CSS Grid Layout

Section Overview

This section covers the following topics:

- CSS Grid Layout
- Basic Code Setup
- display: grid | display: inline-grid
- grid-template-columns
- grid-template-rows
- grid-template
- Grid gap properties
- justify-items
- align-items
- place-items
- justify-content
- align-content
- place-content
- grid-auto-flow
- grid-column
- grid-row
- justify-self, align-self, place-self
- grid-template-area

---

1. CSS Grid Layout

Definition

CSS Grid Layout is a two-dimensional layout system that allows web developers to create complex, responsive designs by defining rows and columns in a grid container.

Key Points

- Allows **precise placement** of items in both rows and columns.
- Uses display: grid to enable the grid layout.
- Grid **containers** define rows and columns.
- Grid **items** are placed inside the grid.
- Supports **automatic placement** and **manual positioning**.
- Works well for complex **page layouts** and **responsive designs**.

Syntax

```
.container {
    display: grid;
    grid-template-columns: 200px 200px 200px; /* Three columns */
    grid-template-rows: auto auto; /* Two rows */
}
```

Example (Real-time Scenario: Responsive Dashboard Layout)

Scenario:

A **dashboard** needs a grid-based layout with **a sidebar, a main content area, and a header**.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Grid Dashboard</title>
    <link rel="stylesheet" href="styles.css">
</head>
```

```html
<body>
    <div class="dashboard">
        <header class="header">Header</header>
        <aside class="sidebar">Sidebar</aside>
        <main class="content">Main Content</main>
    </div>
</body>
</html>
```

CSS Code:

```css
.dashboard {
    display: grid;
    grid-template-columns: 250px 1fr; /* Sidebar fixed, content expands */
    grid-template-rows: 60px 1fr; /* Header fixed, content expands */
    gap: 10px;
}

.header {
    grid-column: span 2; /* Header spans across both columns */
    background-color: darkblue;
    color: white;
    padding: 15px;
}

.sidebar {
    background-color: lightgray;
    padding: 20px;
}

.content {
    background-color: lightblue;
    padding: 20px;
}
```

Common Mistakes & Fixes

Mistake 1: Not Using `display: grid` on the Parent Container

Issue:

```css
.dashboard {
    grid-template-columns: 250px 1fr; /* Won't work without display: grid */
}
```

Fix:

```css
.dashboard {
    display: grid;
    grid-template-columns: 250px 1fr;
}
```

Mistake 2: Using `grid-template-rows` Without Defining Proper Heights

Issue:

```css
.dashboard {
    grid-template-rows: 1fr 1fr; /* May not work as expected */
}
```

Fix:

```css
.dashboard {
    grid-template-rows: 60px 1fr; /* Ensures proper spacing */
}
```

---

2. Basic Code Setup

Definition

The basic code setup for CSS Grid involves defining a **grid container** and **grid items**. The container must have display: grid, and its children will automatically become grid items.

Key Points

- display: grid enables the **grid layout**.
- Rows and columns are defined using grid-template-rows and grid-template-columns.
- gap adds spacing between grid items.
- grid-template-areas can be used for named layouts.
- The browser automatically **places grid items** unless manually positioned.

Syntax

```
.container {
   display: grid;
   grid-template-columns: 1fr 1fr 1fr; /* Three equal columns */
   grid-template-rows: auto auto; /* Two rows */
   gap: 10px;
}
```

Example (Real-time Scenario: Product Listing Grid Layout)

Scenario:

An **e-commerce website** displays product cards in a **grid format** with equal spacing.

HTML Code:

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

```
    <title>Product Grid</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="grid-container">
        <div class="product">Product 1</div>
        <div class="product">Product 2</div>
        <div class="product">Product 3</div>
        <div class="product">Product 4</div>
        <div class="product">Product 5</div>
        <div class="product">Product 6</div>
    </div>
</body>
</html>
```

CSS Code:

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr); /* Three equal columns */
    gap: 15px;
    padding: 20px;
}

.product {
    background-color: lightgray;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Not Using `display: grid` on the Parent Container

Issue:

```css
.grid-container {
    grid-template-columns: repeat(3, 1fr); /* Won't work without grid */
}
```

Fix:

```css
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
}
```

Mistake 2: Forgetting to Add Gaps Between Grid Items

Issue:

```css
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
}
```

Fix:

```css
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 15px; /* Adds spacing */
}
```

---

3. display: grid | display: inline-grid

Definition

- display: grid creates a **block-level** grid container that spans the full width of its parent.
- display: inline-grid creates an **inline-level** grid container that only takes up as much space as needed.

Key Points

- grid makes the container behave like a **block element** (takes full width).
- inline-grid makes the container behave like an **inline element** (fits content).
- Grid **children (items)** automatically become **grid elements**.
- Used for **layouts** (grid) and **small inline components** (inline-grid).

Syntax

```css
.container {
    display: grid; /* Creates a block-level grid */
}

.inline-container {
    display: inline-grid; /* Creates an inline-level grid */
}
```

Example (Real-time Scenario: Price Tag Grid with inline-grid)

Scenario:

An **e-commerce website** displays **price tags** that should align inline but still use grid properties for spacing.

HTML Code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Inline Price Tags</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="price-tags">
        <div class="price">$10</div>
        <div class="price">$20</div>
        <div class="price">$30</div>
    </div>
</body>
</html>
```

CSS Code:

```css
.price-tags {
    display: inline-grid;
    grid-template-columns: auto auto auto; /* Three price tags in a row */
    gap: 10px;
}

.price {
    background-color: lightblue;
    padding: 10px;
    text-align: center;
    border-radius: 5px;
}
```

Common Mistakes & Fixes

Mistake 1: Expecting `inline-grid` to Behave Like a Block Element

Issue:

```
.price-tags {
    display: inline-grid;
    width: 100%; /* Won't expand fully because it's inline */
}
```

Fix:

```
.price-tags {
    display: grid; /* Use grid for full width */
}
```

Mistake 2: Using grid When inline-grid is Needed

Issue:

```
.price-tags {
    display: grid; /* Takes full width when inline alignment is needed */
}
```

Fix:

```
.price-tags {
    display: inline-grid; /* Aligns inline while keeping grid behavior */
}
```

---

4. grid-template-columns

Definition

The grid-template-columns property defines the **number and size of columns** in a grid container.

Key Points

- Specifies the **width of each column**.
- Can use **fixed units (px, %, em)** or **flexible units (fr)**.
- repeat() function helps define **repeating columns**.
- auto lets columns size based on content.
- minmax(min, max) sets a **minimum and maximum** column size.

Syntax

```
.container {
    display: grid;
    grid-template-columns: 200px 1fr 2fr; /* Three columns with different sizes */
}
```

Example (Real-time Scenario: Responsive Three-Column Layout)

Scenario:

A **news website** has three sections: **Sidebar, Main Content, and Ads**. The **sidebar should be fixed**, the **main content should expand**, and the **ads should take less space**.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Three-Column Layout</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <div class="sidebar">Sidebar</div>
        <div class="main">Main Content</div>
        <div class="ads">Ads</div>
    </div>
</body>
</html>
```

CSS Code:

```
.container {
    display: grid;
    grid-template-columns: 250px 1fr 150px; /* Sidebar fixed, main expands, ads smaller
*/
    gap: 10px;
}

.sidebar, .main, .ads {
    padding: 20px;
    background-color: lightgray;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Not Defining Enough Columns for Grid Items

Issue:

```
.container {
    grid-template-columns: 200px; /* Only one column, extra items stack */
}
```

Fix:

```
.container {
    grid-template-columns: 200px 1fr 200px; /* Defines three columns */
}
```

Mistake 2: Using Fixed Width Instead of fr for Flexibility

Issue:

```
.container {
    grid-template-columns: 300px 300px 300px; /* Not responsive */
}
```

Fix:

```
.container {
    grid-template-columns: 1fr 2fr 1fr; /* Scales dynamically */
}
```

---

5. grid-template-rows

Definition

The grid-template-rows property defines the **number and size of rows** in a grid container.

Key Points

- Specifies the **height of each row**.
- Can use **fixed units (px, %, em)** or **flexible units (fr)**.
- repeat() function defines **multiple rows easily**.
- auto allows rows to **size based on content**.
- minmax(min, max) sets a **minimum and maximum** row height.

Syntax

```css
.container {
    display: grid;
    grid-template-rows: 100px 1fr 2fr; /* Three rows with different sizes */
}
```

Example (Real-time Scenario: Dashboard with Header, Content, and Footer)

Scenario:

A **dashboard layout** requires three sections: **a header, main content, and a footer**. The **header and footer should be fixed in height**, while the **main content should expand**.

HTML Code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Dashboard Layout</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="dashboard">
        <header class="header">Header</header>
        <main class="content">Main Content</main>
        <footer class="footer">Footer</footer>
    </div>
</body>
</html>
```

CSS Code:

```css
.dashboard {
    display: grid;
    grid-template-rows: 60px 1fr 50px; /* Header fixed, content expands, footer fixed
*/
    gap: 10px;
}

.header, .content, .footer {
    padding: 20px;
    text-align: center;
    background-color: lightgray;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Not Defining Enough Rows for Grid Items

Issue:

```
.dashboard {
    grid-template-rows: 100px; /* Only one row, extra items overlap */
}
```

Fix:

```
.dashboard {
    grid-template-rows: 100px 1fr 50px; /* Defines three rows */
}
```

Mistake 2: Using Fixed Height Instead of `fr` for Flexibility

Issue:

```
.dashboard {
    grid-template-rows: 300px 300px 300px; /* Not responsive */
}
```

Fix:

```
.dashboard {
    grid-template-rows: 100px 1fr 50px; /* Scales dynamically */
}
```

6. grid-template

Definition

The grid-template property is a shorthand for defining both grid-template-rows and grid-template-columns in a single declaration.

Key Points

- Combines **rows and columns** into one line of code.
- Helps keep **CSS cleaner and more readable**.
- Uses / to separate **rows from columns**.
- Can include **explicit sizes or repeat() functions**.

Syntax

```
.container {
    display: grid;
    grid-template: 100px 1fr / 200px 1fr 1fr;
    /* Rows: 100px (fixed), 1fr (flexible) */
    /* Columns: 200px (fixed), 1fr, 1fr (flexible) */
}
```

Example (Real-time Scenario: Profile Card Layout)

Scenario:

A **user profile layout** needs a **header, a sidebar, and content**, where the **header is fixed**, the **sidebar is smaller**, and the **main content takes most of the space**.

HTML Code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Profile Layout</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="profile-layout">
        <header class="header">Header</header>
        <aside class="sidebar">Sidebar</aside>
        <main class="content">Main Content</main>
    </div>
</body>
</html>
```

CSS Code:

```css
.profile-layout {
    display: grid;
    grid-template: 80px 1fr / 200px 1fr;
    /* Two rows: header (80px) and flexible content */
    /* Two columns: sidebar (200px) and main content */
    gap: 10px;
}

.header {
    grid-column: span 2; /* Header spans across both columns */
    background-color: darkblue;
    color: white;
    padding: 20px;
}

.sidebar {
    background-color: lightgray;
    padding: 20px;
}

.content {
    background-color: lightblue;
    padding: 20px;
}
```

Common Mistakes & Fixes

Mistake 1: Forgetting to Separate Rows and Columns with `/`

Issue:

```
.profile-layout {
    grid-template: 80px 1fr 200px 1fr; /* Incorrect, no separator */
}
```

Fix:

```
.profile-layout {
    grid-template: 80px 1fr / 200px 1fr; /* Correct, using `/` */
}
```

Mistake 2: Expecting `grid-template` to Work Without `display: grid`

Issue:

```
.profile-layout {
    grid-template: 80px 1fr / 200px 1fr; /* Won't work without display: grid */
}
```

Fix:

```
.profile-layout {
    display: grid;
    grid-template: 80px 1fr / 200px 1fr;
}
```

---

7. Grid Gap Properties

Definition

The gap property (formerly grid-gap) defines the **spacing between grid items**, making it easier to manage layout spacing without using margins.

Key Points

- gap applies space **between** rows and columns.
- row-gap controls **vertical spacing**.
- column-gap controls **horizontal spacing**.
- Works **only inside grid containers** (display: grid).
- **More efficient** than using margins on grid items.

Syntax

```
.container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: repeat(2, auto);
    gap: 20px; /* Applies equal spacing between rows and columns */
}
```

Example (Real-time Scenario: Photo Gallery with Spacing)

Scenario:

A **photo gallery** layout needs **even spacing between images**, without using margins or affecting alignment.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Photo Gallery</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="gallery">
        <div class="photo">1</div>
        <div class="photo">2</div>
        <div class="photo">3</div>
        <div class="photo">4</div>
        <div class="photo">5</div>
        <div class="photo">6</div>
    </div>
</body>
</html>
```

CSS Code:

.

```
gallery {
    display: grid;
    grid-template-columns: repeat(3, 1fr); /* 3 columns */
    grid-template-rows: auto auto; /* 2 rows */
    gap: 15px; /* Even spacing between photos */
    padding: 20px;
}

.photo {
    background-color: lightgray;
    padding: 50px;
    text-align: center;
    font-size: 20px;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Using Margins Instead of `gap` for Grid Items

Issue:

```
.photo {
    margin: 15px; /* Uneven spacing due to margin collapsing */
}
```

Fix:

```
.gallery {
    gap: 15px; /* Ensures even spacing inside the grid */
}
```

Mistake 2: Using `gap` Without `display: grid`

Issue:

```css
.gallery {
    gap: 15px; /* Won't work because grid is missing */
}
```

Fix:

```css
.gallery {
    display: grid;
    gap: 15px;
}
```

---

8. justify-items

Definition

The justify-items property controls how **grid items** are aligned **horizontally** within their grid cells.

Key Points

- Aligns **individual items** inside their **assigned grid cells**.
- Works only on **grid containers** (display: grid).
- Affects **all grid items** unless overridden by justify-self.
- Common values:
    - start (aligns items to the **left** of the cell).
    - center (aligns items in the **middle**).
    - end (aligns items to the **right**).
    - stretch (default, items **expand to fill** the cell).

Syntax

```
.container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    justify-items: center; /* Aligns all items in the center of each cell */
}
```

Example (Real-time Scenario: Centering Text in a Grid Layout)

Scenario:

A **dashboard** has **three columns**, and the **text inside each column** should be **centered horizontally** within its grid cell.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Centered Grid Items</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="dashboard">
        <div class="item">Profile</div>
        <div class="item">Settings</div>
        <div class="item">Logout</div>
    </div>
</body>
</html>
```

CSS Code:

```
.dashboard {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    justify-items: center; /* Centers all items horizontally */
```

```css
    gap: 10px;
    padding: 20px;
}

.item {
    background-color: lightblue;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Expecting `justify-items` to Work on a Non-Grid Container

Issue:

```css
.dashboard {
    justify-items: center; /* Won't work because display: grid is missing */
}
```

Fix:

```css
.dashboard {
    display: grid;
    justify-items: center;
}
```

Mistake 2: Using `justify-items` Instead of `justify-content` for Entire Grid Alignment

Issue:

```css
.dashboard {
    justify-items: center; /* Won't align the entire grid */
}
```

Fix:

```css
.dashboard {
    justify-content: center; /* Centers the entire grid */
}
```

---

9. align-items

Definition

The align-items property controls how **grid items** are aligned **vertically** inside their grid cells.

Key Points

- Works on **grid containers** (display: grid).
- Affects **all grid items** inside their respective cells.
- Controls **vertical alignment** along the **cross-axis**.
- Common values:
    - start (aligns items to the **top** of the cell).
    - center (aligns items in the **middle** vertically).
    - end (aligns items to the **bottom**).
    - stretch (**default**, items expand to fill the cell).

Syntax

```css
.container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    align-items: center; /* Aligns all items to the center of their cells */
}
```

Example (Real-time Scenario: Centering Items Vertically in a Feature Section)

Scenario:

A **feature section** contains **three boxes**, and their content should be **vertically centered** within each grid cell.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Vertically Centered Grid Items</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="feature-section">
        <div class="feature">Feature 1</div>
        <div class="feature">Feature 2</div>
        <div class="feature">Feature 3</div>
    </div>
</body>
</html>
```

CSS Code:

```
.feature-section {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    align-items: center; /* Centers items vertically */
    height: 300px; /* Adds height for visible vertical alignment */
    gap: 10px;
}

.feature {
    background-color: lightblue;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Expecting `align-items` to Work Without `display: grid`

Issue:

```
.feature-section {
    align-items: center; /* Won't work because grid is missing */
}
```

Fix:

```
.feature-section {
    display: grid;
    align-items: center;
}
```

Mistake 2: Using `align-items` Instead of `align-content` for Grid Alignment

Issue:

```css
.feature-section {
    align-items: center; /* Doesn't move the whole grid */
}
```

Fix:

```css
.feature-section {
    align-content: center; /* Centers the entire grid */
}
```

---

10. place-items

Definition

The place-items property is a shorthand for **aligning grid items both horizontally (justify-items) and vertically (align-items)** in a single declaration.

Key Points

- Combines justify-items (horizontal alignment) and align-items (vertical alignment).
- Works only inside **grid containers** (display: grid).
- Uses values like:
    - start (aligns items to the **top-left**).
    - center (aligns items to the **middle** both ways).
    - end (aligns items to the **bottom-right**).
    - stretch (**default**, items fill available space).

Syntax

```css
.container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: repeat(2, 100px);
    place-items: center; /* Centers all items horizontally & vertically */
}
```

Example (Real-time Scenario: Centering Cards in a Grid)

Scenario:

A **dashboard contains multiple cards**, and they should be **centered inside their grid cells both vertically and horizontally**.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Centered Grid Cards</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="card-grid">
        <div class="card">Card 1</div>
        <div class="card">Card 2</div>
        <div class="card">Card 3</div>
        <div class="card">Card 4</div>
    </div>
</body>
</html>
```

CSS Code:

```
.card-grid {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
    grid-template-rows: repeat(2, 150px);
    place-items: center; /* Centers items horizontally & vertically */
    gap: 10px;
}

.card {
    background-color: lightgray;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Using place-items on a Non-Grid Container

Issue:

```
.card-grid {
    place-items: center; /* Won't work because display: grid is missing */
}
```

Fix:

```
.card-grid {
    display: grid;
    place-items: center;
}
```

Mistake 2: Using place-items Instead of place-content for Entire Grid Alignment

Issue:

```
.card-grid {
    place-items: center; /* Doesn't move the whole grid */
}
```

Fix:

```
.card-grid {
    place-content: center; /* Centers the entire grid */
}
```

---

11. justify-content

Definition

The justify-content property controls the **horizontal alignment** of the entire grid inside its container.

Key Points

- Works only on **grid containers** (display: grid).
- Affects how the **entire grid** is positioned within the container.
- Common values:
    - start (aligns the grid to the **left**).
    - center (aligns the grid in the **middle**).
    - end (aligns the grid to the **right**).
    - space-between (even spacing between grid columns).
    - space-around (equal spacing **around** grid columns).
    - space-evenly (equal spacing **inside and outside** columns).

Syntax

```
.container {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    justify-content: center; /* Centers the entire grid horizontally */
}
```

Example (Real-time Scenario: Centering a Navigation Bar Grid)

Scenario:

A **navigation menu** is displayed using **grid layout**, and the entire menu should be **centered horizontally** within the header.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Centered Navigation Grid</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header class="navbar">
        <nav class="nav-menu">
            <div class="nav-item">Home</div>
            <div class="nav-item">About</div>
            <div class="nav-item">Services</div>
            <div class="nav-item">Contact</div>
        </nav>
    </header>
</body>
</html>
```

CSS Code:

```
.nav-menu {
    display: grid;
    grid-template-columns: repeat(4, auto);
    justify-content: center; /* Centers the entire menu */
    gap: 15px;
}

.nav-item {
    background-color: lightblue;
    padding: 10px 15px;
    text-align: center;
    border-radius: 5px;
}
```

Common Mistakes & Fixes

Mistake 1: Using `justify-content` Without `display: grid`

Issue:

```
.nav-menu {
    justify-content: center; /* Won't work because display: grid is missing */
}
```

Fix:

```
.nav-menu {
    display: grid;
    justify-content: center;
}
```

Mistake 2: Expecting `justify-content` to Align Individual Items

Issue:

```
.nav-item {
   justify-content: center; /* Won't align individual items */
}
```

Fix:

```
.nav-menu {
   justify-content: center; /* Aligns the entire grid */
}

.nav-item {
   text-align: center; /* Centers text inside items */
}
```

---

12. align-content

Definition

The align-content property controls the **vertical alignment** of the entire grid inside its container. It only works when there is **extra space** in the container.

Key Points

- Works only on **grid containers** (display: grid).
- Affects the **entire grid's vertical alignment**, not individual items.
- Works when the grid **does not take up the full height** of the container.
- Common values:
  - start (aligns the grid to the **top**).
  - center (aligns the grid in the **middle**).
  - end (aligns the grid to the **bottom**).
  - space-between (even spacing between grid rows).

    - space-around (equal spacing **around** grid rows).
    - space-evenly (equal spacing **inside and outside** rows).

Syntax

```
.container {
   display: grid;
   grid-template-rows: repeat(3, 100px);
   height: 500px; /* Extra height to see the effect */
   align-content: center; /* Centers the grid vertically */
}
```

Example (Real-time Scenario: Centering a Pricing Table Grid Vertically)

Scenario:

A **pricing table** layout needs to be **vertically centered** inside a section with extra height.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Centered Pricing Table</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <section class="pricing-section">
      <div class="pricing-table">
         <div class="plan">Basic Plan</div>
         <div class="plan">Standard Plan</div>
         <div class="plan">Premium Plan</div>
      </div>
   </section>
```

```
</body>
</html>
```

CSS Code:

```
.pricing-section {
    height: 500px; /* Extra space to demonstrate vertical centering */
    display: flex;
    justify-content: center;
}

.pricing-table {
    display: grid;
    grid-template-rows: repeat(3, 100px);
    align-content: center; /* Centers the grid vertically */
    gap: 15px;
}

.plan {
    background-color: lightgray;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Using `align-content` When There's No Extra Space

Issue:

```
.pricing-table {
    align-content: center; /* Won't work if grid fills the container */
}
```

Fix:

```css
.pricing-section {
    height: 500px; /* Extra height allows align-content to work */
}
```

Mistake 2: Expecting align-content to Center Individual Items

Issue:

```css
.plan {
    align-content: center; /* Won't work, should use align-items */
}
```

Fix:

```css
.pricing-table {
    align-items: center; /* Aligns individual grid items */
}
```

---

13. place-content

Definition

The place-content property is a shorthand for **aligning the entire grid both vertically (align-content) and horizontally (justify-content)** in a single declaration.

Key Points

- Combines align-content and justify-content into a single property.
- Works **only on grid containers** (display: grid).
- Affects the **entire grid**, not individual items.

- Useful for centering the **whole grid** inside its container.
- Common values:
  - start (aligns grid to **top-left**).
  - center (aligns grid **in the middle** both ways).
  - end (aligns grid to **bottom-right**).
  - space-between (evenly distributes grid rows and columns).
  - space-around (equal spacing **around** grid elements).
  - space-evenly (equal spacing **inside and outside** the grid).

Syntax

```
.container {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-template-rows: repeat(2, 100px);
    height: 500px;
    width: 500px;
    place-content: center; /* Centers the entire grid both horizontally & vertically */
}
```

Example (Real-time Scenario: Centering a Card Layout in a Section)

Scenario:

A **dashboard contains multiple cards**, and the **entire grid should be centered** inside a section.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Centered Card Grid</title>
    <link rel="stylesheet" href="styles.css">
```

```html
</head>
<body>
    <section class="dashboard">
        <div class="card-grid">
            <div class="card">Card 1</div>
            <div class="card">Card 2</div>
            <div class="card">Card 3</div>
            <div class="card">Card 4</div>
        </div>
    </section>
</body>
</html>
```

CSS Code:

```css
.dashboard {
    height: 500px; /* Extra space for vertical alignment */
    display: flex;
    justify-content: center;
}

.card-grid {
    display: grid;
    grid-template-columns: repeat(2, 150px);
    grid-template-rows: repeat(2, 150px);
    place-content: center; /* Centers the entire grid */
    gap: 15px;
}

.card {
    background-color: lightgray;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Using place-content on Non-Grid Elements

Issue:

```
.card-grid {
    place-content: center; /* Won't work because grid is missing */
}
```

Fix:

```
.card-grid {
    display: grid;
    place-content: center;
}
```

Mistake 2: Using place-content Instead of place-items for Individual Item Alignment

Issue:

```
.card {
    place-content: center; /* Won't work, needs place-items */
}
```

Fix:

```
.card-grid {
    place-items: center; /* Aligns individual grid items */
}
```

14. grid-auto-flow

Definition

The grid-auto-flow property controls the **automatic placement** of grid items when they are not explicitly assigned to specific rows or columns.

Key Points

- Defines how **extra grid items** are placed when no position is specified.
- Works **only on grid containers** (display: grid).
- Common values:
    - row (default) – items are placed **row by row**.
    - column – items are placed **column by column**.
    - dense – fills gaps by **rearranging smaller items** (may change item order).

Syntax

```
.container {
   display: grid;
   grid-template-columns: repeat(3, 100px);
   grid-auto-flow: column; /* Automatically places items column by column */
}
```

Example (Real-time Scenario: Auto-Arranging a List of Items)

Scenario:

A **dashboard has a list of widgets**, but some widgets are not assigned specific grid positions, so they should **auto-place in columns**.

HTML Code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Auto-Placed Grid Items</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="dashboard">
        <div class="widget">Widget 1</div>
        <div class="widget">Widget 2</div>
        <div class="widget">Widget 3</div>
        <div class="widget">Widget 4</div>
        <div class="widget">Widget 5</div>
    </div>
</body>
</html>
```

CSS Code:

```css
.dashboard {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-auto-flow: column; /* Places extra items in columns */
    gap: 10px;
}

.widget {
    background-color: lightblue;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
}
```

Common Mistakes & Fixes

Mistake 1: Using `grid-auto-flow: column` Without Enough Columns

Issue:

```css
.dashboard {
    grid-template-columns: 100px; /* Only one column */
    grid-auto-flow: column;
}
```

Fix:

```css
.dashboard {
    grid-template-columns: repeat(3, 100px); /* Multiple columns for placement */
    grid-auto-flow: column;
}
```

Mistake 2: Expecting `dense` to Maintain Item Order

Issue:

```css
.dashboard {
    grid-auto-flow: dense; /* May change the order of items */
}
```

Fix:

```css
.dashboard {
    grid-auto-flow: row; /* Keeps items in order */
}
```

---

15. `grid-column`

Definition

The grid-column property controls **how many columns** a grid item spans inside a grid container.

Key Points

- Works only inside **grid containers** (display: grid).
- Can specify **starting and ending** column positions.
- Shorthand for grid-column-start and grid-column-end.
- Common values:
  - grid-column: 1 / 3; (starts at **column 1**, ends at **column 3**).
  - grid-column: span 2; (spans across **2 columns**).

Syntax

```
.item {
    grid-column: 1 / 3; /* Starts at column 1, ends at column 3 */
}
```

Example (Real-time Scenario: Grid Layout with Featured Content Spanning Multiple Columns)

Scenario:

A **blog layout** has a **featured article** that should span **across two columns**, while other articles take **one column** each.

HTML Code:

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

```html
    <title>Featured Blog Layout</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="blog-grid">
        <div class="featured">Featured Article</div>
        <div class="article">Article 1</div>
        <div class="article">Article 2</div>
    </div>
</body>
</html>
```

CSS Code:

```css
.blog-grid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 10px;
}

.featured {
    grid-column: span 2; /* Featured article spans across two columns */
    background-color: lightblue;
    padding: 20px;
    text-align: center;
}

.article {
    background-color: lightgray;
    padding: 20px;
    text-align: center;
}
```

Common Mistakes & Fixes

Mistake 1: Using grid-column Without Enough Columns in the Grid

Issue:

```css
.featured {
    grid-column: span 3; /* Won't work if there are only 2 columns */
}
```

Fix:

```css
.blog-grid {
    grid-template-columns: repeat(3, 1fr); /* Ensure enough columns exist */
}
```

Mistake 2: Expecting grid-column to Work Outside a Grid Container

Issue:

```css
.featured {
    grid-column: span 2; /* Won't work because it's not inside a grid */
}
```

Fix:

```css
.blog-grid {
    display: grid;
}

.featured {
    grid-column: span 2;
}
```

16. grid-row

Definition

The grid-row property controls **how many rows** a grid item spans inside a grid container.

Key Points

- Works only inside **grid containers** (display: grid).
- Can specify **starting and ending** row positions.
- Shorthand for grid-row-start and grid-row-end.
- Common values:
  - grid-row: 1 / 3; (starts at **row 1**, ends at **row 3**).
  - grid-row: span 2; (spans across **2 rows**).

Syntax

```
.item {
    grid-row: 1 / 3; /* Starts at row 1, ends at row 3 */
}
```

Example (Real-time Scenario: Grid Layout with Sidebar Spanning Multiple Rows)

Scenario:

A **dashboard layout** has a **sidebar** that should span **across two rows**, while the main content and widgets take **separate rows**.

HTML Code:

```
<!DOCTYPE html>
```

```html
<html>
<head>
    <title>Grid with Row Spanning</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="dashboard">
        <div class="sidebar">Sidebar</div>
        <div class="header">Header</div>
        <div class="content">Main Content</div>
    </div>
</body>
</html>
```

CSS Code:

```css
.dashboard {
    display: grid;
    grid-template-columns: 200px 1fr;
    grid-template-rows: 100px 1fr;
    gap: 10px;
}

.sidebar {
    grid-row: span 2; /* Sidebar spans across two rows */
    background-color: lightgray;
    padding: 20px;
    text-align: center;
}

.header {
    background-color: lightblue;
    padding: 20px;
    text-align: center;
}
```

```
.content {
    background-color: lightcoral;
    padding: 20px;
    text-align: center;
}
```

Common Mistakes & Fixes

Mistake 1: Using `grid-row` Without Enough Rows in the Grid

Issue:

```
.sidebar {
    grid-row: span 3; /* Won't work if there are only 2 rows */
}
```

Fix:

```
.dashboard {
    grid-template-rows: 100px 1fr 100px; /* Ensure enough rows exist */
}
```

Mistake 2: Expecting `grid-row` to Work Outside a Grid Container

Issue:

```
.sidebar {
    grid-row: span 2; /* Won't work because it's not inside a grid */
}
```

Fix:

```
.dashboard {
    display: grid;
}

.sidebar {
    grid-row: span 2;
}
```

---

17. justify-self, align-self, place-self

Definition

These properties control **how individual grid items are positioned within their assigned grid cells**:

- justify-self → Aligns the item **horizontally** within its cell.
- align-self → Aligns the item **vertically** within its cell.
- place-self → A shorthand for align-self and justify-self.

Key Points

- Works only on **grid items** (not the container).
- Overrides the container's justify-items and align-items.
- Common values:
    - start → Aligns item to the **top-left** of its cell.
    - center → Centers the item inside the cell.
    - end → Aligns item to the **bottom-right**.
    - stretch (**default**) → Makes the item **fill the cell**.

Syntax

```
.item1 {
    justify-self: center; /* Centers item horizontally */
    align-self: end; /* Aligns item to the bottom */
}

.item2 {
    place-self: center; /* Centers item both horizontally & vertically */
}
```

Example (Real-time Scenario: Aligning Grid Buttons Inside Cells)

Scenario:

A **grid-based button layout** requires each button to be **aligned differently inside its cell**.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Self Alignment in Grid</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="button-grid">
        <button class="btn btn-start">Start</button>
        <button class="btn btn-center">Center</button>
        <button class="btn btn-end">End</button>
    </div>
</body>
</html>
```

CSS Code:

```
.button-grid {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-template-rows: 100px;
    gap: 10px;
}

.btn {
    padding: 10px 20px;
    background-color: blue;
    color: white;
    border: none;
    cursor: pointer;
}

.btn-start {
    justify-self: start; /* Aligns button to the left of its cell */
}

.btn-center {
    justify-self: center; /* Centers button in its cell */
}

.btn-end {
    justify-self: end; /* Aligns button to the right of its cell */
}
```

Common Mistakes & Fixes

Mistake 1: Using `justify-self` or `align-self` on a Non-Grid Item

Issue:

```
.btn {
    justify-self: center; /* Won't work because the parent is not grid */
}
```

Fix:

```
.button-grid {
    display: grid;
}

.btn {
    justify-self: center;
}
```

Mistake 2: Using `justify-self` Instead of `justify-items` When Targeting All Items

Issue:

```
.button-grid {
    justify-self: center; /* Won't work because it's applied to the container */
}
```

Fix:

```
.button-grid {
    justify-items: center; /* Aligns all buttons inside the grid */
}
```

---

18. `grid-template-areas`

Definition

The `grid-template-areas` property allows **naming grid sections** and **positioning elements using names instead of numbers**, making grid layouts more readable and easier to manage.

Key Points

- Works only on **grid containers** (display: grid).
- Uses **named areas** instead of numerical row/column positions.
- Requires grid-area on grid items to assign them to named sections.
- Helps define **complex layouts** clearly.

Syntax

```css
.container {
    display: grid;
    grid-template-areas:
        "header header"
        "sidebar main"
        "footer footer";
    grid-template-columns: 200px 1fr;
    grid-template-rows: 60px 1fr 50px;
}

.header {
    grid-area: header;
}

.sidebar {
    grid-area: sidebar;
}

.main {
    grid-area: main;
}

.footer {
    grid-area: footer;
}
```

Example (Real-time Scenario: Page Layout with Named Grid Areas)

Scenario:

A **website layout** includes a **header, sidebar, main content, and footer**, which should be arranged using **grid-template-areas** for better readability.

HTML Code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Grid Template Areas</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="page-layout">
        <header class="header">Header</header>
        <aside class="sidebar">Sidebar</aside>
        <main class="main">Main Content</main>
        <footer class="footer">Footer</footer>
    </div>
</body>
</html>
```

CSS Code:

```css
.page-layout {
    display: grid;
    grid-template-areas:
        "header header"
        "sidebar main"
        "footer footer";
    grid-template-columns: 200px 1fr;
    grid-template-rows: 60px 1fr 50px;
    gap: 10px;
}

.header {
    grid-area: header;
    background-color: darkblue;
    color: white;
    padding: 20px;
}

.sidebar {
    grid-area: sidebar;
    background-color: lightgray;
    padding: 20px;
}

.main {
    grid-area: main;
    background-color: lightblue;
    padding: 20px;
}

.footer {
    grid-area: footer;
    background-color: darkgray;
    padding: 20px;
    text-align: center;
}
```

Common Mistakes & Fixes

Mistake 1: Forgetting to Assign `grid-area` to Grid Items

Issue:

```
.page-layout {
    grid-template-areas: "header header" "sidebar main" "footer footer";
}
```

Fix:

```
.header {
    grid-area: header;
}

.sidebar {
    grid-area: sidebar;
}

.main {
    grid-area: main;
}

.footer {
    grid-area: footer;
}
```

Mistake 2: Using grid-template-areas with an Incorrect Layout Structure

Issue:

```
.page-layout {
    grid-template-areas:
        "header"
        "sidebar main"
        "footer";
} /* Missing second column for sidebar/main */
```

Fix:

```css
.page-layout {
    grid-template-areas:
        "header header"
        "sidebar main"
        "footer footer";
}
```

This **completes the CSS Grid Layout section!** 🎉