Pseudo class -

CSS pseudo-classes are special keywords added to selectors to define a specific state or condition of an element, allowing you to style elements based on their state, user interaction, position in the document, or other criteria without needing to add extra classes or modify the HTML.

Key Characteristics

- A pseudo-class starts with a colon (:) followed by the pseudo-class name (e.g., :hover).
- It targets elements in a special state, such as when a user hovers over them, when an element is the first child of its parent, when a form element is checked, or when a link is visited.
- Pseudo-classes help apply styles dynamically based on element states or contextual conditions.

Syntax

```
selector:pseudo-class {
  property: value;
}
```

For example, to change the color of a button when hovered:

```
button:hover {
  color: blue;
}
```

Common CSS Pseudo-classes

Pseudo-class	Description	Example
:hover	When the user hovers over an element	a:hover { color: red; }
:active	When an element (e.g., a link or button) is clicked	button:active { background: darkgray; }
:focus	When an element gains keyboard focus	input:focus { outline: 2px solid blue; }
:visited	Matches links that the user has already visited	a:visited { color: purple; }
:first-child	Selects an element that is the first child of its parent	p:first-child { font-weight: bold; }
:last-child	Selects an element that is the last child of its parent	li:last-child { color: green; }
:checked	Selects checked checkboxes or radio buttons	input:checked { border-color: blue; }
:not(selector)	Selects elements that do not match the given selector	div:not(.active) { opacity: 0.5; }
:nth-child(n)	Matches elements that are the nth child of their parent	li:nth-child(2) { background: yellow; }

Usage Examples

1. Change link color when hovering:

```
a:hover {
  color: red;
}
```

1. Style the first paragraph inside a section:

```
section p:first-child {
  font-weight: bold;
}
```

1. Style a checked checkbox:

```
input[type="checkbox"]:checked {
  background-color: lightblue;
}
```

Important Notes

- Pseudo-classes allow styling elements based on dynamic states without modifying the HTML.
- Some pseudo-classes take arguments inside parentheses, e.g., :nth-child(2).
- The order of pseudo-classes in CSS matters for things like link states (:link, :visited, :hover, :active).
- Pseudo-classes differ from pseudo-elements, which style parts of elements (using ::).

Using pseudo-classes enhances interactivity, usability, and contextual styling in web design efficiently.

Column Layout -

The CSS column layout, often called the multi-column layout, allows you to divide content into multiple columns, similar to how text appears in newspapers or magazines. This layout enhances readability and organizes large blocks of text efficiently by flowing content across columns.

Key Properties of CSS Multi-Column Layout

1. **column-count**
   Specifies the number of columns into which the content is divided.
2. column-count: 3;
3. **column-width**
   Defines the ideal width for each column. The browser will create as many columns as can fit while respecting this width.
4. column-width: 150px;
5. **columns** (shorthand)
   Sets both column-width and column-count at once.

6. columns: 150px 3; */* width 150px and max 3 columns */*
7. **column-gap**
   Specifies the gap (space) between columns.
8. column-gap: 20px;
9. **column-rule**
   Defines a dividing line between columns. This is a shorthand for column-rule-width, column-rule-style, and column-rule-color.
10. column-rule: 2px solid gray;
11. **column-span**
    Determines whether an element spans across all columns or just one. For example, you could make a heading span across all columns.
    Values: none (default), all.
12. h2 {
13. column-span: all;
14. }
15. **Content Flow Controls: break-before, break-after, break-inside**
    Control how content breaks between columns, helping to avoid awkward splits, especially for paragraphs, images, or other block elements.

Simple Example: Dividing text into three columns

```
.article {

 column-count: 3;

 column-gap: 20px;

 column-rule: 1px solid #ccc;

}
```

This CSS splits the .article content into three vertical columns with a 20px gap and a subtle line separating each column.

How it Works

- The content flows from one column to the next automatically.
- You can specify either the number of columns (column-count) or the minimum width (column-width) to let the browser determine the best fit.
- The layout adapts responsively by automatically adjusting the number of columns based on available space.
- You can combine multi-column layout with other CSS layout tools like Flexbox or Grid for complex designs.

Use Cases

- Displaying text-heavy content like articles, blogs, newsletters.

- Creating magazine or newspaper style layouts.
- Enhancing readability by breaking long lines of text into readable chunks.

Related CSS Layout Methods

- **Flexbox and Grid** can also be used for column layouts but work by placing child elements into separate containers, whereas multi-column layouts flow continuous content across columns.
- Multi-column layout is best for divided text flow rather than arranging separate block-level items.

Flexbox -

CSS Flexbox (Flexible Box Layout) is a powerful and modern layout system designed to arrange elements efficiently in one dimension—either a row or a column. It simplifies the process of distributing space and aligning items inside a container, particularly when building flexible, responsive web designs.

Core Concepts of Flexbox

1. Flex Container and Flex Items

- **Flex Container**: The parent element with display: flex (or inline-flex). This container activates the flex context and its direct children become flex items.
- **Flex Items**: The children inside the flex container that are arranged according to Flexbox rules.

2. Main Axis and Cross Axis

- Flexbox layouts work along two axes:
  - **Main Axis**: Defined by flex-direction (row or column). Items are placed along this axis.
  - **Cross Axis**: Perpendicular to the main axis.

For example, if flex-direction: row; then the main axis is horizontal, and the cross axis is vertical.

Key Properties for Flex Container

Property	Description	Default Value
display	Defines the element as a flex container	flex (or inline-flex)
flex-direction	Sets the direction of the main axis (row, row-reverse, column, column-reverse)	row
flex-wrap	Controls whether flex items wrap to next line if they overflow (nowrap, wrap, wrap-reverse)	nowrap
justify-content	Aligns items along the main axis (start, end, center, space-between, space-around, space-evenly)	flex-start
align-items	Aligns items along the cross axis (start, end, center, stretch, baseline)	stretch
align-content	Aligns multiple lines along the cross axis when wrapping occurs	stretch
gap	Sets spacing between flex items (horizontal and vertical gap)	0

Key Properties for Flex Items

Property	Description	Default Value
flex-grow	Defines ability to grow relative to other flex items	0
flex-shrink	Defines ability to shrink relative to other flex items	1
flex-basis	Initial main size before growing or shrinking	auto
flex	Shorthand for flex-grow, flex-shrink, and flex-basis	0 1 auto
align-self	Overrides align-items for individual item alignment	auto (follows container)

Basic Example of Flexbox Setup

css

```
.container {
  display: flex;              /* Defines flex container */
  flex-direction: row;        /* Items arranged in a row */
  justify-content: space-between; /* Space between items along main axis */
  align-items: center;        /* Vertically center items */
  gap: 10px;                  /* Space between items */
}

.item {
  flex-grow: 1;               /* Items grow equally */
  padding: 10px;
  background-color: lightblue;
```

```
}
```

xml

```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2 with more content</div>
  <div class="item">Item 3</div>
</div>
```

The items will be distributed horizontally with equal growing space, centered vertically, and spaced nicely with a 10px gap.

Common Use Cases and Behaviors

- **Responsive Navigation Bars:** Flexbox can adapt navigation items to different screen sizes with wrapping.
- **Vertical/Horizontal Centering:** Easily center content inside containers.
- **Dynamic Size Adjustments:** Flex items can grow and shrink dynamically without fixed dimensions.
- **Reversing Layout:** Use row-reverse or column-reverse to reverse item order visually.
- **Wrapping Items:** Add flex-wrap: wrap; to allow items to wrap onto multiple lines if container width is insufficient.

Summary of Important Flexbox Properties and Effects

Property	What It Controls	Values/Examples
display	Activates flexbox	flex, inline-flex
flex-direction	Direction of items (main axis)	row, column, row-reverse, column-reverse
flex-wrap	Wrapping behavior	nowrap (default), wrap
justify-content	Main axis alignment	flex-start, center, space-between, etc.
align-items	Cross axis alignment	stretch (default), center, flex-start, etc.
align-self	Overrides cross axis alignment per item	auto, center, flex-start, etc.
flex-grow	How items grow relative to others	Number (0 or positive)
flex-shrink	How items shrink relative to others	Number (default 1)
flex-basis	Initial size before grow/shrink	auto, or width/height values
gap	Space between flex items	10px, 1rem, 2%, etc.

Mastering Flexbox empowers you to build clean, responsive web layouts with minimal code and great control over spacing, alignment, and distribution.

Grid -

CSS Grid Layout is a powerful two-dimensional layout system designed specifically for managing rows and columns in web design. It allows you to create complex and responsive

grid-based layouts with ease, providing fine control over element placement both vertically and horizontally.

Core Concepts of CSS Grid

1. Grid Container and Grid Items

- **Grid Container**: The parent element with display: grid or display: inline-grid. This enables grid layout for its direct children.
- **Grid Items**: The immediate children of the grid container that are placed onto the grid.

2. Rows and Columns

CSS Grid works in two dimensions with explicit rows and columns defined in the container. You can control the size, number, and spacing of rows and columns.

Key Properties for the Grid Container

Property	Description	Example Value
display	Defines the element as a grid container	grid or inline-grid
grid-template-columns	Defines the number and widths of columns	100px 200px, 1fr 2fr
grid-template-rows	Defines the number and heights of rows	50px 100px, auto 1fr
grid-template-areas	Defines named grid areas for layout	"header header" "main sidebar"
grid-column-gap / grid-row-gap	Sets gap/space between columns and rows	20px
gap or grid-gap	Shorthand for setting row and column gaps	10px 20px
justify-items	Aligns grid items horizontally inside their cells	start, center, stretch
align-items	Aligns grid items vertically inside their cells	start, center, stretch
justify-content	Aligns the whole grid inside the container horizontally	start, center, space-between
align-content	Aligns the whole grid inside the container vertically	start, center, space-around

Key Properties for Grid Items

Property	Description	Example Value
grid-column-start	Starts grid item at a specific column line	1, span 2
grid-column-end	Ends grid item at a specific column line	3
grid-row-start	Starts grid item at a specific row line	2
grid-row-end	Ends grid item at a specific row line	span 3
grid-area	Shorthand for setting both rows and columns	1 / 1 / 3 / 4 or a named area

Units Used in Grid

- px — Fixed pixels
- fr — Fractional unit, representing a fraction of available space
- % — Percentage relative to container
- auto — Size based on content or other settings
- minmax(min, max) — Defines a size range for a track
- repeat(count, value) — Repeats a pattern of track sizes

Basic Example

css

```
.container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr;
  grid-template-rows: 50px auto 50px;
  gap: 10px 20px; /* row gap 10px, column gap 20px */
}
```

```
.item1 {
  grid-column: 1 / 2; /* from column line 1 to 2 */
  grid-row: 1 / 3;    /* spans rows 1 and 2 */
}

.item2 {
  grid-column: 2 / 4; /* spans columns 2 and 3 */
  grid-row: 2 / 4;    /* spans rows 2 and 3 */
}
```

Named Grid Areas Example

css

```
.container {
  display: grid;
  grid-template-columns: 150px 1fr 150px;
  grid-template-rows: 80px 1fr 80px;
  grid-template-areas:
    "header  header  header"
    "sidebar main    ads"
    "footer  footer  footer";
  gap: 10px;
}

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

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

.main {
  grid-area: main;
}

.ads {
  grid-area: ads;
}

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

Benefits of CSS Grid

- Precise two-dimensional control for layouts.
- Easily create responsive designs with flexible units.
- Simplifies positioning of elements without floats or complex margin tricks.
- Supports overlapping of grid items.
- Works well in combination with Flexbox for versatile web layouts.

Using CSS Grid empowers you to build advanced, clean, and highly customizable page layouts efficiently. It is increasingly supported by all modern browsers and is a foundational tool in modern web design.