University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development

Cascading Style Sheets (CSS)

12 mars 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

What is CSS ?

CSS : **C**ascading **S**tyle **S**heets

- A style sheet language used to describe the presentation (appearance) of documents written in HTML or XML

  Describes *how* information is to be displayed, not *what* is being displayed

- Can be embedded in HTML document or placed into separate **.css** file

Why CSS ?

- CSS separates content from presentation
- It allows for consistent styling across multiple pages of a website
- CSS simplifies the process of updating the look and feel of a website

Basic CSS rule syntax

```
selector {
  property: value;

  ...

  property: value;
}
```

The selector can either be a grouping of elements, an identifier, class, or single XHTML element (body, div, etc.)

```
p {
  font-family: sans-serif;
  color: red;
}
```

Attaching a CSS file <link>

```
<head>
  ...
<link href="filename" type="text/css" rel="stylesheet" />
  ...
</head>
```

- A page can link to multiple style sheet files
- In case of a conflict (two sheets define a style for the same HTML element), the latter sheet's properties will be used

```
<link href="style.css" type="text/css" rel="stylesheet" />
<link href="http://www.google.com/uds/css/gsearch.css"
rel="stylesheet" type="text/css" />
```

Absolute Units

Unit	Description
px	Pixels, ideal for screen-based design.
pt	Points, equal to 1/72 of an inch, used in print.
pc	Picas, equal to 12 points or 1/6 of an inch.
in	Inches, a physical unit of measurement.
cm	Centimeters, a physical unit of measurement.
mm	Millimeters, a physical unit of measurement.

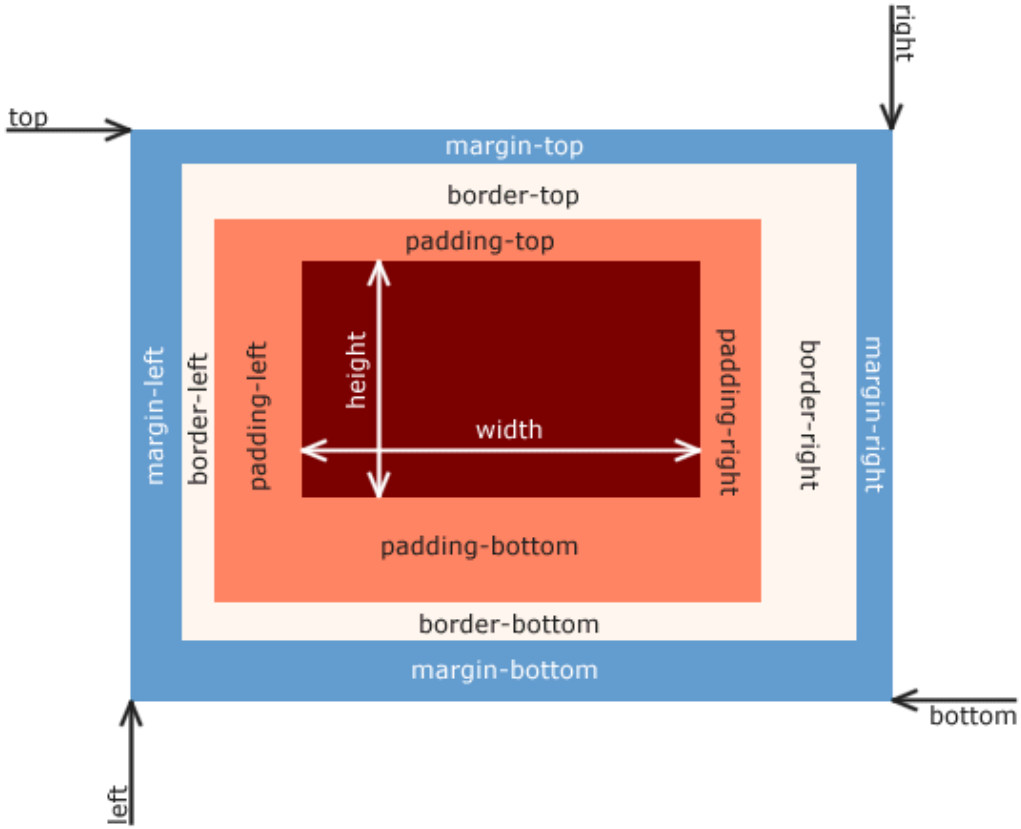
Relative Units

Unit	Description
em	Relative to the font size of the element.
rem	Relative to the font-size of the root element.
%	Percent, relative to the parent element's property.
vw	Viewport width, 1% of the viewport's width.
vh	Viewport height, 1% of the viewport's height.
vmin	1% of the viewport's smaller dimension.
vmax	1% of the viewport's larger dimension.

Experimental/Less Common Units

Unit	Description
ex	Roughly the height of a lowercase letter.
ch	Width of the "0" character in the current font.
q	Quarter-millimeters, mainly used in print contexts.

The Box Model



- Every element in the DOM (Document Object Model) has a conceptual "box" for presentation.
- The box consists of margin, padding, border, content (width, height), and offset (top, left)

Color formats in CSS

- **Color Names** : Predefined names for basic and extended colors (e.g., red, blue, green)
- **Hexadecimal Codes (Hex)** : 6-digit code preceded by # (e.g., #FF0000 for red) RGB and RGBA Values : Specify intensity of red, green, and blue components (0-255) (e.g., rgb(255, 0, 0) for red)
- **HSL and HSLA Values** : Define color based on hue, saturation, lightness (e.g., hsl(0, 100%, 50%) for red

Choosing the right color format (1)

- **Color names** : Good for basic colors or quick prototypes
- **Hex codes** : Ideal for precise color control or matching design palettes
- **RGB/RGBA values** : Useful for programmatic color manipulation or working with design tools
- **HSL/HSLA** : Suitable for users who prefer a more intuitive approach to describing colors

Choosing the right color format (2)

- **Ease of use** : Color names are the simplest, while hex codes and RGB/RGBA values require more technical knowledge.
- **Precision** : Hex codes and RGB/RGBA values offer the most precise color control.
- **Flexibility** : RGB/RGBA values are well-suited for programmatic manipulation.
- **Intuition** : HSL/HSLA can be easier to understand for some users.

Color Names

- A color can be specified by using a predefined color name : red, green, blue, yellow, black, white
- CSS/HTML support 140 standard color names
- Offer a limited range compared to other methods (hex, RGB, ..)

```
body {
  background-color: lightblue;
  color: darkslategray;
}
h1 {
  color: red;
}
p {
  color: blue;
}
```

Color Names

- **RGB(red, green, blue)** : each parameter defines the intensity of the color between 0 and 255
- **RGBA (Red, Green, Blue, Alpha)** is an extension of the RGB color model in CSS. It allows to define colors with both color and transparency alpha : Alpha channel value (0.0 - 1.0) representing transparency :
  1. 0.0 : Fully transparent
  2. 1.0 : Fully opaque (default)

```
p {    /* Applying RGB color to text - Green color */
  color: rgb(0, 128, 0);
}
div { /* Red color with 50% opacity */
    background-color: rgba(255, 0, 0, 0.5);
}
```

Hexadecimal Colors (Hex)

- Offer precise control and are widely used in web development
- Starts with # followed by 6 hexadecimal digits (0-9, A-F)
- **#rrggbb** : rr (red), gg (green) and bb (blue)

```
h1 {
  color: #FF0000; /* Red */
}

p {
  color: #333333; /* Gray */
}

a:link {
  color: #0000FF; /* Blue */
}
```

HSL/HSLA Colors

HSL(hue, saturation, lightness)

- **Hue** : Represents the color itself on a color wheel (0-360 degrees, where 0 is red and 180 is cyan).
- **Saturation** : color intensity (0% is gray, 100% is full saturation).
- **Lightness** : Controls the brightness of the color (0% is black, 100% is white).

HSLA : Alpha : Represents transparency (0.0 - 1.0, where 0.0 is fully transparent and 1.0 is fully opaque)

```
h1 {
  color: hsl(0, 100%, 50%); /* Red */
}
.header { /* Green color with 70% opacity */
  background-color: hsla(120, 100%, 50%, 0.7);
}
```

Property	Description	Example
`color`	Sets the text color	`color: #333;`
`font-family`	Specifies the font	`font-family: Arial, sans-serif;`
`font-size`	Sets the font size	`font-size: 16px;`
`font-weight`	Controls the font weight	`font-weight: bold;`
`font-style`	Applies font style (italic, etc.)	`font-style: italic;`
`text-align`	Aligns text horizontally	`text-align: center;`
`text-decoration`	Adds text decoration	`text-decoration: underline;`
`text-transform`	Controls text casing	`text-transform: uppercase;`
`line-height`	Sets the line height	`line-height: 1.5;`
`letter-spacing`	Adjusts character spacing	`letter-spacing: 2px;`
`text-shadow`	Applies shadow to text	`text-shadow: 2px 2px #000;`
`text-overflow`	Specifies text overflow behavior	`text-overflow: ellipsis;`
`white-space`	Specifies how white space is handled	`white-space: nowrap;`
`overflow-wrap`	Controls word wrapping	`overflow-wrap: break-word;`

Key CSS Text Formatting Properties

Basic Formatting

- **font-family** : Specifies the desired font family for the text.
- **font-size** : Sets the size of the text in various units (pixels, ems, rems, etc.).
- **font-weight** : Controls the boldness of the text (normal, bold, bolder, etc.).
- **color** : Defines the color of the text, using color names, hexadecimal codes, or RGB values.
- **text-decoration** : Adds decorative lines to the text (underline, overline, line-through, none).
- **text-align** : Aligns the text within the element (left, right, center, justify).
- **line-height** : Controls the vertical spacing between lines of text.

Advanced Formatting

- **letter-spacing** : Adjusts the amount of space between individual characters.
- **text-transform** : Transforms the text case (uppercase, lowercase, capitalize, etc.).
- **text-shadow** : Adds a shadow effect to the text.
- **text-indent** : Indents the first line of text.
- **font-style** : Specifies additional styles like italic or oblique.

Selecting Text :

- **: :selection** : Styles the text that is currently selected by the user

Example : CSS Text Formatig

```css
body {
  font-family: Arial,
        sans-serif;
  font-size: 1em;
  line-height: 1.5;
  margin: 0;
  padding: 0;
}

header {
  background-color: #f0f0f0;
  padding: 20px;
  text-align: center;
}

header p {
  font-style: italic;
}
```

```css
main {
  padding: 20px;
}

h1 {
  font-size: 2em;
}

p {
  margin-bottom: 15px;
  text-align: justify;
}

footer {
    text-align: center;
    padding: 10px;
    background-color: #f0f0f0;
}
```

Example : CSS Input Formatig

```
input[type="text"],
input[type="email"] {
  margin-bottom: 10px;
  width: 200px;
  height: 30px;
  border: 1px solid #ccc;
  padding: 5px;
  font-size: 16px;
}
```

```
textarea {
  width: 100%;
    /* width : 100% of its
            container */
  min-height: 100px;
  border: 1px solid #ccc;
  padding: 10px;
  font-size: 16px;
}
```

Before the Flexbox

Before the Flexbox, there were four layout modes :

- Block, for sections in a web page
- Inline, for text
- Table, for two-dimensional table data
- Positioned, for explicit position of an element

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

About Flex

- Flexbox is a one-dimensional layout model
- Designed to provide greater control over alignment and space distribution between items within a container.
- Being one-dimensional, it only deals with layout in a single direction - columns or rows - at a time. This works well for smaller layouts, such as components

Flex container properties

- **display** : Defines a flex container ; set to flex or inline-flex.
- **flex-direction** : Sets the main axis direction (row, row-reverse, column, column-reverse).
- **flex-wrap** : Controls the items' wrapping (nowrap, wrap, wrap-reverse).
- **flex-flow** : Shorthand for flex-direction and flex-wrap.
- **justify-content** : Aligns items along the main axis (f lex-start, flex-end, center, space-between, space-around, space-evenly).
- **align-items** : Aligns items along the cross axis (stretch, flex-start, flex-end, center, baseline).
- **align-content** : Distributes space between rows (stretch, flex-start, flex-end, center, space-between, space-around).

Flex Items properties

The direct child elements of a flex container automatically becomes flexible (flex) items.

- **order** : control order of items (override source order).
- **flex-grow** : how much an item can grow to fill extra space.
- **flex-shrink** : how much an item can shrink if there's not enough space.
- **flex-basis** : initial size of the item before considering grow/shrink.
- **flex** : shorthand for flex-grow, flex-shrink, and flex-basis.
- **align-self** : override default alignment for individual items.

Flex container properties

The use media of queries to create different layouts for different screen sizes and devices

Example : create a two-column layout for most screen sizes, and a one-column layout for small screen sizes (such as phones and tablets)

```
.flex-container {
    display: flex;
    flex-direction: row;
  }

  /* Responsive layout - makes a one column layout instead of a two-column
          layout */
  @media (max-width: 800px) {
    .flex-container {
      flex-direction: column;
    }
  }
```

Example

```
.flex-container {
  display: flex; /*Establishes this container as a flex container */
  justify-content: space-around; /* Distributes space around items */
  align-items: center; /* Vertically centers items in the container */
  flex-wrap: wrap; /* Allows items to wrap onto multiple lines as needed */
  padding: 20px;
  background: lightgrey;
}
.flex-item {
  background: navy;
  color: white;
  padding: 20px;
  margin: 10px;
  flex: 1; /* Allows items to grow to fill available space */
  text-align: center;
}
  /* Responsive behavior */
@media (max-width: 600px) {
  .flex-container {
      flex-direction: column; /* Stack items vertically on small screens */
  }
}
```

TP CSS

Create three web pages using **flexbox**, **text formatting**, and **input elements** :

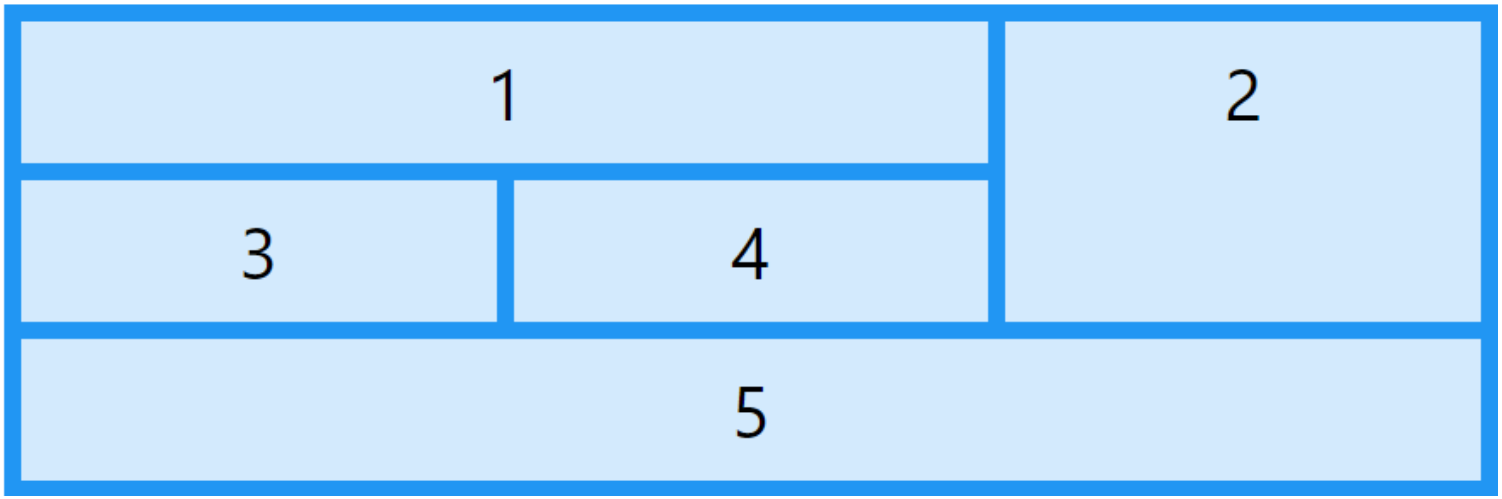
- Login Page
- Profile Page
- University Information Page

Showcase your skills in design and implementation while focusing on usability and creativity.

Example : https://github.com/hamdani2023/Flex_ISIL_2024

About Grid

Offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning



Grid example

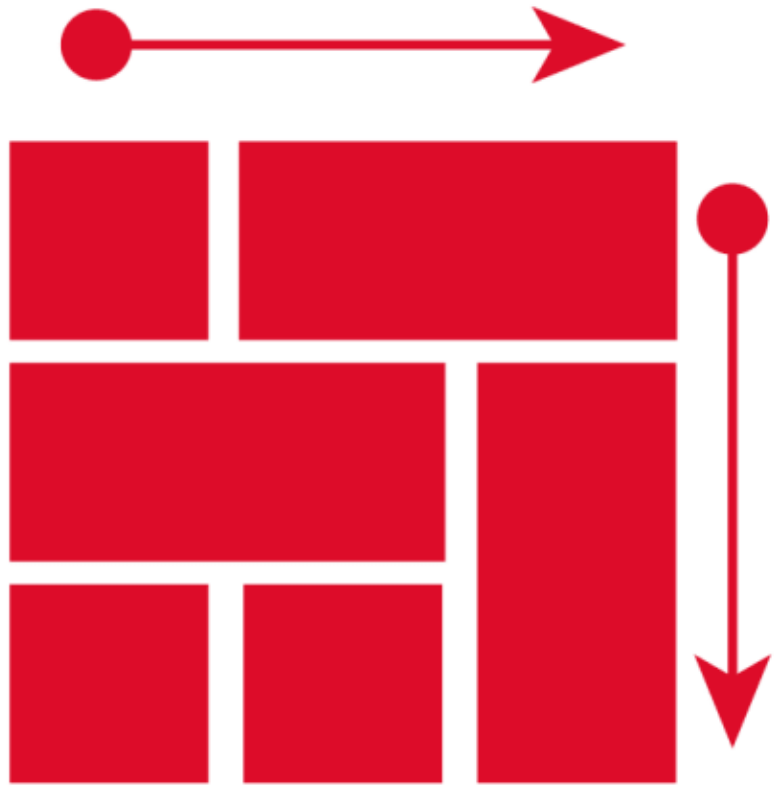
What Grid is ?

- Two-dimensional layout system
- Control of larger layouts, such as whole page layouts
- Similar to tables, it allows for items to be aligned in columns and rows.
- Easier to control and provides more layout options than old table-based layouts.

Grid vs Flexbox

Flexbox

ONE DIMENSION



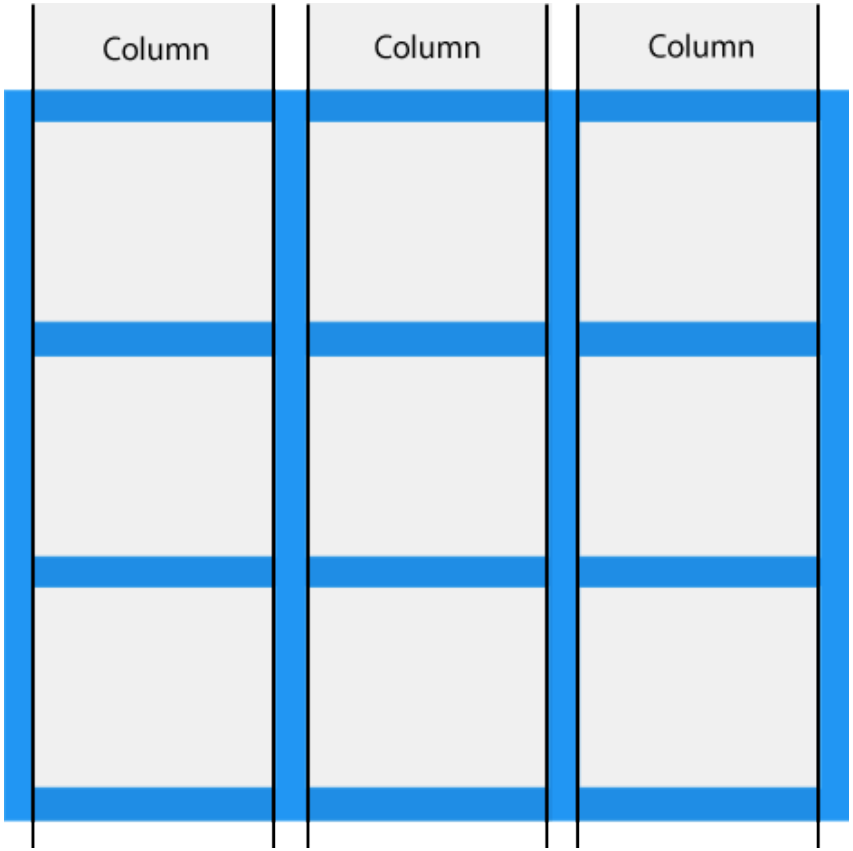
CSS Grids

TWO DIMENSIONS

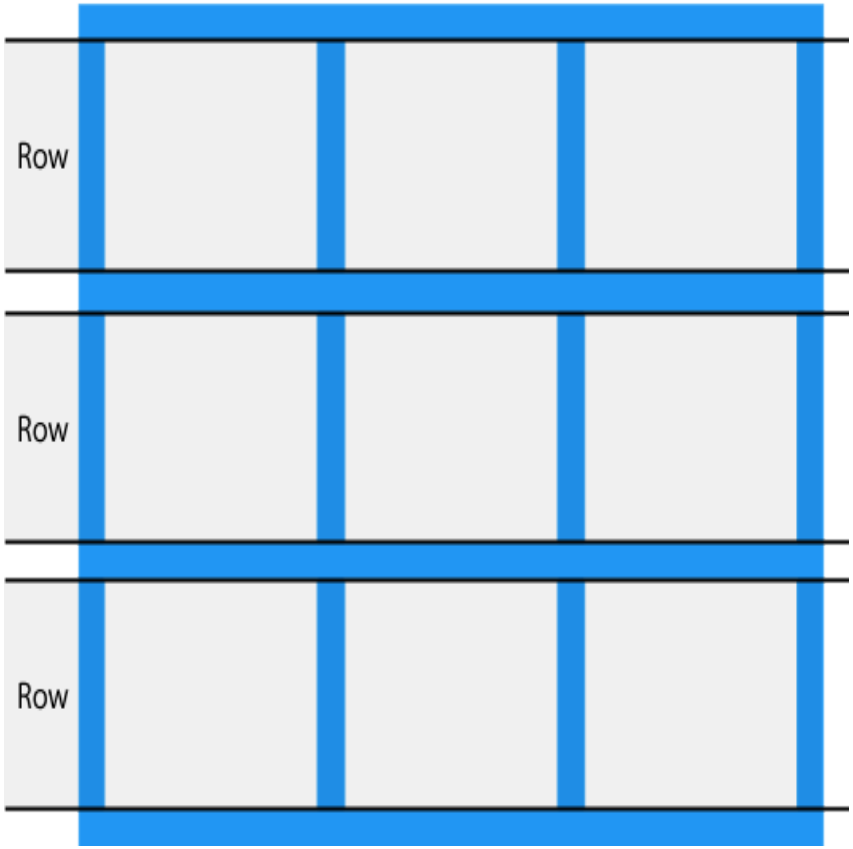
Difference between CSS Grid and Flexbox

- Flexbox is one-dimensional and CSS Grid is two-dimensional
- Flexbox is content-first and CSS Grid is layout-first :
  1. Flexbox is more content-first, adapting to the size of its content. It's useful for distributing space and aligning items in a container when their size is dynamic or unknown.
  2. Grid is more layout-first, meaning you define the grid structure and then place items into it, which can be more aligned with a designer's approach to layout planning.

Grid : Columns and Rows

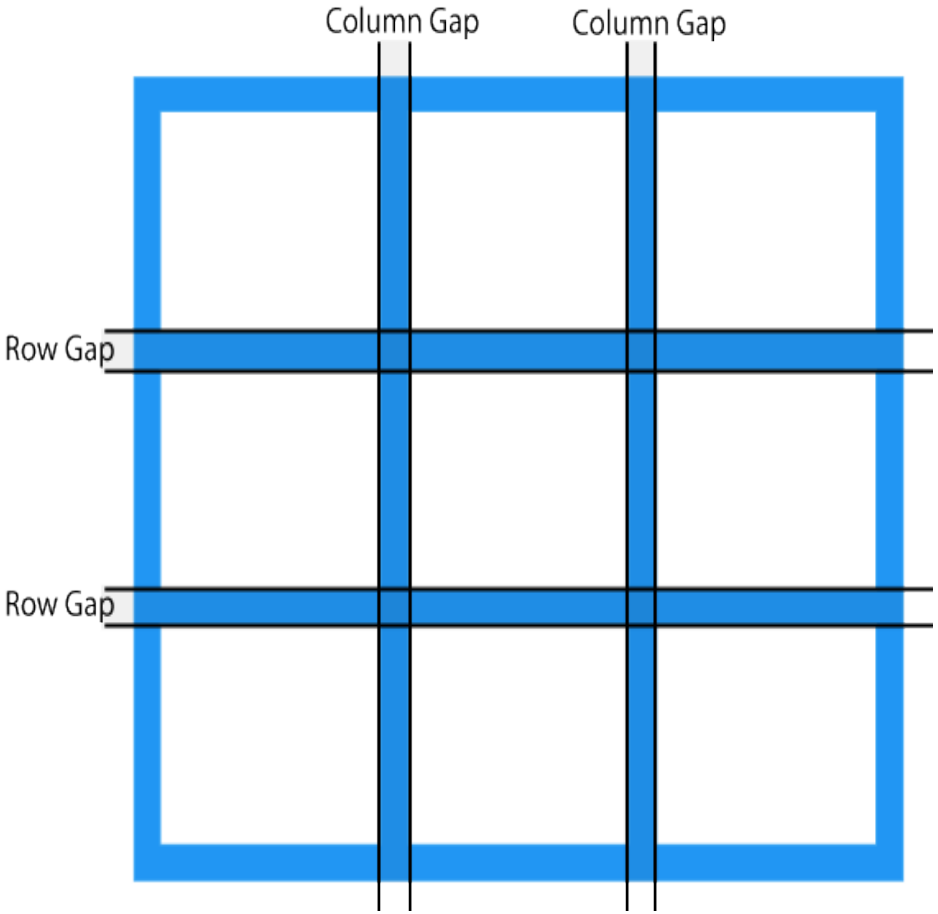


Grid Columns

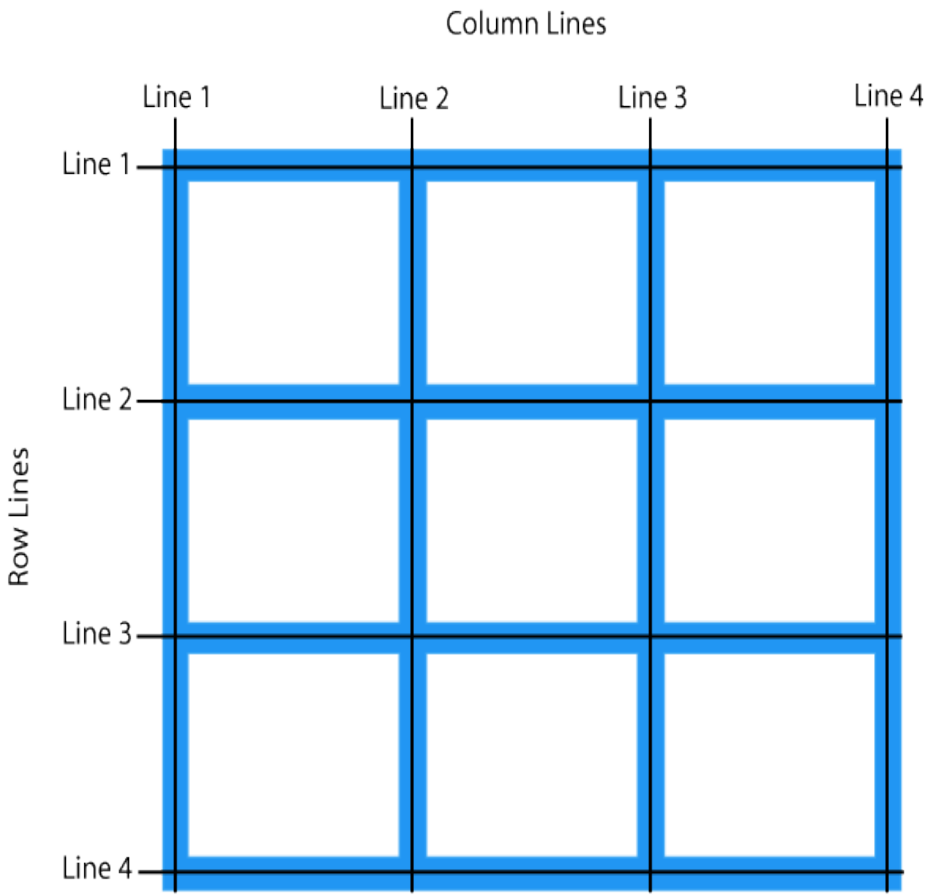


Grid Rows

Grid : Gaps and Lines



Grid Gaps



Grid Lines

Grid container properties

- **display** : Activates grid layout ; grid or inline-grid
- **grid-template-columns, grid-template-rows** : Define sizes of columns and rows.
- **grid-template-areas** : Assigns names to parts of the grid layout.
- **gap (grid-gap)** : Sets space between rows and columns.
- **grid-auto-columns, grid-auto-rows** : Sizes for implicitly created grid tracks.
- **grid-auto-flow** :Directs auto-placement of grid items ; row, column, dense.
- **justify-items** : Aligns items horizontally within their grid area.
- **align-items** : Aligns items vertically within their grid area.
- **justify-content** : Aligns the grid within the container horizontally.
- **align-content** : Aligns the grid within the container vertically.
- **grid-template** : Shorthand for grid-template-rows, grid-template-columns, and grid-template-areas.

Example

```
.boxes {
  display: grid;
     /*  grid-template-columns: 1fr 2fr 1fr;
      can be written :
      grid-template-columns: repeat(3, 1fr)

      grid-template-columns: repeat(auto-fit,
            minmax(100px, 1fr));
      */
  grid-template-columns: repeat(3, 1fr);
       /*1fr: one fraction*/
  gap: 1em;
  grid-auto-rows: minmax(100px, auto);
    /* define the size of rows */
}
```

Grid Item Properties

- **grid-column-start** : Item's start line in grid columns.
- **grid-column-end** : Item's end line in grid columns.
- **grid-row-start** : Item's start line in grid rows.
- **grid-row-end** : Item's end line in grid rows.
- **grid-column** : Shorthand for column start/end (e.g., 1 / 3).
- **grid-row** : Shorthand for row start/end (e.g., 2 / 4).
- **grid-area** : Shorthand for row/column start/end or named area.
- **justify-self** : Aligns item in cell along row axis.
- **align-self** : Aligns item in cell along column axis.
- **order** : Defines the order in which an item appears in the grid

Naming Grid Items

Use the grid-template-areas property on the grid container to define named areas. Each name corresponds to a specific area of the grid. Unnamed areas can be marked with a period (.).

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: auto;
  grid-template-areas:
      "header header header"
      "sidebar content content"
     "footer footer footer";
}
.header  {  grid-area: header;  }
.sidebar {  grid-area: sidebar; }
.content {  grid-area: content; }
.footer  {  grid-area: footer;  }
```

Questions ?