Section 4: CSS Flexbox

Definition

- Flexbox is a **CSS layout system** used to arrange items in a row or column.
- It makes it easy to align, space out, and resize items without needing extra code.
- Flexbox is an **alternative** to older layout methods like display: block or inline-block.

Key Points

- Flexbox works with **one-dimensional layouts** (either row or column).
- It helps **distribute space** and **align items** inside a container.
- Flexbox automatically adjusts to fit items within the available space.
- **Main axis** is where items are placed (horizontal by default).
- **Cross axis** is the opposite direction (vertical by default).

Advantages of Flexbox

- **Align items easily** in both horizontal and vertical directions.
- **Items resize automatically** based on available space.
- **No need for extra CSS** to distribute space or adjust item positions.
- Works well for **responsive designs**.

Flexbox Properties

Flexbox Container Properties

- **display: flex | display: inline-flex**
  - Activates the **Flexbox layout** on the container.
  - inline-flex makes the container behave like an **inline-level** element while still using Flexbox.
- **flex-direction**
  - Defines the direction in which flex items are arranged.
  - Options: row, column, row-reverse, column-reverse.
- **flex-wrap**
  - Controls whether flex items should wrap to a new line when they exceed the container width.
  - Options: nowrap (default), wrap, wrap-reverse.
- **flex-flow**
  - A shorthand property for **flex-direction** and **flex-wrap** combined.

- **column-gap**
    - Defines the **gap (space) between columns** in a column-based flex layout.
- **row-gap**
    - Defines the **gap (space) between rows** in a row-based flex layout.
- **gap**
    - A shorthand property for setting both **column-gap** and **row-gap** together.
- **justify-content**
    - Aligns flex items **along the main axis** (horizontal if flex-direction: row, vertical if flex-direction: column).
- **align-items**
    - Aligns flex items **along the cross axis** (vertical if flex-direction: row, horizontal if flex-direction: column).
- **align-content**
    - Aligns multiple lines of flex items when there is **extra space** in the cross axis.
    - Works when flex-wrap is set to wrap.

Flexbox Item Properties

- **order**
    - Controls the **visual order** of flex items.
    - Default is 0, higher values move items **further right or down**.
- **flex-grow**
    - Defines how much a flex item should **expand** if extra space is available.
- **flex-shrink**
    - Defines how much a flex item should **shrink** when space is limited.
- **flex-basis**
    - Specifies the **initial size** of a flex item **before** extra space is distributed.
- **flex**
    - A shorthand for **flex-grow**, **flex-shrink**, and **flex-basis** combined.
    - Example: flex: 1 1 auto;
- **align-self**
    - Overrides the align-items property for a **single flex item**.
    - Allows an item to have **custom alignment** within the flex container.

---

display: flex | display: inline-flex

Definition

- The display property in CSS **activates Flexbox layout** for a container.
- It has two main values for Flexbox:
  - **display: flex;** → Makes the container a **block-level flex container** (takes full width).
  - **display: inline-flex;** → Makes the container an **inline-level flex container** (fits content size).

Key Points

- **display: flex;**
  - The container behaves like a **block element**.
  - Takes up the **full width** of the parent container.
  - Items inside follow **flexbox rules** (aligned in a row/column).
- **display: inline-flex;**
  - The container behaves like an **inline element**.
  - It **shrinks to fit** the content inside it.
  - Useful when flex items should align in **inline flow**.

Example (User-Provided Code)

```
<style>
  .container {
    display: flex; /* OR use display: inline-flex */
  }
</style>

<div class="container">
  <div class="box">box-1</div>
  <div class="box">box-2</div>
  <div class="box">box-3</div>
</div>
```

- display: flex; → The container **expands** to full width.
- display: inline-flex; → The container **shrinks** to fit content.

Related Example (Real-World Use Case: Button Group Layout)

Using **inline-flex** to keep buttons aligned in a row.

```
<style>
  .button-group {
    display: inline-flex;
    gap: 10px;
  }

  .button {
    padding: 10px 20px;
    background-color: #3498db;
    color: white;
    border: none;
    cursor: pointer;
  }
</style>

<div class="button-group">
  <button class="button">Home</button>
  <button class="button">About</button>
  <button class="button">Contact</button>
</div>
```

- display: inline-flex; → The buttons stay in a **single row** without taking full width.

Common Mistakes & Fixes

1. **Using both display: flex; and display: inline-flex; together**
   - ❌ **Mistake:**

```
.container {
  display: flex;
  display: inline-flex;
}
```

    - The **last applied value** (inline-flex) will override the previous one (flex).

- ✅ **Fix:**
  - Use **only one** display value:

```
.container {
   display: flex; /* OR display: inline-flex */
}
```

2. **Expecting inline-flex to behave like inline-block**
   - ❌ **Mistake:**

```
.element {
   display: inline-block;
}
```

  - inline-block does not use **flexbox properties**.
- ✅ **Fix:**
  - Use inline-flex for inline alignment with **flexbox features**.

Flex-Direction

Definition

- flex-direction controls the **direction** in which flex items are arranged inside a flex container.
- It determines whether items **flow horizontally (row) or vertically (column)**.
- The four possible values are:
  - row → Items are placed from left to right (default).
  - row-reverse → Items are placed from right to left.
  - column → Items are placed from top to bottom.
  - column-reverse → Items are placed from bottom to top.

Example (User-Provided Code)

```
<style>
   .container {
      display: flex;
      flex-direction: row; /* Change this value to row-reverse, column, or
column-reverse */
   }
```

```
  .box {
    width: 50px;
    height: 50px;
    background-color: lightblue;
    margin: 5px;
    display: flex;
    justify-content: center;
    align-items: center;
  }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

Related Example (Real-World Use Case: Navigation Bar)

A **website navigation menu** can use flex-direction to arrange menu items in a **horizontal or vertical layout**.

```
<style>
  .navbar {
    display: flex;
    flex-direction: row; /* Change to column for vertical menu */
    background-color: #333;
    padding: 10px;
  }

  .navbar a {
    color: white;
    text-decoration: none;
    padding: 10px 20px;
  }
</style>
```

```
<div class="navbar">
  <a href="#">Home</a>
  <a href="#">About</a>
  <a href="#">Services</a>
  <a href="#">Contact</a>
</div>
```

- flex-direction: row; → Displays links **horizontally**.
- flex-direction: column; → Displays links **vertically** (good for mobile menus).

Common Mistakes & Fixes

1. **Using multiple flex-direction values at the same time**
   - ❌ **Mistake:**

```
flex-direction: row;
flex-direction: column;
```

     - The last value (column) **overrides** the first one (row).
   - ✅ **Fix:**
     - Use only **one** flex-direction property at a time:

```
flex-direction: row;
```

2. **Not setting display: flex; on the container**
   - ❌ **Mistake:**

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

     - flex-direction **won't work** if display: flex; is missing.
   - ✅ **Fix:**
     - Always apply display: flex; before using flex-direction:

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

3. **Forgetting to adjust item widths/heights when switching to column layout**
    - ❌ **Mistake:**

```
.container {
   display: flex;
   flex-direction: column;
}
.box {
   width: 50px;
}
```

        - Items might **not align properly** in a column layout.
    - ✅ **Fix:**
        - Use width: 100% for a column layout:

```
.box {
   width: 100%;
}
```

Here's your common stylesheet in **code format** for all upcoming subtopics:

```
body {
  background-color: #ecf0f1;
  padding: 50px;
  color: #2c3e50;
}

h2 {
  font-size: 50px;
}

.container {
  border: 5px solid #2c3e50;
  padding: 10px;
  margin-top: 50px;
}
```

```
.box {
  padding: 25px;
  margin: 10px;
  font-size: 50px;
  font-weight: bold;
  background-color: #e74c3c;
  color: white;
}
```

I'll use this format for all subtopics without extra explanation.

Flex-Wrap

Definition

- flex-wrap controls whether **flex items** should stay in a **single line** or wrap into **multiple lines** when they exceed the container's width.
- It helps manage layouts in **responsive designs** where items need to adjust dynamically.
- The three possible values are:
    - nowrap → **Default**. All items stay in a single line, even if they overflow.
    - wrap → Items move to a **new line** if they don't fit.
    - wrap-reverse → Similar to wrap, but the **new row appears above** the previous row.

Example (User-Provided Code)

```
<style>
  .container {
    width: 300px;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap; /* Change to nowrap or wrap-reverse */
  }
```

```
  .box {
    width: 80px;
    height: 50px;
    background-color: lightblue;
    margin: 5px;
    display: flex;
    justify-content: center;
    align-items: center;
  }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

Related Example (Real-World Use Case: Image Gallery Grid)

A **responsive image gallery** that wraps images into multiple rows based on screen width.

```
<style>
  .gallery {
    display: flex;
    flex-wrap: wrap;
    gap: 10px;
  }

  .gallery img {
    width: 150px;
    height: 100px;
  }
</style>

<div class="gallery">
  <img src="img1.jpg" alt="Image 1">
```

```
  <img src="img2.jpg" alt="Image 2">
  <img src="img3.jpg" alt="Image 3">
  <img src="img4.jpg" alt="Image 4">
  <img src="img5.jpg" alt="Image 5">
</div>
```

- flex-wrap: wrap; → Ensures images **move to the next row** when the container is full.

Common Mistakes & Fixes

1. **Using multiple flex-wrap values at the same time**
   - ❌ **Mistake:**

```
flex-wrap: nowrap;
flex-wrap: wrap;
flex-wrap: wrap-reverse;
```

   - ✅ **Fix:**
     - Use only **one** flex-wrap property at a time:

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

2. **Not setting a width for the container**

   - ❌ **Mistake:**

```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

     - Without a width, items might **not wrap correctly**.
   - ✅ **Fix:**
     - Set a fixed or percentage width:

```
.container {
   width: 300px;
   display: flex;
   flex-wrap: wrap;
}
```

3. **Forgetting to adjust item sizes when wrapping**
   - ❌ **Mistake:**

```
.box {
   width: auto;
}
```

     - Items may **not align properly** when wrapping.
   - ✅ **Fix:**
     - Set a proper width for flex items:

```
.box {
   width: 80px;
}
```

Flex-Flow

Justify-Content

Definition

- justify-content is a CSS property used to **align flex items along the main axis** (horizontal by default).
- It determines how items are **distributed inside the flex container** when there is extra space.
- Works only when display: flex; is applied.

Key Values of **justify-content**

- **flex-start** → Items align at the **beginning** of the container. *(default)*
- **flex-end** → Items align at the **end** of the container.

- **center** → Items are centered **horizontally** in the container.
- **space-between** → Items are spaced **evenly**, with no gap at the edges.
- **space-around** → Items have **equal space** around them.
- **space-evenly** → Items are spaced **equally**, including at the edges.

Example (User-Provided Code)

```
<style>
  .container {
    width: 1000px;
    display: flex;
    justify-content: space-between; /* Change to other values to see effects */
  }

  .box {
    width: 80px;
    height: 50px;
    background-color: lightblue;
    margin: 5px;
    display: flex;
    justify-content: center;
    align-items: center;
  }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

Related Example (Real-World Use Case: Navigation Bar Alignment)

Using justify-content to control navigation bar item alignment.

```
<style>
  .navbar {
    display: flex;
    justify-content: space-between; /* Adjust to center, flex-end, etc. */
    background-color: #333;
    padding: 10px;
  }

  .navbar a {
    color: white;
    text-decoration: none;
    padding: 10px 20px;
  }
</style>

<div class="navbar">
  <a href="#">Home</a>
  <a href="#">About</a>
  <a href="#">Services</a>
  <a href="#">Contact</a>
</div>
```

- `justify-content: space-between;` → Aligns items **evenly with no gap at edges**.
- `justify-content: center;` → Aligns items in the **middle** of the navbar.

Common Mistakes & Fixes

1. **Using multiple `justify-content` values at the same time**
   - ❌ **Mistake:**

```
justify-content: flex-start;
justify-content: space-between;
justify-content: center;
```

   - ✅ **Fix:**

        - Use only **one** justify-content property at a time:

```
justify-content: center;
```

2. **Forgetting to set display: flex; on the container**
    - ❌ **Mistake:**

```
.container {
  justify-content: center;
}
```

        - justify-content **won't work** without display: flex;.
    - ✅ **Fix:**

```
.container {
  display: flex;
  justify-content: center;
}
```

3. **Misunderstanding space distribution values**
    - ❌ **Mistake:** Expecting space-between to create equal spacing everywhere.
    - ✅ **Fix:**
        - Use space-evenly if you want **equal spacing including edges**.

Align-Content

Definition

- align-content controls the **vertical alignment** of multiple flex lines **along the cross-axis**.
- It works only when **flex-wrap: wrap;** is applied, meaning multiple rows are present.
- Determines how space is distributed **between and around flex rows**.

Key Values of align-content

- **flex-start** → Rows are packed at the **top** of the container.
- **flex-end** → Rows are packed at the **bottom** of the container.
- **center** → Rows are **centered** within the container.
- **space-between** → Rows are spread out with **no extra space at the top or bottom**.
- **space-around** → Rows have **equal space** around them.
- **stretch** *(default)* → Rows **fill the container's height**.

Example (User-Provided Code)

```
<style>
  .container {
    width: 300px;
    height: 600px;
    display: flex;
    flex-wrap: wrap;
    align-content: space-between; /* Change this value to see different effects */
  }

  .box {
    width: 80px;
    height: 50px;
    background-color: lightblue;
    margin: 5px;
    display: flex;
    justify-content: center;
    align-items: center;
  }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

Related Example (Real-World Use Case: Vertical Card Alignment in a Grid)

A **dashboard layout** with multiple content blocks that should align within the grid.

```
<style>
  .dashboard {
    display: flex;
    flex-wrap: wrap;
    align-content: center; /* Adjust to flex-start, flex-end, etc. */
    height: 600px;
    background-color: #f4f4f4;
  }

  .card {
    width: 100px;
    height: 120px;
    background-color: #3498db;
    color: white;
    display: flex;
    justify-content: center;
    align-items: center;
    margin: 5px;
  }
</style>

<div class="dashboard">
  <div class="card">Card 1</div>
  <div class="card">Card 2</div>
  <div class="card">Card 3</div>
  <div class="card">Card 4</div>
  <div class="card">Card 5</div>
</div>
```

- `align-content: center;` → Centers the rows **vertically** in the container.
- `align-content: space-around;` → Spaces out the rows **evenly**.

Common Mistakes & Fixes

1. **Using align-content without flex-wrap: wrap;**
    - ❌ **Mistake:**

```
.container {
   display: flex;
   align-content: center;
}
```

        - align-content **won't work** because flex items are in a **single line**.
    - ✅ **Fix:**
        - Ensure flex-wrap: wrap; is applied:

```
.container {
   display: flex;
   flex-wrap: wrap;
   align-content: center;
}
```

2. **Using multiple align-content values at the same time**
    - ❌ **Mistake:**

```
align-content: flex-start;
align-content: space-between;
```

    - ✅ **Fix:**
        - Use only **one** align-content value at a time:

```
align-content: space-between;
```

3. **Confusing align-content with align-items**
    - ❌ **Mistake:**

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

        - align-items aligns **individual items**, not multiple flex rows.
    - ✅ **Fix:**

- Use align-content for **row alignment**:

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

Flex Order

Definition

- order is a CSS property used to **change the visual arrangement** of flex items without modifying the HTML structure.
- By default, all flex items have an **order value of 0**, meaning they appear in the same order as in the HTML.
- A **higher order value** moves an item **further to the end**, while a **lower value** moves it **closer to the start**.

Key Points of order Property

- **Default value:** order: 0; (Items appear as they are in HTML).
- **Positive values:** Move items **towards the end** of the flex container.
- **Negative values:** Move items **towards the start** of the flex container.
- **Same order value:** If two items have the same order, they follow their **HTML structure** order.

Example (User-Provided Code)

```
<style>
  .container {
    width: 800px;
    display: flex;
  }

  .box:nth-child(1) { order: 5; }
  .box:nth-child(2) { order: 1; }
  .box:nth-child(3) { order: 4; }
  .box:nth-child(4) { order: 2; }
  .box:nth-child(5) { order: 3; }
```

```
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

- order: 1; → Moves **closer to the start**.
- order: 5; → Moves **further to the end**.

Related Example (Real-World Use Case: Changing Button Order in a Form)

Using order to adjust button placement in a **form layout**.

```
<style>
  .form-container {
    display: flex;
    gap: 10px;
  }

  .btn {
    padding: 10px 20px;
    background-color: #3498db;
    color: white;
    border: none;
    cursor: pointer;
  }

  .submit { order: 2; } /* Submit button appears later */
  .reset { order: 1; }  /* Reset button appears first */
</style>

<div class="form-container">
  <button class="btn reset">Reset</button>
  <button class="btn submit">Submit</button>
</div>
```

- order: 1; → Reset button appears **before** Submit.
- order: 2; → Submit button moves to the **end**.

Common Mistakes & Fixes

1. **Using multiple order values on the same element**
    - ❌ **Mistake:**

```
.box {
  order: 3;
  order: 1;
}
```

    - ✅ **Fix:**
        - Use only **one** order value per element:

```
.box {
  order: 1;
}
```

2. **Expecting order to work without display: flex;**
    - ❌ **Mistake:**

```
.container {
  order: 2;
}
```

        - order **won't work** unless display: flex; is set on the parent.
    - ✅ **Fix:**
        - Always apply display: flex; first:

```
.container {
  display: flex;
}
```

3. **Confusing order with z-index**

- ❌ **Mistake:**

```
.box {
  z-index: 3;
}
```

    - z-index controls **stacking (depth)**, not item position.
- ✅ **Fix:**
    - Use order for **positioning within flex containers**.

Align-Self

Definition

- align-self is used to **override the alignment** of an individual flex item along the **cross-axis** (vertical by default).
- It allows a specific item to **differ from other items**, even if align-items is set on the container.

Key Values of align-self

- **auto** *(default)* → The item follows the container's align-items value.
- **flex-start** → Aligns the item at the **top** of the flex container.
- **flex-end** → Aligns the item at the **bottom** of the flex container.
- **center** → Centers the item **vertically**.
- **stretch** → Stretches the item to **fill the full height** of the container.
- **baseline** → Aligns the item **based on text baselines**.

Example (User-Provided Code)

```
<style>
  .container {
    width: 350px;
    height: 600px;
    display: flex;
    flex-wrap: wrap;
```

```
    align-items: flex-start;
  }

  .box:nth-child(1) { align-self: stretch; }
  .box:nth-child(5) { align-self: flex-end; }
  .box:nth-child(3) { align-self: center; }
  .box:nth-child(2) { align-self: flex-start; }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

- align-self: stretch; → Box 1 stretches to fill the full height.
- align-self: flex-end; → Box 5 moves to the bottom.
- align-self: center; → Box 3 is centered vertically.
- align-self: flex-start; → Box 2 moves to the top.

Related Example (Real-World Use Case: Adjusting Button Alignment in a Sidebar Layout)

Using align-self to position buttons **independently** inside a sidebar.

```
<style>
  .sidebar {
    display: flex;
    flex-direction: column;
    height: 400px;
    background-color: #f4f4f4;
    padding: 20px;
  }

  .button {
    padding: 10px 20px;
    background-color: #3498db;
```

```
    color: white;
    border: none;
    cursor: pointer;
  }

  .logout { align-self: flex-end; } /* Moves the logout button to the bottom */
</style>

<div class="sidebar">
  <button class="button">Dashboard</button>
  <button class="button">Settings</button>
  <button class="button logout">Logout</button>
</div>
```

- align-self: flex-end; → Moves the **Logout** button to the bottom while keeping others at the top.

Common Mistakes & Fixes

1. **Using align-self without display: flex;**
    - ❌ **Mistake:**

```
.box {
  align-self: center;
}
```

        - align-self **won't work** unless the parent has display: flex;.
    - ✅ **Fix:**

```
.container {
  display: flex;
}
```

2. **Expecting align-self to work on all items together**
    - ❌ **Mistake:**

```
.container {
```

```
    align-self: center;
}
```

        - align-self **only works on individual items**, not the container.
    - ✅ **Fix:**
        - Apply it to specific flex items instead:

```
.box:nth-child(2) {
    align-self: center;
}
```

3. **Forgetting to set align-items first**

    - ❌ **Mistake:**
        - Using align-self alone without a base alignment.
    - ✅ **Fix:**
        - Set a default alignment using align-items, then adjust individual items with align-self.

Flex-Grow

Definition

- flex-grow determines **how much an item should expand** to fill available space inside a flex container.
- The default value is 0, meaning items **won't grow** beyond their initial size.
- Items with a **higher flex-grow value** take up **more space** than those with a lower value.

Key Points of flex-grow

- **Default value:** flex-grow: 0; (Items do not expand).
- **Higher value:** The item grows **more relative to others**.
- **Equal values:** All items grow **at the same rate**.
- **Works only with display: flex;** on the parent container.

Example (User-Provided Code)

```
<style>
  .container {
    width: 1000px;
    display: flex;
  }

  .box {
    margin: 1px;
    flex-grow: 1; /* All boxes grow equally */
  }

  .box:nth-child(2) {
    flex-grow: 3; /* Box 2 grows 3x more than others */
  }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

- flex-grow: 1; → All boxes **grow equally** by default.
- flex-grow: 3; → Box 2 **takes three times more space** than others.

Related Example (Real-World Use Case: Responsive Navigation Bar)

Using flex-grow to make **some menu items larger than others**.

```
<style>
  .navbar {
    display: flex;
    background-color: #333;
  }
```

```
  .nav-item {
    flex-grow: 1;
    padding: 10px;
    color: white;
    text-align: center;
  }

  .logo {
    flex-grow: 2; /* Logo takes up more space */
  }
</style>

<div class="navbar">
  <div class="nav-item logo">LOGO</div>
  <div class="nav-item">Home</div>
  <div class="nav-item">About</div>
  <div class="nav-item">Contact</div>
</div>
```

- flex-grow: 2; → The **logo expands more** than other items.
- flex-grow: 1; → Other menu items grow equally.

Common Mistakes & Fixes

1. **Using flex-grow without display: flex;**
   - ❌ **Mistake:**

```
.box {
  flex-grow: 1;
}
```

     - flex-grow **won't work** unless the parent container has display: flex;.
   - ✅ **Fix:**

```
.container {
  display: flex;
}
```

2. **Expecting flex-grow to work with fixed widths**
    - ❌ **Mistake:**

```
.box {
   width: 100px;
   flex-grow: 1;
}
```

        - The width **restricts growth**, preventing flexible expansion.
    - ✅ **Fix:**
        - Avoid setting fixed widths:

```
.box {
   flex-grow: 1;
}
```

3. **Using negative values for flex-grow**

    - ❌ **Mistake:**

```
.box {
   flex-grow: -1;
}
```

        - flex-grow **does not accept negative values**.
    - ✅ **Fix:**
        - Use only 0 or positive values.

Flex-Shrink

Definition

- flex-shrink controls **how much an item should shrink** when there isn't enough space in the flex container.
- The default value is 1, meaning items **shrink at an equal rate** when necessary.
- A **higher flex-shrink value** causes an item to **shrink more** compared to others.
- A **value of 0** prevents the item from shrinking.

Key Points of flex-shrink

- **Default value:** flex-shrink: 1; (All items shrink equally).
- **Higher values:** Items shrink **more** than those with a lower value.
- **Zero (0) value:** The item **won't shrink**, even if the container is too small.
- Works only when **items exceed the available space** in the flex container.

Example (User-Provided Code)

```
<style>
  .container {
    width: 400px;
    display: flex;
    padding: 0;
  }

  .box {
    margin: 1px;
    padding: 1px;
    width: 200px;
    text-align: center;
    flex-shrink: 1; /* All boxes shrink equally */
  }

  .box:nth-child(2) {
    flex-shrink: 2; /* Box 2 shrinks twice as much */
  }
</style>

<div class="container">
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
</div>
```

- flex-shrink: 1; → All boxes shrink **equally** by default.

- flex-shrink: 2; → Box 2 **shrinks twice as much** as the others.

Related Example (Real-World Use Case: Responsive Table Layout)

Using flex-shrink to **control column width** when space is limited.

```
<style>
  .table {
    display: flex;
    width: 500px;
    border: 2px solid black;
  }

  .column {
    flex: 1;
    padding: 10px;
    text-align: center;
    background-color: #f4f4f4;
    border-right: 1px solid black;
    flex-shrink: 1;
  }

  .wide-column {
    flex-shrink: 0; /* Prevents shrinking */
  }
</style>

<div class="table">
  <div class="column">Name</div>
  <div class="column wide-column">Description</div>
  <div class="column">Price</div>
</div>
```

- flex-shrink: 1; → Columns shrink equally.
- flex-shrink: 0; → The **description column stays the same size** and doesn't shrink.

Common Mistakes & Fixes

1. **Using flex-shrink without display: flex;**
    - ❌ **Mistake:**

```
.box {
   flex-shrink: 1;
}
```

        - flex-shrink **won't work** unless the parent container has display: flex;.
    - ✅ **Fix:**

```
.container {
   display: flex;
}
```

2. **Setting flex-shrink: 0; when items should adjust**
    - ❌ **Mistake:**

```
.box {
   flex-shrink: 0;
}
```

        - This prevents shrinking, **causing overflow issues**.
    - ✅ **Fix:**
        - Allow shrinking for better responsiveness:

```
.box {
   flex-shrink: 1;
}
```

3. **Expecting flex-shrink to work when there is enough space**
    - ❌ **Mistake:**
        - flex-shrink only works when items **exceed the container width**.
    - ✅ **Fix:**
        - Test it by **reducing the container's width** to see shrinking behavior.

Task: Responsive Product Card Layout

Scenario:

You need to create a **responsive product card layout** that dynamically adjusts to different screen sizes. The layout should include:

- **Product cards** arranged in a **flexbox container**.
- Items should **resize (flex-grow)** when extra space is available.
- Items should **shrink (flex-shrink)** when space is limited.
- Items should be aligned using **justify-content, align-content, and align-self**.
- The order of items should be controlled using **flex-order**.

Task Requirements:

- Create a **flexbox container** that holds multiple **product cards**.
- Use **flex-grow** to allow cards to expand and **flex-shrink** to shrink when necessary.
- Use **justify-content** to arrange items in the center.
- Use **align-content** to control the vertical alignment of rows.
- Use **align-self** to change the alignment of individual items.
- Use **order** to rearrange specific product cards dynamically.

Solution Code:

HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product Card Layout</title>
  <link rel="stylesheet" href="base.css">
  <style>
```

```
.product-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    align-content: center;
    width: 90%;
    margin: auto;
    padding: 20px;
}

.product-card {
    width: 200px;
    height: 300px;
    background-color: white;
    border: 2px solid #2c3e50;
    padding: 10px;
    margin: 10px;
    text-align: center;
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    align-items: center;
    flex-grow: 1;
    flex-shrink: 1;
}

.product-card:nth-child(2) {
    flex-grow: 2; /* Second card grows twice as much */
    order: -1; /* Moves it to the first position */
}

.product-card:nth-child(4) {
    flex-shrink: 2; /* Fourth card shrinks twice as much */
    align-self: flex-end; /* Moves to the bottom */
}

.product-card img {
    width: 100%;
    height: auto;
}
```

```
      .button {
         background-color: #e74c3c;
         color: white;
         padding: 10px;
         border: none;
         cursor: pointer;
         width: 100%;
         align-self: stretch;
      }
   </style>
</head>
<body>

   <h2>Responsive Product Layout</h2>

   <div class="product-container">
      <div class="product-card">
         <img src="product1.jpg" alt="Product 1">
         <h3>Product 1</h3>
         <p>$10.00</p>
         <button class="button">Buy Now</button>
      </div>

      <div class="product-card">
         <img src="product2.jpg" alt="Product 2">
         <h3>Product 2</h3>
         <p>$15.00</p>
         <button class="button">Buy Now</button>
      </div>

      <div class="product-card">
         <img src="product3.jpg" alt="Product 3">
         <h3>Product 3</h3>
         <p>$20.00</p>
         <button class="button">Buy Now</button>
      </div>

      <div class="product-card">
         <img src="product4.jpg" alt="Product 4">
         <h3>Product 4</h3>
         <p>$25.00</p>
```

```
        <button class="button">Buy Now</button>
      </div>
    </div>

  </body>
  </html>
```

Explanation of the Code:

- **display: flex;** → The .product-container uses flexbox to arrange product cards.
- **flex-wrap: wrap;** → Cards move to the next row if needed.
- **justify-content: space-around;** → Distributes cards with even spacing.
- **align-content: center;** → Centers rows vertically.
- **flex-grow: 2;** (on 2nd card) → Expands this card **twice as much**.
- **flex-shrink: 2;** (on 4th card) → Shrinks more when space is limited.
- **order: -1;** (on 2nd card) → Moves it **before others**.
- **align-self: flex-end;** (on 4th card) → Moves it **lower** in the container.

Expected Behavior:

✔ The second product card **expands more** and appears **first** due to order: -1;.
✔ The fourth product card **shrinks more** and moves **down** due to align-self: flex-end;.
✔ All product cards **adjust dynamically** as the screen size changes.

Task Challenge for You:

1️⃣ Modify the flex-grow value for different cards and observe the effect.
2️⃣ Change justify-content to center or space-between to test alignment.
3️⃣ Set align-self on another product card and see how it behaves.
