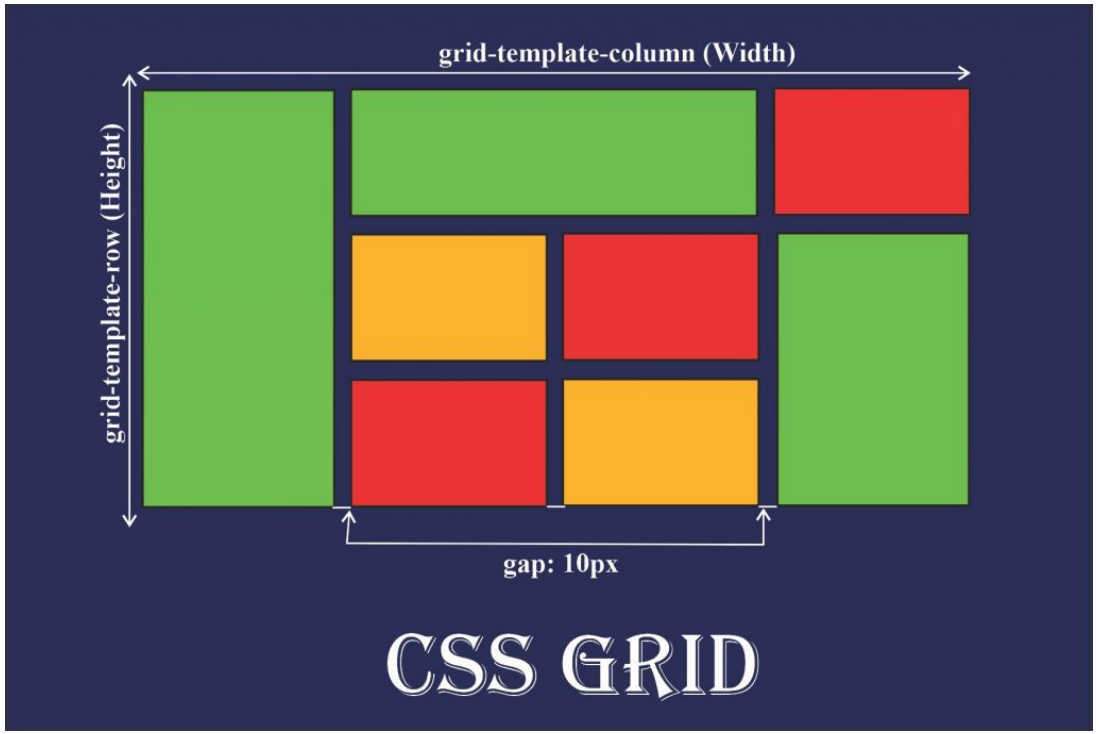
CSS Grid

Introduction to CSS Grid

What is CSS Grid?

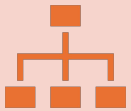


- **CSS Grid** is a two-dimensional layout system for the web, allowing you to design both **rows** and **columns** simultaneously.
- It provides a powerful way to create complex and responsive layouts.
- **CSS Grid** gives you full control over **rows**, **columns**, and **gaps** between elements in a grid, which can be tricky to achieve with traditional layout methods like floats.

Why Use CSS Grid?

Enables creating **complex layouts** without needing floats or positioning.

Makes it easier to build **responsive designs** by adjusting the grid for different screen sizes.



Provides better control over **alignment** and **spacing** of elements.

Basic CSS Grid Structure

```
.container {
    display: grid;
    grid-template-columns:
repeat(3, 1fr);
    grid-gap: 20px;
  }
```

Basic CSS Grid Container

To create a grid, you need to define a container with **display: grid**.

The items inside this container will automatically become grid items.

Defining Rows and Columns

```
.container {
    display: grid;

    grid-template-columns: 1fr 2fr;
/* 1 fraction for the first column, 2
fractions for the second column */

    grid-template-rows: auto 100px;
/* Auto for the first row, 100px for the
second row */
  }
```

Creating Grid with Columns and Rows

grid-template-columns: Defines the number and size of columns.

grid-template-rows: Defines the number and size of rows.

You can set sizes in **px**, **%,** or **fr** (fractional units).

Grid Gaps

```
 .container {
    display: grid;

    grid-template-columns: repeat(3,
1fr);
    grid-gap: 20px;
/* Adds 20px gap between both rows
and columns */
  }
```

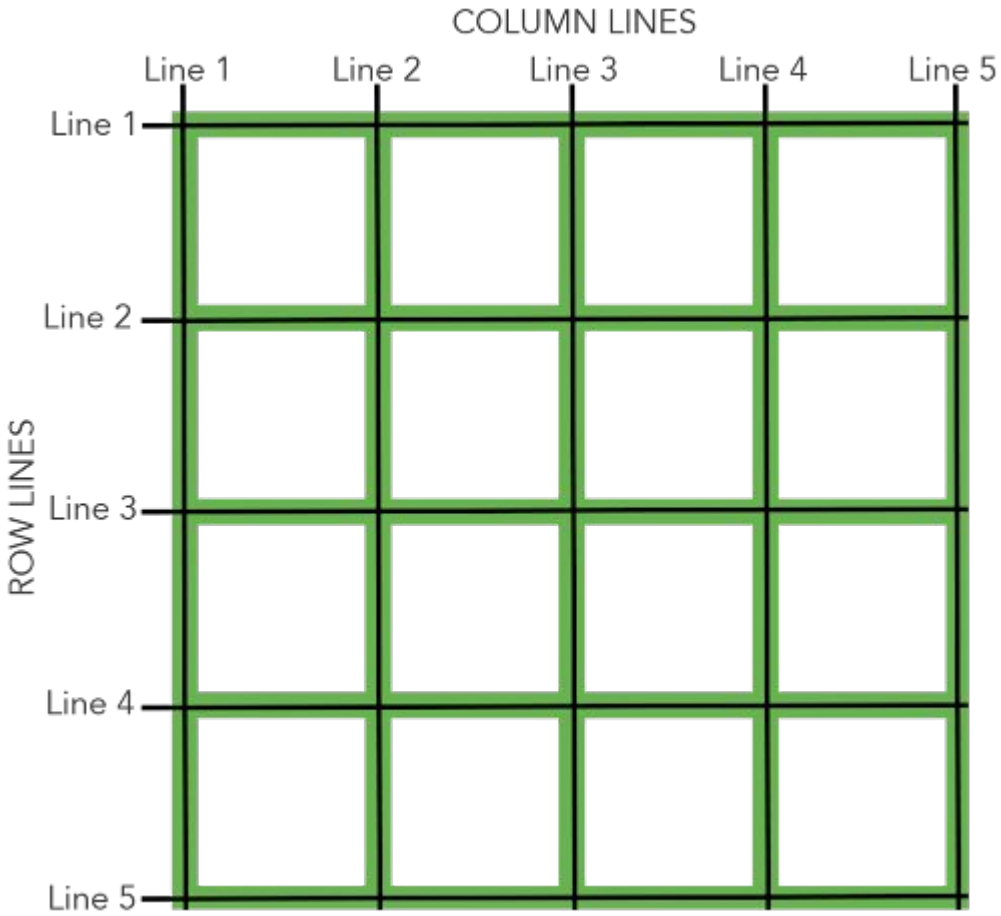
Setting Gaps Between Grid Items

grid-gap: Defines the space between grid items, both vertically and horizontally.

You can set it separately for rows and columns.

grid-row-gap and grid-column-gap can be used to set individual row or column gaps.

Grid Lines, Tracks, and Cells



Understanding Grid Lines and Tracks

Grid lines: These are the boundaries that define the edges of your grid rows and columns.

- **grid-template-columns** defines columns (vertical lines).
- **grid-template-rows** defines rows (horizontal lines).

Grid Tracks: The space between two adjacent grid lines (a row or column).

Grid Cells: The individual areas where grid items are placed.

Placing Grid Items

```
.item1 {

    grid-column: 1 / 3;
/* Spans from column 1 to column 3 */

    grid-row: 1 / 2;
/* Spans from row 1 to row 2 */

  }
```

Positioning Grid Items

You can place grid items on specific rows and columns using **grid-column** and **grid-row** properties.

grid-column: Defines the starting and ending points for columns.grid-row: Defines the starting and ending points for rows.

Grid Item Sizing

```
.container {
    display: grid;

    grid-template-columns: 1fr 1fr;

    grid-auto-rows: 200px;

 /* Items beyond the defined grid
will be placed in new rows with a
height of 200px */

  }
```

Explicit and Implicit Grid Sizing

Explicit sizing: You define the grid structure using **grid-template-columns** and **grid-template-rows**.

Implicit sizing: Automatically generates rows or columns for grid items that overflow the defined grid.

Class Activity 3: CSS Grid Practice - Build a Responsive Layout

Objective:

- The goal of this activity is to practice and reinforce your understanding of **CSS Grid** by building a simple, responsive webpage layout. You will work with **grid containers**, **grid items**, and **media queries** to create a layout that adjusts based on screen size.

Class Activity 4: Grid Attack - CSS Grid Challenges

Objective:

The goal of this activity is to practice and reinforce your understanding of CSS Grid by completing all the challenges on the Grid Garden website.

Instructions:

- Visit the Grid Attack Website:
  - Go to the following URL: https://codingfantasy.com/games/css-grid-attack/play
  - This website presents a series of 80 interactive challenges that will help you understand and use various CSS Grid properties to help the plants grow in the garden.
- Complete At least 25 Challenges:
  - Start from the first challenge and complete each one in sequence. Each challenge introduces new CSS Grid concepts and guides you through solving the problems.

Introduction to Media Queries

Making websites fully responsive

What is a Media Query?

A **Media Query** is a CSS technique used to apply styles based on the **device's characteristics** such as **screen width**, **height**, **resolution**, and **orientation**.

- Media queries are a fundamental tool in **responsive web design**, enabling developers to create layouts that adapt to different screen sizes and devices.

Why Use Media Queries?

- **Adapt Layouts**: Media queries allow a webpage to adjust its layout for different screen sizes (desktop, tablet, mobile).
- **Improve User Experience**: Ensures that content is readable and usable on all devices.
- **Create Responsive Designs**: Helps implement flexible designs without using fixed-width layouts.

Media Query Syntax

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

    /* Styles will be applied for
screens 768px or smaller */
    body {
      background-color: lightblue;
    }
  }
```

Basic Syntax of Media Queries

@media: The keyword that defines a media query.

Condition: The feature to query, such as width, height, orientation, etc.

CSS styles: The styles that apply when the condition is true.

Media Query Features

```
@media (max-width: 600px) and
(orientation: portrait) {
    body {
      font-size: 14px;
    }
  }
```

Common Media Features:
width: The width of the viewport.
height: The height of the viewport.
orientation: The orientation of the device (landscape or portrait).
resolution: The screen resolution (useful for high-DPI displays like Retina screens).
aspect-ratio: The ratio of the viewport's width to its height.

Media Query Types

```
/* For devices with a width of 600px
or more */
@media (min-width: 600px) {
    body {
      background-color: lightgreen;
    }
  }

  /* For devices with a width of 600px
or less */
  @media (max-width: 600px) {
    body {
      background-color: lightblue;
    }
  }
```

min-width and max-width

min-width: Applies styles when the viewport is **greater than** the specified width.

max-width: Applies styles when the viewport is **smaller than** the specified width.

Orientation

```
/* For portrait mode */
@media (orientation: portrait) {
    body {
      font-size: 16px;
    }
  }

  /* For landscape mode */
  @media (orientation: landscape) {
    body {
      font-size: 18px;
    }
  }
```

orientation

portrait: Applies styles for devices with a **vertical** screen orientation.

landscape: Applies styles for devices with a **horizontal** screen orientation.

Responsive Grid Layouts

```
.container {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-gap: 20px;
  }

  @media (max-width: 768px) {
    .container {
      grid-template-columns: 1fr 1fr;
 /* Stack items in 2 columns on small
screens */
    }
  }
```

Use **media queries** to adjust the grid layout for different screen sizes.

grid-template-columns and **grid-template-rows** can be adjusted inside media queries to make the grid layout more responsive.

Lab Activity 5: Build a Personal Portfolio Website

Objective:

- The goal of this activity is to create a **Personal Portfolio Website** using **HTML**, **CSS**, **Flexbox**, and **CSS Grid**. You will apply everything you've learned so far to design and structure a professional-looking webpage. This will allow you to showcase your skills and knowledge of web development and responsive design.