual appeal and accessibility of the header content.
4. **padding: 20px;**
   * Adds 20 pixels of padding inside the header on all sides.
   * Creates space between the header’s content and its border.
   * Prevents content from touching the edges of the header, improving readability.
   * Contributes to a well-spaced and visually balanced layout.
   * Padding is part of the element’s box model, affecting its total size.

**aside:**

**aside {**

**grid-area: sidebar; /\* Place sidebar in the specified grid area \*/**

**background-color: #444; /\* Add background color for visualization \*/**

**color: #fff; /\* Set text color to white \*/**

**padding: 20px; /\* Add padding for spacing \*/**

**}**

1. **grid-area: sidebar;**
   * Specifies that the aside element should occupy the grid area named "sidebar."
   * Places the sidebar element in the first column of the second row of the grid layout.
   * Ensures that the sidebar aligns with the grid template defined in the container.
   * Simplifies the placement and alignment of the sidebar within the grid layout.
   * Aligns the sidebar according to the layout map provided by grid-template-areas.
2. **background-color: #444;**
   * Sets the background color of the aside to #444.
   * #444 is a medium-dark gray color in hexadecimal format.
   * Provides a distinct background for the sidebar, making it visually separate from other sections.
   * Enhances visual differentiation and adds contrast to the sidebar.
   * Ensures consistency in appearance across different devices and browsers.
3. **color: #fff;**
   * Sets the text color within the aside to #fff.
   * #fff is white in hexadecimal format.
   * Provides high contrast with the dark gray background, ensuring text readability.
   * Applies to all text within the sidebar element.
   * Enhances the visual appeal and accessibility of the sidebar content.
4. **padding: 20px;**
   * Adds 20 pixels of padding inside the aside on all sides.
   * Creates space between the sidebar’s content and its border.
   * Prevents content from touching the edges of the sidebar, improving readability.
   * Contributes to a well-spaced and visually balanced layout.
   * Padding is part of the element’s box model and affects its total size.

**main:**

**main {**

**grid-area: content; /\* Place main content area in the specified grid area \*/**

**padding: 20px; /\* Add padding for spacing \*/**

**}**

1. **grid-area: content;**
   * Specifies that the main element should occupy the grid area named "content."
   * Places the main content area in the second column of the second row of the grid layout.
   * Ensures that the main content aligns with the grid template defined in the container.
   * Simplifies the placement and alignment of the main content within the grid layout.
   * Aligns the main content according to the layout map provided by grid-template-areas.
2. **padding: 20px;**
   * Adds 20 pixels of padding inside the main element on all sides.
   * Creates space between the content and the border of the main element.
   * Prevents content from touching the edges, improving readability and visual appeal.
   * Contributes to a well-spaced and visually balanced layout.
   * Padding is part of the element’s box model and affects its total size.

**footer:**

**footer {**

**grid-area: footer; /\* Place footer in the specified grid area \*/**

**background-color: #333; /\* Add background color for visualization \*/**

**color: #fff; /\* Set text color to white \*/**

**padding: 20px; /\* Add padding for spacing \*/**

**}**

1. **grid-area: footer;**
   * Specifies that the footer element should occupy the grid area named "footer."
   * Places the footer element in the third row of the grid layout, spanning both columns.
   * Ensures that the footer aligns with the grid template defined in the container.
   * Simplifies the placement and alignment of the footer within the grid layout.
   * Aligns the footer according to the layout map provided by grid-template-areas.
2. **background-color: #333;**
   * Sets the background color of the footer to #333.
   * #333 is a dark gray color in hexadecimal format.
   * Provides a distinct background for the footer, making it visually separate from other sections.
   * Enhances visual differentiation and adds contrast to the footer.
   * Ensures consistency in appearance across different devices and browsers.
3. **color: #fff;**
   * Sets the text color within the footer to #fff.
   * #fff is white in hexadecimal format.
   * Provides high contrast with the dark gray background, ensuring text readability.
   * Applies to all text within the footer element.
   * Enhances the visual appeal and accessibility of the footer content.
4. **padding: 20px;**
   * Adds 20 pixels of padding inside the footer on all sides.
   * Creates space between the footer’s content and its border.
   * Prevents content from touching the edges, improving readability and visual appeal.
   * Contributes to a well-spaced and visually balanced layout.
   * Padding is part of the element’s box model and affects its total size.

**body:**

**body {**

**font-family: Arial, sans-serif; /\* Set a common font for better readability \*/**

**margin: 0; /\* Remove default margin \*/**

**overflow-x: hidden; /\* Hide horizontal scrollbar \*/**

**}**

1. **font-family: Arial, sans-serif;**
   * Sets the font family for the entire body of the document.
   * Uses Arial as the primary font choice, with a fallback to a generic sans-serif font if Arial is not available.
   * Ensures consistent text appearance across different devices and browsers.
   * Affects all text content within the body, providing a clean and readable typeface.
2. **margin: 0;**
   * Removes any default margin around the body element.
   * Ensures there is no extra space outside the body content.
   * Provides a clean edge for the body content and sets a consistent starting point for layout adjustments.
3. **overflow-x: hidden;**
   * Hides any horizontal overflow, preventing a horizontal scrollbar from appearing.
   * Ensures that any content extending beyond the viewport width is not visible.
   * Helps maintain a clean layout and prevents layout issues related to horizontal scrolling.
   * Useful for managing content that might exceed the viewport width.

This breakdown should help you understand how each CSS rule contributes to the layout and styling of the page

**Explanation:**

* We create a grid container with **display: grid**.
* We define the grid structure using **grid-template-columns** and **grid-template-rows** to specify the size of columns and rows respectively.
* We use **grid-template-areas** to define the layout of the grid by assigning named areas for each section: header, sidebar, content, and footer.
* Each section (header, sidebar, main, footer) is given a **grid-area** property to indicate its placement in the grid.
* Inside each section, we add content such as headings, lists, paragraphs, etc., to simulate a realistic dashboard layout.

grid-template-columns

grid-template-rows

grid-template-areas

grid-area

overflow-x

fr(grid-template-columns: 200px 1fr)

**display: inline-grid**

The **display: inline-grid** property in CSS is similar to **display: grid**, but it makes the grid container behave as an inline-level element rather than a block-level element.

**Key Differences Between grid and inline-grid**

* **display: grid**: The grid container behaves as a block-level element, taking up the full width available (by default) and starting on a new line, similar to a <div>.
* **display: inline-grid**: The grid container behaves as an inline-level element, meaning it only takes up as much width as its content requires, and it can sit inline with other text or inline elements, similar to a <span> or <img>.

**Syntax**

.container {

display: inline-grid;

}

**Example: Inline Grid Layout**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Inline Grid Layout Example</title>

<style>

.container {

display: inline-grid;

grid-template-columns: auto auto; /\* 2 columns with automatic width \*/

grid-gap: 10px; /\* Gap between grid items \*/

padding: 10px;

background-color: #f0f0f0;

border: 2px solid #333;

}

.item {

background-color: #4CAF50;

color: white;

padding: 10px;

text-align: center;

}

</style>

</head>

<body>

<p>

This is an inline grid:

<span class="container">

<span class="item">Item 1</span>

<span class="item">Item 2</span>

<span class="item">Item 3</span>

<span class="item">Item 4</span>

</span>

placed within a paragraph.

</p>

</body>

</html>

**Explanation**

* **Container (.container)**:
  + display: inline-grid: The container is a grid but behaves like an inline element. It takes up only as much space as needed to fit its content, and it can be placed inline with other elements.
  + grid-template-columns: auto auto;: Defines two columns where each column's width is automatically determined by its content.
  + grid-gap: 10px;: Adds a 10px gap between the grid items.
* **Items (.item)**:
  + The grid items are laid out in a two-column grid within the inline grid container.

**Result**

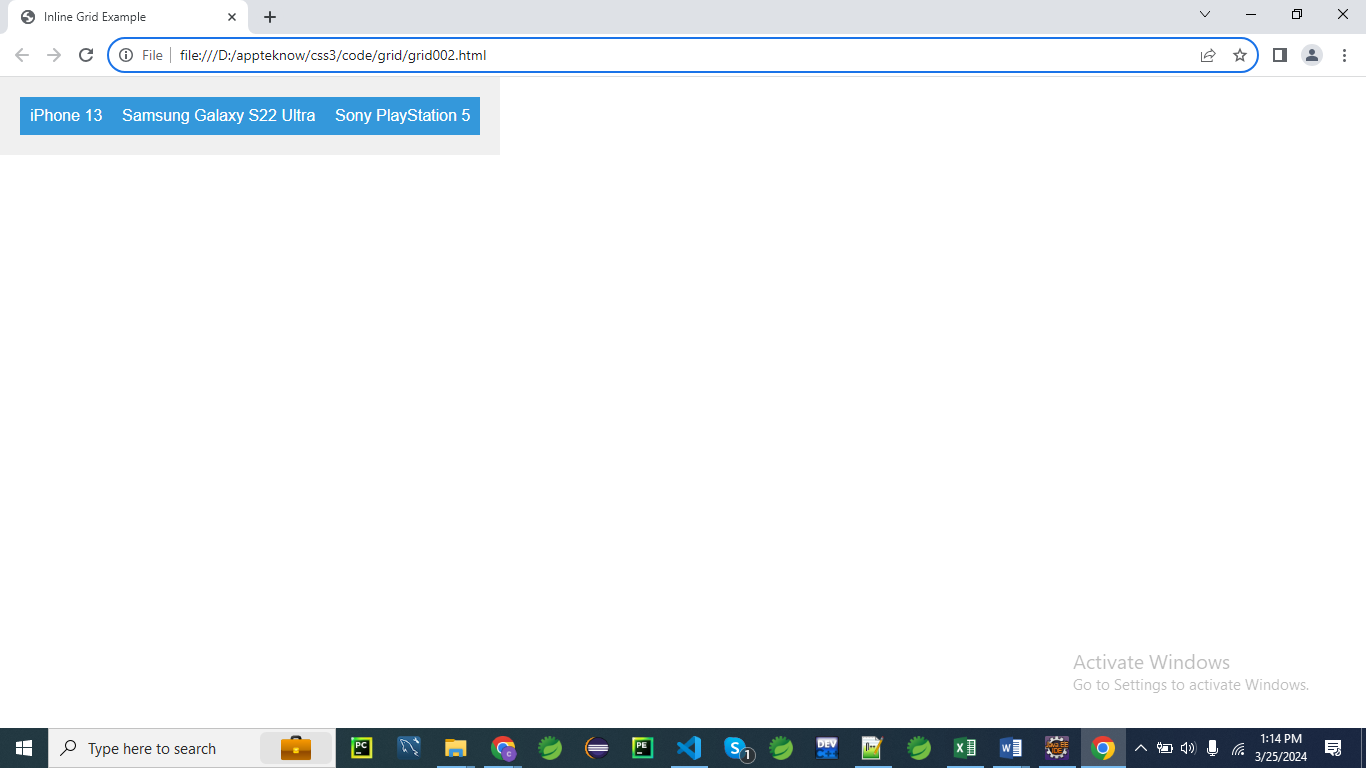
In this example, the grid container is placed within a paragraph. It behaves like an inline element, so the text before and after the grid container remains on the same line. The grid itself contains four items arranged in two columns.

**Use Cases for inline-grid**

* **Inline Layouts**: Use inline-grid when you need the layout benefits of a grid, but the grid container must remain inline with surrounding text or other inline elements.
* **Complex Inline Elements**: You can create more complex inline elements, such as buttons with icons, badges, or labels that require precise layout control.
* **Inline Component Styling**: It's useful for styling inline components like toolbars, menus, or inline cards where you need a grid layout but want them to stay inline with other content.

inline-grid offers the power of grid layouts in scenarios where you need the grid container to be inline, providing flexibility and control over complex layouts without disrupting the flow of surrounding content.

**Example 01:**



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Inline Grid Example</title>

<style>

    /\* CSS styles \*/

    .container {

        display: inline-grid; /\* Activate Inline Grid layout for the container \*/

        grid-template-columns: auto auto auto; /\* Define three auto-sized columns \*/

        background-color: #f0f0f0; /\* Add a background color for visualization \*/

        padding: 20px; /\* Add padding for spacing \*/

    }

    .item {

        padding: 10px; /\* Add padding for spacing \*/

        background-color: #3498db; /\* Set a background color for visualization \*/

        color: #fff; /\* Set text color to white for better contrast \*/

        text-align: center; /\* Center align text \*/

    }

    /\* Additional styling for better visualization \*/

    body {

        font-family: Arial, sans-serif; /\* Set a common font for better readability \*/

        margin: 0; /\* Remove default margin \*/

    }

</style>

</head>

<body>

    <!-- HTML structure -->

    <div class="container">

        <!-- Three grid items -->

        <div class="item">iPhone 13</div>

        <div class="item">Samsung Galaxy S22 Ultra</div>

        <div class="item">Sony PlayStation 5</div>

    </div>

</body>

</html>

**CSS Rules Explanation**

**.container:**

**.container {**

**display: inline-grid; /\* Activate Inline Grid layout for the container \*/**

**grid-template-columns: auto auto auto; /\* Define three auto-sized columns \*/**

**background-color: #f0f0f0; /\* Add a background color for visualization \*/**

**padding: 20px; /\* Add padding for spacing \*/**

**}**

1. **display: inline-grid;**
   * Activates the CSS Grid layout for the .container, making it an inline-level grid container.
   * Allows the container to behave like an inline element, meaning it will only take up as much width as its content requires.
   * Enables the use of grid-based layout properties for arranging child elements.
   * Provides a two-dimensional layout system for managing both rows and columns.
   * Allows the container to be positioned within inline context while still using grid layout for its children.
2. **grid-template-columns: auto auto auto;**
   * Defines the number and size of columns in the grid layout.
   * Creates three columns where each column size adjusts automatically based on the content inside.
   * Ensures that columns are sized to fit their content, without a fixed width.
   * Provides a flexible layout where column widths are determined by the content they contain.
   * Useful for layouts where column sizes need to adapt to varying content sizes.
3. **background-color: #f0f0f0;**
   * Sets the background color of the .container to #f0f0f0.
   * #f0f0f0 is a light gray color in hexadecimal format.
   * Adds a visual background to the container, making it easier to see the layout and spacing.
   * Helps differentiate the container from other page elements.
   * Ensures a consistent look and feel across different devices and browsers.
4. **padding: 20px;**
   * Adds 20 pixels of padding inside the .container on all sides.
   * Creates space between the container’s border and its content.
   * Prevents content from touching the edges of the container, improving readability.
   * Contributes to a well-spaced and visually balanced layout.
   * Padding is part of the element’s box model and affects its total size.

**.item:**

**.item {**

**padding: 10px; /\* Add padding for spacing \*/**

**background-color: #3498db; /\* Set a background color for visualization \*/**

**color: #fff; /\* Set text color to white for better contrast \*/**

**text-align: center; /\* Center align text \*/**

**}**

1. **padding: 10px;**
   * Adds 10 pixels of padding inside the .item element on all sides.
   * Creates space between the item’s border and its content.
   * Prevents content from touching the edges of the item, improving readability.
   * Contributes to a well-spaced and visually balanced layout.
   * Padding is part of the element’s box model and affects its total size.
2. **background-color: #3498db;**
   * Sets the background color of the .item to #3498db.
   * #3498db is a shade of blue in hexadecimal format.
   * Adds a visual background to each item, making them stand out within the grid.
   * Helps differentiate items from each other and from the container background.
   * Ensures consistency in appearance across different devices and browsers.
3. **color: #fff;**
   * Sets the text color within the .item to #fff.
   * #fff is white in hexadecimal format.
   * Creates high contrast with the blue background, ensuring text readability.
   * Applies to all text within the item element.
   * Enhances the visual appeal and accessibility of the content.
4. **text-align: center;**
   * Centers text horizontally within the .item element.
   * Applies to inline content such as text and inline-block elements.
   * Helps create a balanced and organized look for the content within each item.
   * Ensures that text is aligned centrally, which can be visually appealing.
   * Useful for items where central alignment of text is desired for symmetry.

**body:**

**body {**

**font-family: Arial, sans-serif; /\* Set a common font for better readability \*/**

**margin: 0; /\* Remove default margin \*/**

**}**

1. **font-family: Arial, sans-serif;**
   * Sets the font family for the entire body of the document.
   * Uses Arial as the primary font choice, with a fallback to a generic sans-serif font if Arial is not available.
   * Ensures a consistent text appearance across different devices and browsers.
   * Affects all text content within the body, providing a clean and readable typeface.
2. **margin: 0;**
   * Removes any default margin around the body element.
   * Ensures there is no extra space outside the body content.
   * Provides a clean edge for the body content and sets a consistent starting point for layout adjustments.
   * Helps in managing the layout and spacing of elements within the body.

This breakdown should help you understand how each CSS rule contributes to the layout and styling of the page

**Explanation:**

* We have a **.container** div with **display: inline-grid**. This activates the inline grid layout for the container, meaning it behaves like an inline-level element but with grid layout capabilities.
* Inside the container, we have three **.item** divs, each representing a grid item.
* We use **grid-template-columns: auto auto auto;** to define three auto-sized columns.
* Each item has some basic styling for padding and background color to visualize the grid layout.
* The body has some additional styling to set a common font and remove default margin for better consistency.

This example demonstrates how to use **display: inline-grid** to create an inline-level grid layout, which can be useful for situations where you want grid-like behavior without causing line breaks like a block-level grid would.