✅ CSS Flexbox (Flexible Box Layout)

We use Flexbox because it makes creating responsive and flexible layouts much easier.
If we have a container with elements, it is difficult to adjust their position, spacing, and make them responsive using older methods. But with the CSS Flexbox property, we can easily control layout.

- We can set the **direction** of elements (row or column).
- Even if elements are block-level, we can arrange them on the same line.
- `justify-content` → to align items horizontally
- `align-items` → to align items vertically
- `flex-grow`, `flex-shrink`, `flex-basis` → to control item sizes

This helps make the page responsive and visually well-structured. We mostly use Flexbox to handle layouts like cards, navigation bars, and content sections.

Example:

css Copy code

```css
.container {
  display: flex;
  justify-content: center; /* center horizontally */
  align-items: center;     /* center vertically */
}
```

✅ This version is clear, grammatically correct, and good for interview explanation.

3. `align-items` (Vertical alignment → cross axis, single line)

- `flex-start` → top
- `flex-end` → bottom
- `center` → middle
- `stretch` → stretch to fill height
- `baseline` → align text baselines

```css
align-items: center;
```

---

4. `align-content` (Vertical alignment when multiple lines exist)

- Works **only when flex-wrap creates multiple rows.**
- Values: `flex-start`, `flex-end`, `center`, `space-between`, `space-around`, `stretch`.

```css
align-content: center;
```

---

5. `flex-wrap`

- `nowrap` (default) → all items in one line
- `wrap` → items move to next line if needed
- `wrap-reverse` → wrap but in reverse order

```css
flex-wrap: wrap;
```

---

6. `flex-flow`

Shorthand for `flex-direction` + `flex-wrap`

```css
flex-flow: row wrap;
```

7. `gap`, `row-gap`, `column-gap`

- `gap: 20px;` → space between items
- `row-gap: 40px;` → vertical gap
- `column-gap: 10px;` → horizontal gap
- `gap: 40px 10px;` → row-gap first, column-gap second

🔹 Item-Level Properties (Children of Flex Container)

1. `flex-grow`

- Defines how much an item **expands** to fill available space.
- Default: `0` (no extra growth).
- If all items have `flex-grow: 1;` → equal width.

```css
.item {
  flex-grow: 1; /* all items take equal width */
}
```

Copy code

---

2. `flex-shrink`

- Defines how much an item **shrinks** if space is limited.
- Default: `1`.
- Example: `flex-shrink: 2;` → shrinks **twice as fast** as others.

---

3. `flex-basis`

- Defines **initial size** of item before grow/shrink applies.
- Can be `auto`, px, %, etc.

```css
.item {
  flex-basis: 200px;
}
```

Copy code

Saved memory full

4. `flex` (shorthand)

```css
.item {
  flex: 1 1 200px; /* grow | shrink | basis */
}
```

5. `align-self`

- Overrides `align-items` for a single item.

```css
.item {
  align-self: flex-end;
}
```

🔹 Example Code

```css
.container {
  border: 2px solid red;
  display: flex;
  height: 80vh;

  justify-content: space-between; /* horizontal alignment */
  align-items: center;            /* vertical alignment */
  align-content: flex-start;      /* for multiple lines */

  flex-direction: row;            /* default left to right */
  flex-wrap: wrap;                /* items move to next line */
  flex-flow: row wrap;            /* shorthand */

  gap: 40px 10px;                 /* row-gap 40px, column-gap 10px */
}
```

✅ Where Flexbox is used?

- Navigation bars
- Cards & grids
- Centering content
- Responsive layouts
- Content alignment inside containers

Saved memory full

✅ CSS Grid Layout

🔹 What is CSS Grid?

CSS Grid is a powerful **two-dimensional layout system** in CSS. Unlike Flexbox (which works in one direction — row *or* column), Grid allows us to control layouts in **both rows and columns simultaneously**.

We use CSS Grid because it:

- Gives full control over **rows, columns, and cells**.
- Allows placing items anywhere inside the grid.
- Supports overlapping elements.
- Makes complex page layouts (dashboards, galleries, entire webpages) much easier and more responsive.

🔹 How to Use CSS Grid?

To create a grid, we first define a **grid container**:

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

Inside this container, every child becomes a **grid item**.

🔹 Important Grid Properties

For the container:

- `display: grid;` → activates grid.
- `grid-template-columns` → defines number & size of columns.

```css
grid-template-columns: 120px 100px auto;
grid-template-columns: repeat(4, 1fr); /* 4 equal flexible columns */
grid-template-columns: repeat(4, minmax(100px, 1fr)); /* responsive columns */
```

- `grid-template-rows` → defines rows.
- `gap`, `row-gap`, `column-gap` → spacing between cells.
- `justify-items` → horizontal alignment inside each cell.
- `align-items` → vertical alignment inside each cell.
- `justify-content` / `align-content` / `place-content` → align the entire grid inside the container.

For the items (children): Saved memory full

- `grid-column` → place an item across specific columns.

```css
.item1 { grid-column: 1 / 3; } /* spans from column line 1 to 3 */
```

- `grid-row` → place an item across specific rows.

```css
.item2 { grid-row: 1 / 2; } /* occupies first row */
```

- `grid-area` → combine row and column placement or name areas.
- `justify-self` → align a single item horizontally in its cell.
- `align-self` → align a single item vertically in its cell.

🔹 Grid Terminology

- **Grid line** → lines that divide rows and columns.
- **Grid cell** → single "box" in the grid.
- **Grid track** → space between two grid lines (a row or column).
- **Grid area** → space covered by one or more grid cells.

🔹 Example

```css
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;   /* flexible columns */
  grid-template-rows: auto auto;
  gap: 15px;
  justify-content: center; /* align grid horizontally in container */
  align-content: end;      /* align grid vertically in container */
}

.item1 {
  grid-column: 1 / 3;  /* spans first two columns */
  grid-row: 1;         /* placed in first row */
}

.item2 {
  grid-column: 3;      /* placed in third column */
  grid-row: 1 / 3;     /* spans two rows */
}
```

🔹 Why use CSS Grid?

- To design **two-dimensional layouts** easily.
- To place items **anywhere** inside the grid (unlike Flexbox, which only flows in one direction).
- To build **responsive, structured, and clean page designs** such as cards, galleries, or entire websites.

🔹 CSS Flexbox

One-dimensional layout → works in one direction at a time (row or column).

Best for aligning items in a single line or arranging content like navbars, buttons, cards in a row.

Properties: `flex-direction`, `justify-content`, `align-items`, `flex-grow`, `flex-shrink`.

Example: Centering elements horizontally and vertically inside a container.

```css
.container {
  display: flex;
  justify-content: center; /* horizontal */
  align-items: center;     /* vertical */
}
```

Process:
First, we create a container and apply `display: flex;`. Then, the elements inside the container (children) become flex items, and we can align, space, and resize them to create a responsive and flexible layout.

🔹 CSS Grid

Two-dimensional layout → works in rows and columns simultaneously.

Best for creating page layouts, dashboards, or image galleries.

Properties: `grid-template-columns`, `grid-template-rows`, `gap`, `grid-column`, `grid-row`.

Example: Creating a layout with rows and columns.

```css
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr; /* three columns */
  gap: 20px;
}
```

Process:
First, we create a container and apply `display: grid;`. Then, the elements inside the container (children) become grid items, and we can place them in rows and columns to create complex, responsive layouts.

🔹 Key Differences

Feature	Flexbox	Grid
Layout Type	One-dimensional (row OR column)	Two-dimensional (rows AND columns)
Use Case	Small components, content flow	Full page layouts, complex designs
Alignment	Great for distributing items on one axis	Great for controlling both axes
Placement	Items flow naturally in order	Items can be placed anywhere (even overlap)
Example	Navbars, forms, card rows	Dashboards, galleries, full webpages

---

✅ **In my experience, I mostly use Flexbox for navbars (because it's perfect for horizontal alignment) and Grid for cards or page layouts (because it gives better two-dimensional control).**

Media Query in CSS

Saved memory full

A media query in CSS is used to create responsive designs that adapt to different screen sizes (desktop, tablet, mobile).

👉 We define breakpoints (specific screen widths) and apply different styles before or after that width.
👉 This way, the same element looks good on all devices.

Example with Breakpoints

css

Copy code

```css
/* Default styles (for desktop) */
.container {
  width: 80%;
  font-size: 18px;
}

/* For tablets (max width 768px) */
@media (max-width: 768px) {
  .container {
    width: 90%;
    font-size: 16px;
  }
}

/* For mobile (max width 480px) */
@media (max-width: 480px) {
  .container {
    width: 100%;
    font-size: 14px;
  }
}

/* For landscape orientation */
@media screen and (orientation: landscape) {
  body {
    border: 2px solid rgb(8, 0, 8);
  }
}

/* For very small devices (max width 455px) */
@media only screen and (max-width: 455px) {
  body {
    background-color: blue;
  }
  .boxes {
    flex-direction: column;
  }
}
```