Lab: Introduction to CSS Grid

Objectives

By the end of this lab, you will:

- Understand how to define rows and columns using CSS Grid.
- Position items using grid-column and grid-row.
- Use gap, fr, and repeat() for flexible layouts.
- Create responsive grid layouts using auto-fit and minmax().

Example 1: Creating a Basic Grid

Goal: Introduce the display: grid property and define columns.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Grid Example 1 – Basic Grid</title>
    <style>
        .container {
            display: grid; /* enable grid layout */
            grid-template-columns: 100px 100px 100px; /* 3 fixed columns */
            gap: 10px; /* space between items */
            background: #ecf0f1;
            padding: 10px;
        }

        .item {
            background: #3498db;
            color: white;
            padding: 20px;
            text-align: center;
            font-weight: bold;
            border-radius: 6px;
        }
    </style>
</head>
<body>

    <h2>Example 1 – Basic 3-Column Grid</h2>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
        <div class="item">4</div>
        <div class="item">5</div>
        <div class="item">6</div>
    </div>

</body>
</html>
```

Explanation:

- display: grid turns .container into a grid layout.

- grid-template-columns defines 3 equal 100px columns.
- gap adds spacing between grid cells.
- Items automatically fill each cell left to right, then wrap to the next row.

Example 2: Using Flexible Columns

Goal: Use fractional units (fr) and the repeat() function for flexibility.

```
<style>
    .container {
        display: grid;
        grid-template-columns: repeat(3, 1fr); /* 3 equal columns */
        gap: 10px;
        background: #ecf0f1;
        padding: 10px;
    }
</style>
```

Explanation:

- 1fr divides the space equally among columns, regardless of screen size.
- The repeat(3, 1fr) shorthand means “repeat this column definition 3 times.”
- Resize the browser — the columns expand and shrink evenly.

Example 3: Mixing Fixed and Flexible Columns

Goal: Combine fixed and flexible columns.

```
<style>
    .container {
        display: grid;
        grid-template-columns: 200px 1fr 2fr; /* sidebar + two flexible columns */
        gap: 10px;
        background: #ecf0f1;
        padding: 10px;
    }
</style>
```

Explanation:

- First column (200px) stays fixed (e.g., sidebar).
- Second column takes up one part of remaining space.
- Third column takes up twice that amount.

Example 4: Placing Items Manually

Goal: Position items using grid-column and grid-row.

```
<style>
    .container {
        display: grid;
        grid-template-columns: repeat(3, 1fr);
        grid-template-rows: 100px 100px;
        gap: 10px;
        background: #ecf0f1;
        padding: 10px;
```

```
    }

    .item {
        background: #e67e22;
        color: white;
        display: flex;
        align-items: center;
        justify-content: center;
        border-radius: 6px;
    }

    .item1 {
        grid-column: 1 / 3;
    }
    /* spans 2 columns */
    .item4 {
        grid-row: 2 / 3;
        grid-column: 2 / 4;
    }
    /* custom placement */
</style>

<div class="container">
    <div class="item item1">1 (span 2 cols)</div>
    <div class="item item2">2</div>
    <div class="item item3">3</div>
    <div class="item item4">4 (custom position)</div>
    <div class="item item5">5</div>
    <div class="item item6">6</div>
</div>
```

Explanation:

- grid-column: 1 / 3 means start at line 1 and end before line 3 (span 2 columns).
- You can span rows and columns independently for complex layouts.

Example 5: Responsive Auto-Fitting Grid

Goal: Create a grid that adapts to screen width automatically.

Explanation:

- auto-fit automatically adds or removes columns depending on space.
- minmax(150px, 1fr) ensures each cell is at least 150px but grows as needed.
- Try resizing — the grid adapts fluidly!

```
<style>
    .container {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
        gap: 10px;
        background: #ecf0f1;
        padding: 10px;
    }

    .item {
        background: #27ae60;
```

```
        color: white;
        padding: 20px;
        text-align: center;
        border-radius: 6px;
    }
</style>

<h2>Example 5 – Responsive Grid</h2>
<div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
</div>
```

Example 6: Combining Grid Areas

Goal: Label sections for full-page layout (header, nav, main, aside, footer).

```
<style>
    body {
        margin: 0;
        font-family: Arial, sans-serif;
    }

    .container {
        display: grid;
        grid-template-areas: "header header header" "nav main aside" "footer footer
footer";
        grid-template-columns: 150px 1fr 200px;
        grid-template-rows: auto 1fr auto;
        height: 100vh;
    }

    header {
        grid-area: header;
        background: #34495e;
        color: white;
        text-align: center;
        padding: 15px;
    }

    nav {
        grid-area: nav;
        background: #2ecc71;
        padding: 15px;
    }

    main {
        grid-area: main;
        background: #ecf0f1;
        padding: 15px;
    }

    aside {
        grid-area: aside;
```

```
        background: #f1c40f;
        padding: 15px;
    }

    footer {
        grid-area: footer;
        background: #34495e;
        color: white;
        text-align: center;
        padding: 10px;
    }
</style>

<div class="container">
    <header>Header</header>
    <nav>Navigation</nav>
    <main>Main Content</main>
    <aside>Sidebar</aside>
    <footer>Footer</footer>
</div>
```

Explanation:

- grid-template-areas lets you define your layout visually with names.
- Each child uses grid-area to slot into that layout.
- Resize the window — the layout stays perfectly structured.

END