**CSS Flexbox**

**1. Introduction**

Flexbox (Flexible Box Layout) is a CSS layout module designed to make it easier to arrange items in rows or columns and distribute space dynamically, even when their size is unknown.  
It excels at:

* Centering items vertically and horizontally
* Equal spacing between items
* Creating fluid, responsive layouts without floats or complex calculations

**2. Flex Container vs. Flex Items**

* Flex Container — The parent element with display: flex or display: inline-flex.
* Flex Items — The direct children of a flex container.

**3. Creating a Flex Container**

.container {

display: flex; ------- Creates a block-level flex container.

display: inline-flex; ---------Creates an inline-level flex container.

}

4. Main Axis vs. Cross Axis

* Main Axis — The direction flex items are placed in (row or column).
* Cross Axis — The perpendicular direction to the main axis.

If flex-direction: row, the main axis is horizontal and cross axis is vertical.  
If flex-direction: column, the main axis is vertical and cross axis is horizontal.

**5. Flex Container Properties**

**5.1 flex-direction**

Defines the main axis direction.

flex-direction: row; --- ---**default, left to right.**

flex-direction: row-reverse; ------ **right to left.**

flex-direction: column; ------ **top to bottom.**

flex-direction: column-reverse; ------ **bottom to top.**

Note: This effects all will be applied to the flex-items

**5.2 flex-wrap**

Controls whether items wrap to a new line.

flex-wrap: nowrap; /\* default, all items in one line \*/

flex-wrap: wrap; /\* items wrap to new rows/columns \*/

flex-wrap: wrap-reverse; /\* wrap in reverse order \*/

**5.3 flex-flow**

Shorthand for flex-direction + flex-wrap.

flex-flow: row wrap; /\* direction + wrap mode \*/

**5.4 justify-content (along main axis)**

Aligns items horizontally if flex-direction: row or vertically if flex-direction: column.

justify-content: flex-start; /\* default, items at start \*/

justify-content: flex-end; /\* items at end \*/

justify-content: center; /\* centered \*/

justify-content: space-between; /\* equal space between items \*/

justify-content: space-around; /\* equal space around items \*/

justify-content: space-evenly; /\* equal space between & outside \*/

**5.5 align-items (along cross axis)**

Aligns items vertically if flex-direction: row or horizontally if flex-direction: column.

align-items: stretch; /\* default, stretch to fill \*/

align-items: flex-start; /\* align to start of cross axis \*/

align-items: flex-end; /\* align to end of cross axis \*/

align-items: center; /\* centered \*/

align-items: baseline; /\* align text baselines \*/

**5.6 align-content (only works with multiple rows/columns)**

align-content: stretch; /\* default \*/

align-content: flex-start;

align-content: flex-end;

align-content: center;

align-content: space-between;

align-content: space-around;

align-content: space-evenly;

6. Flex Item Properties

Note:*Use these when you want to style a specific flex item differently from the others inside the same flex container.*

**6.1 order**

Purpose:  
Changes the visual order of flex items without altering the HTML source order.

Syntax:

.item {

order: 2; /\* default: 0 \*/

}

**How it works:**

* All flex items have an implicit order: 0.
* The browser sorts items by order value (lowest to highest) before rendering.
* If multiple items have the same order, they follow HTML source order.

Example:

css

.first { order: 2; }

.second { order: 1; }

.third { order: 3; }

HTML:

<div class="container">

<div class="first">A</div>

<div class="second">B</div>

<div class="third">C</div>

</div>

Rendered order: B → A → C

Note: This only changes *visual* order, not the DOM order (important for accessibility and screen readers).

**6.2 flex-grow**

**Purpose:**  
Defines how much a flex item will grow relative to other items when there’s extra space in the container.

Syntax:

css

.item {

flex-grow: 1; /\* default: 0 \*/

}

**How it works:**

* A value of 0 means don’t grow at all.
* A value of 1 means grow proportionally to other items’ flex-grow values.
* The browser distributes extra space in proportion to the sum of flex-grow values.

Example:

css

.a { flex-grow: 1; }

.b { flex-grow: 2; }

.c { flex-grow: 1; }

If the container has 200px free space, distribution:

* .a → 50px
* .b → 100px
* .c → 50px

**6.3 flex-shrink**

Purpose:  
Defines how much a flex item will shrink relative to others when there’s not enough space in the container.

Syntax:

css

.item {

flex-shrink: 1; /\* default: 1 \*/

}

**How it works:**

* 0 → don’t shrink at all (item keeps its base size, might cause overflow).
* 1 → shrink proportionally to others when necessary.
* Shrink ratios work similarly to grow ratios but in reverse (removing space instead of adding).

Example:

css

.a { flex-shrink: 1; }

.b { flex-shrink: 3; }

If 100px needs to be removed, .b will shrink 3x more than .a.

**6.4 flex-basis**

Purpose:  
Sets the initial size of the flex item before growing or shrinking is applied.

Syntax:

css

.item {

flex-basis: 200px; /\* default: auto \*/

}

How it works:

* auto means the size is based on content, width/height, or min/max constraints.
* Can be in px, %, em, rem, etc.
* When flex-basis is set, it overrides the element’s width (for horizontal main axis) or height (for vertical main axis) for the flex calculation.

Example:

css

.a { flex-basis: 100px; }

.b { flex-basis: 200px; }

These sizes are the starting point before applying flex-grow or flex-shrink.

**6.5 flex (shorthand)**

Purpose:  
Shorthand for flex-grow flex-shrink flex-basis.

Syntax:

css

.item {

flex: 1 1 200px; /\* grow=1, shrink=1, basis=200px \*/

}

How it works:

* First value = flex-grow
* Second value = flex-shrink
* Third value = flex-basis
* Common patterns:
  + flex: 1 → grow=1, shrink=1, basis=0%
  + flex: auto → grow=1, shrink=1, basis=auto
  + flex: none → grow=0, shrink=0, basis=auto

Example:

css

.a { flex: 2 1 150px; }

Starts at 150px, can grow 2x compared to others, can shrink at normal speed.

6.6 align-self

Purpose:  
Overrides the container’s align-items setting for an individual item.

Syntax:

css

.item {

align-self: center;

}

Values:

* auto (default, inherits from align-items)
* flex-start / flex-end / center / baseline / stretch

Example:

Css

.container { align-items: flex-start; }

.special { align-self: flex-end; }

All items align at the top except .special, which aligns at the bottom.

7. Flexbox Cheatsheet Summary

| Property | Applies To | Purpose |
| --- | --- | --- |
| display | container | Creates flex context |
| flex-direction | container | Main axis direction |
| flex-wrap | container | Wrapping behavior |
| flex-flow | container | Direction + wrap shorthand |
| justify-content | container | Align on main axis |
| align-items | container | Align on cross axis |
| align-content | container | Space between lines |
| order | item | Change order |
| flex-grow | item | Grow ratio |
| flex-shrink | item | Shrink ratio |
| flex-basis | item | Initial size |
| flex | item | Grow/shrink/basis shorthand |
| align-self | item | Override cross-axis alignment |