CSS Combinators:

- The CSS combinators represent the relationship between two selectors.

- The CSS selectors are the patterns that can be used for styling the particular HTML element. Sometimes, it is possible that there is more than one simple selector, and to combine the multiple simple selectors, we use combinators.

Why use CSS Combinator ?

- Learning about combinators makes you better at writing CSS and helps you to avoid excess CSS code.
- Combinators can also help you pinpoint the section or part of HTML you want to style with high accuracy because they are based on the relationship between the selectors.

Types of Combinators:

The combinators are of 4 types, which are given below:

1. `Descendant selector ( ) (space)`
2. `Child selector (>)`
3. `General sibling selector (~)`
4. `Adjacent sibling selector (+)`

1. Descendant Selector: (space)

- Descendant Meaning: A person who is related to you and who lives after you, such as your child or grandchild.
- Selects an element that is a descendant of another element. It selects all elements that are inside another specified element, no matter how deeply nested.

Syntax:

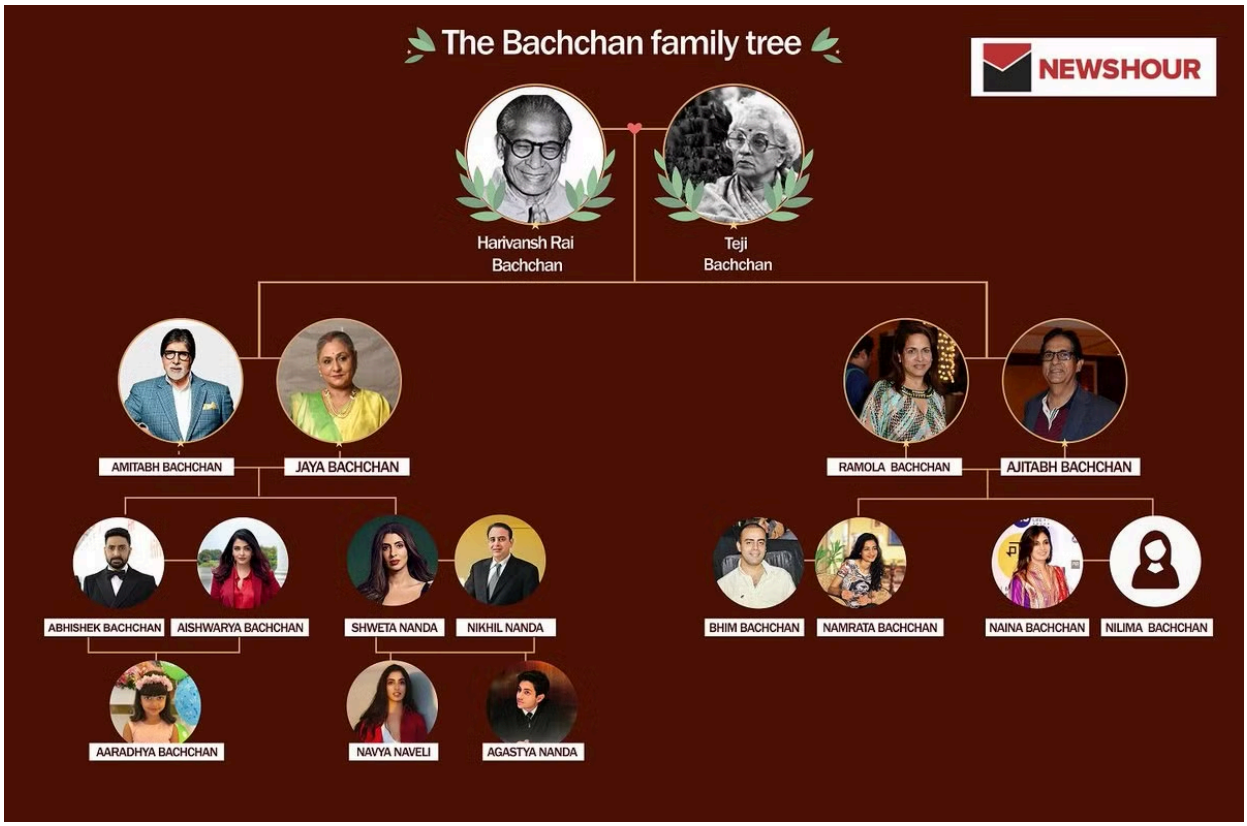
selector1 selector2 selector3... {

// style properties

}

Example:

```
Div p {
 /* Selects all <p> elements that are descendants of <div> elements */
}
```



- Look at Amitabh's family tree
    - Number of Descendants for Harivansh Rai Bachan - 15
    - Number of Descendants for Amitabh Bachan - 7

Example:

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
/* It will target all the <p> which is the decendent of <div> tag */

    div p {
      color: red;
    }
  </style>



</head>

<body>

  <div>
    <p>Lorem ipsum dolor sit amet.1</p>
    <div>
      <p>Lorem ipsum dolor sit amet.2</p>
      <div>
        <p>Lorem ipsum dolor sit amet.3</p>
      </div>
    </div>
  </div>

  <p>Lorem ipsum dolor sit amet.4</p>


</body>

</html>
```

2. Child Selector:

- The child selector uses the greater than sign (>) to separate the elements. The child selector is used when we want to apply the styling properties to the immediate child/children of the particular HTML element.

- This combinator is quite strict than the descendant selector and the styling properties are acquired only when the second selector is the direct child of the first one.

- Look at Amitabh family tree
    - Number of Childs for Harivansh Rai bachan - 2
    - Number of Childs for Amitabth Bachan - 2

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
/* It will target all the <p> which is the direct children of the <div> tag */
    .d1>p {
      color: red;
    }
  </style>


</head>

<body>

  <div class="d1">
    <p>Lorem ipsum dolor sit amet.1</p>
```

```
    <div>
      <p>Lorem ipsum dolor sit amet.2</p>
      <div>
        <p>Lorem ipsum dolor sit amet.3</p>
      </div>
    </div>
  </div>

  <p>Lorem ipsum dolor sit amet.4</p>


</body>

</html>
```

3. Adjacent Sibling Selector: (+)

- Adjacent / Immediate sibling Selector in CSS
- The adjacent sibling selector is used when we want to apply the CSS property or styling to the adjacent sibling of any element.
- The siblings should have the same parent element and also the second element must be the immediate follower of the first element.
- The selectors are separated by adding the (+) sign between the separators.

- Look at Amitabh's family tree
    - Number of immediate siblings for Amitabh Bachan - 1 (Ajitabh Bachan)
    - Number of immediate siblings for Abhishek Bachan - 1 (Sweta Nanda)
    - Number of immediate siblings for Bhim Bachan - 1 (Namrata Bachan)
    - Number of immediate siblings for Navya Naveli - 1
- Remember that siblings will be of the same generation

Syntax:

```
h2+p {
/* Selects <p> elements that are immediately preceded by <h2> elements */
  }
```

Example:

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    /* It will target all the <p> which is immediately preceded by the <div> tag */
    .d1+p {
      color: red;
    }
  </style>



</head>

<body>

  <div class="d1">
    <p>Lorem ipsum dolor sit amet.1</p>
    <div>
      <p>Lorem ipsum dolor sit amet.2</p>
      <div>
        <p>Lorem ipsum dolor sit amet.3</p>
      </div>
    </div>
  </div>

  <p>Lorem ipsum dolor sit amet.4</p>
  <p>Lorem ipsum dolor sit amet.5</p>
```

```
</body>

</html>
```

4. General Sibling Selector: (~)

- **Sibling meaning:** A brother or sister from the same parent
- This selector is used when we have to set the styling properties of elements with the same parent element. This selector can be separated by adding the (~) sign between them.

  - Number of general siblings for Amitabh Bachan - 1 (Ajitabh Bachan)
  - Number of general siblings for Abhishek Bachan - 1 (Sweta Nanda)

Syntax:

```
h2~p {
      /* Selects <p> elements that are siblings of <h2> elements */
  }
```

Example:

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    /* It will target all the <p> which are the siblings of <div> tag */
    .d1~p {
      color: red;
    }
```

```
  </style>


</head>

<body>

  <div class="d1">
    <p>Lorem ipsum dolor sit amet.1</p>
    <div>
      <p>Lorem ipsum dolor sit amet.2</p>
      <div>
        <p>Lorem ipsum dolor sit amet.3</p>
      </div>
    </div>
  </div>

  <p>Lorem ipsum dolor sit amet.4</p>
  <p>Lorem ipsum dolor sit amet.5</p>

</body>

</html>
```

Note: These combinators allow you to create more specific and targeted CSS selectors, making it easier to style elements based on their relationships with other elements in the HTML structure.

Attribute selectors:

It is possible to style HTML elements that have specific attributes or attribute values.

The `[attribute]` selector is used to select elements with a specified attribute.

Example1: The following example selects all <p> elements with a **class** attribute:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        p[class] {
            background-color: yellow;
        }
    </style>

</head>

<body>



    <p>This is the paragraph 1</p>
    <p class="a">This is the paragraph 2</p>
    <p class="a">This is the paragraph 3</p>
    <p class="b">This is the paragraph 4</p>

</body>

</html>
```

The [attribute="value"] selector is used to select elements with a specified attribute and value.

Example2: The following example selects all <p> elements with a **class="b"** attribute:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        p[class="b"] {
            background-color: yellow;
        }
    </style>

</head>

<body>


    <p>This is the paragraph 1</p>
    <p class="a">This is the paragraph 2</p>
    <p class="a">This is the paragraph 3</p>
    <p class="b">This is the paragraph 4</p>

</body>

</html>
```

CSS pseudo-classes:

A CSS pseudo-class is a keyword added to a **selector** that specifies a special state of the selected element(s).

Pseudo-classes are used in CSS to define the special states of an element. They allow you to style elements based on their **state**, **position** in the DOM, or **user interaction**. This helps to apply styles dynamically without needing to add or remove classes via JavaScript.

It can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

Syntax:

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

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked `<input>` element
:disabled	input:disabled	Selects every disabled `<input>` element
:empty	p:empty	Selects every `<p>` element that has no children
:enabled	input:enabled	Selects every enabled `<input>` element
:first-child	p:first-child	Selects every `<p>` element that is the first child of its parent
:first-of-type	p:first-of-type	Selects every `<p>` element that is the first `<p>` element of its parent
:focus	input:focus	Selects the `<input>` element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects `<input>` elements with a value within a specified range
:invalid	input:invalid	Selects all `<input>` elements with an invalid value

:lang(language)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"
:last-child	p:last-child	Selects every <p> element that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute specified
:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

Dynamic Styling: Pseudo-classes allow you to apply styles dynamically based on **user actions** (like :hover or :focus) or element states (like :checked or :disabled).

Positioning: Pseudo-classes such as :first-child, :nth-child(n), :last-child, and :only-child let you style elements based on their **position** in the DOM.

Form Control: Pseudo-classes like :valid, :invalid, :required, and :optional are helpful for styling form inputs depending on their state or validity.

Examples:

1. :hover

```
h1:hover{
        color: red;
        text-decoration: underline;
    }
```

This changes the color of <h1> to red and adds an underline when the user hovers over it.

2. :focus

```
input:focus {
            border: 3px solid blue;
            background-color: beige;
```

```
        }
```

This applies a blue border and a beige background to an input field when it receives focus.

3. :first-child

- Targets the first child of a parent element.

Example1:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <style>

        p:first-child {

           color: red;

        }

    </style>

</head>

<body>

    <p>Para1</p>

    <p>Para2</p>
```

```
    <div>
        <p>Para3</p>
        <p>Para4</p>
    </div>
    <p>para5</p>
</body>
</html>
```

Here para1(first child of <body> and para3 first child of will be selected)

Guess the Output: Which p will be selected ?

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test</title>
    <style>
        p:first-child {
            color: red;
        }
```

```
    </style>
</head>

<body>
    <p>Para1</p>
    <p>Para2</p>
    <div>
        <p>Para 3</p>
        <p>Para 4</p>
        <p>Para 5</p>
        <p>Para 6</p>
    </div>
    <div>
        <h3>Heading</h3>
        <p>Para 7</p>
        <p>Para 8</p>
        <p>Para 9</p>
        <p>Para 10</p>
    </div>
    <div>
        <p>Para 11</p>
```

```
        <p>Para 12</p>

        <p>Para 13</p>

        <p>Para 14</p>

    </div>

    <p>Para15</p>

    <p>Para16</p>

</body>

</html>
```

4. first-of-type

The `:first-of-type` selector matches every element that is the first child, of a particular type, of its parent.

Tip: This is the same as :nth-of-type(1).

In the above example if we use **:first-of-type** in place of **:first-child** then Para7 will also be selected.

5. :last-child

- Selects the last child element within its parent.

Example:

```
<!DOCTYPE html>
```

```html
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        p:last-child {
            color: red;
        }
    </style>
</head>

<body>
    <p>Para1</p>
    <p>Para2</p>
    <div>
        <p>Para3</p>
        <p>Para4</p>
    </div>
    <p>para5</p>
```

```
</body>

</html>
```

Note: Here the last element inside the <body> tag is the automatically injected <script> tag by the live-server, we can see it by inspecting the browser and can remove it by right clicking it to select the para5.

Example2: Guess the Output:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Test</title>

    <style>

        p:last-child {

            color: red;

        }

    </style>

</head>

<body>

    <p>Para1</p>

    <p>Para2</p>
```

```
<div>
    <p>Para 3</p>
    <p>Para 4</p>
    <p>Para 5</p>
    <p>Para 6</p>
</div>
<div>
    <p>Para 7</p>
    <p>Para 8</p>
    <p>Para 9</p>
    <p>Para 10</p>
    <h3>heading1</h3>
</div>
<div>
    <p>Para 11</p>
    <p>Para 12</p>
    <p>Para 13</p>
    <p>Para 14</p>
</div>
<p>Para15</p>
<p>Para16</p>
```

```
    <h3>heading2</h3>

</body>

</html>
```

6. :last-of-type

The `:last-of-type` selector matches every element that is the last child, of a particular type, of its parent.

Tip: This is the same as :nth-last-of-type(1).

In the above example if we use **:last-of-type** in place of **:last-child** then Para10 and para16 will also be selected.

7. :nth-child

- The `:nth-child(n)` selector matches every element that is the *n*th child of its parent.
- *n* can be a number, a keyword (odd or even).
-

Example:1

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test</title>
    <style>
        :nth-child(3) {
            color: red;
        }
    </style>
</head>
<body>
    <p>Para1</p>
    <p>Para2</p>
    <div>
        <p>Para 3</p>
        <p>Para 4</p>
    </div>
    <div>
        <p>Para 5</p>
        <p>Para 6</p>
        <h3>Heading1</h3>
        <p>Para 7</p>
    </div>
```

```
    <p>Para8</p>

    <p>Para9</p>

    <h3>Heading2</h3>

</body>

</html>
```

Here 3rd element of every parent will be selected, which contains para3 and para4 is the 3rd child of the body, where as heading1 is the 3rd child of 2nd

Example:2

```
   p:nth-child(3) {

            color: red;

    }
```

In the above example there is no any <p> which is the 3rd child of the any parent.

Example3:

```
 p:nth-child(2) {

            color: red;

 }
```

**In the above example para2 is the 2nd child of the <body> where as para4 is the 2nd child of 1st and para6 is the 2nd child of the 2nd **

Example4:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Test</title>

    <style>

        p:nth-child(odd) {

            color: red;

        }

    </style>

</head>

<body>

    <div>

        <p>Para1</p>

        <p>Para2</p>

    </div>

    <p>Para3</p>
```

```
    <p>Para4</p>
    <p>Para5</p>
    <p>Para6</p>
    <p>Para7</p>
    <div>
        <p>Para8</p>
        <p>Para9</p>
    </div>
    <p>Para10</p>
    <p>Para11</p>
</body>
</html>
```

Here para1, Para4, Para6, Para8, Para11 will the targeted.

8. nth-of-type:

The `:nth-of-type(n)` selector matches every element that is the *n*th child, of the same type (tag name), of its parent.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test</title>
    <style>
        p:nth-child(3) {
            color: red;
        }
    </style>
</head>
<body>
    <p>Para1</p>
    <p>Para2</p>
    <div>
        <p>Para 3</p>
        <p>Para 4</p>
    </div>
    <div>
```

```
        <p>Para 5</p>
        <p>Para 6</p>
        <h3>Heading1</h3>
        <p>Para 7</p>
    </div>
    <p>Para8</p>
    <p>Para9</p>
    <h3>Heading2</h3>
</body>
</html>
```

Note: In the above example non of the <p> element will be selected, because no <p> element which is the 3rd child of any parent.

But in place of **p:nth-child(3)** if we use **p:nth-of-type(3)** then Para7 and Para8 will be selected.

Because Para7 is the 3rd type <p> of the 2nd and Para8 will be the 3rd type of <p> of the <body>

9. only-child:

The `:only-child` selector matches every element that is the only child of its parent.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test</title>
    <style>
        p:only-child {
            color: red;
        }
    </style>
</head>
<body>
    <p>Para1</p>
    <p>Para2</p>
    <div>
        <p>Para 3</p>
        <h3>Heading1</h3>
    </div>
```

```
    <div>
        <p>Para4</p>
    </div>
    <p>Para5</p>
    <p>Para6</p>
    <h3>Heading2</h3>
</body>
</html>
```

Here Para4 will be selected.

10. only-of-type:

```
p:only-of-type {
            color: red;
        }
```

In the above example Para3 and Para4 will be selected.

11. target:

The `:target` selector can be used to style the current active target element.

It is used with the internal document links.

Example:

```
<!DOCTYPE html>

<html>

<head>

<style>

    :target{

        background-color: aqua;

        border: 2px solid green;

        color: antiquewhite;

    }

</style>

</head>

<body>


<h1>This is a heading</h1>

<a href="#news1">Jump to New content 1</a>

<a href="#news2">Jump to New content 2</a>

<p>Click on the links above and the :target selector highlight the current active HTML anchor.</p>

<p id="news1"> Paragraph 1: lorem500</p>

<p id="news2"> Paragraph2: lorem500</p>

</body>
```

```
</html>
```

Example2:

```
<!DOCTYPE html>
<html>
<head>
    <style>
         div {
            display: none;
        }
         :target {
            display: block;
        }
    </style>
</head>
<body>
    <a href="#link1">Link 1</a>
    <a href="#link2">Link 2</a>
    <a href="#link3">Link 3</a>
```

```
        <div id="link1">
            <h3>Content to Link 1</h3>
            <p>Hello World!</p>
        </div>

        <div id="link2">
            <h3>Content to Link 2</h3>
            <h4>Great success!</h4>
        </div>
        <div id="link3">
            <h3>Content to Link 3</h3>
            <p>Yeah!</p>
        </div>
</body>

</html>
```

12. :link, :visited, :active

Example:

```
<!DOCTYPE html>
<html>
```

```
<head>
    <style>
        /* unvisited link */
        a:link {
            color: purple;
        }

        /* visited link */
        a:visited {
            color: green;
        }

        /* mouse over link */
        a:hover {
            color: red;
        }

        /* selected link */
        a:active {
            color: yellow;
```

```
        }

    </style>

</head>

<body>

    <p>Mouse over and click the link: <a href="./b.html">Click Here to go
b.html</a></p>

</body>



</html>
```

Combining the pseudo-classes with the combinators:

pseudo-classes can be combined with the combinators also to create complex CSS rules that target elements based on their position, state, or relationship with other elements.

Example1:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>


    <style>

        .box p:hover{
            color: red;
```

```
            text-decoration: underline;
        }

    </style>

</head>
<body>

    <p>Para1</p>
    <p>Para2</p>

    <div class="box">

        <p>Para3</p>
        <p>Para4</p>

    </div>

    <div>
        <p>Para5</p>
        <p>Para6</p>
    </div>

    <p>Para7</p>

</body>
</html>
```

Example2:

```
  div>p:first-child {
            color: blue;
  }
```

Here selecting only the 1st child <p> element inside any

CSS pseudo-elements:

- **Pseudo-elements** are used to style specific parts of an element. Unlike pseudo-classes (which target elements based on their state or position),
- pseudo-elements allow you to style **parts of an element's content** or insert content into the page without modifying the HTML.
- Pseudo-elements are preceded by two colons (`::`) to differentiate them from pseudo-classes.

Syntax:

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

pseudo-elements	Example	Description
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::selection	::selection	Selects the portion of an element that is selected by a user
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element

Example:

```
<!DOCTYPE html>
<html>


<head>
    <style>
        ::selection {
            color: red;
            background: yellow;
```

```
        }


        h3::before {
            content: "(Added using Before)";
            color: purple;
        }


        h3::after {
            content: "(Added after)";
            color: blue;
        }


        h1::first-letter {
            color: green;
        }


        p::first-line {
            color: aquamarine;
            font-size: 20px;
        }
    </style>
</head>


<body>


    <h1>Select some text on this page:</h1>

    <h3>This is a paragraph.</h3>
    <p>
Lorem ipsum dolor sit amet consectetur adipisicing elit. Excepturi facilis nobis aperiam facere rem eligendi
minima blanditiis consequatur nostrum possimus, nulla obcaecati accusantium sequi provident molestias amet nam.
```

```
Earum, fugiat.
</p>

</body>

</html>
```

CSS Flex-box

Introduction:

- Flexbox was introduced in 2009 as a new layout system, with the goal of helping us build responsive web pages and organise our elements easily, and since then, it's gained more and more attention. It turns out it's now used as the main layout system for modern web pages.
- Flexbox is a one-dimensional layout system that we can use to create a row or a column axis layout.
- It makes our life easier to design and build responsive web pages without having to use tricky hacks and a lot of margins and padding properties in our CSS code.

The following simple layout designs are either difficult or impossible to achieve with margin, padding, in any kind of convenient, flexible way:

- Vertically centering a block of content inside its parent.
- Making all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Making all columns in a multiple-column layout adopt the same height even if they contain a different amount of content.

Flex mimics how we think about layouts (visually) .. elements to be horizontal, centered, spaced equally vs elements vertically - flexbox automatically calculates the

distance n how elements should be placed n move when needed .... compare this to inline etc where you need to worry about the physical dimensions of the screen.

Following are the notable features of Flexbox layout :

- **Direction −** You can arrange the items on a web page in any direction such as left to right, right to left, top to bottom, and bottom to top.
- **Order −** Using Flexbox, you can rearrange the order of the contents of a web page.
- **Wrap −** In case of inconsistent space for the contents of a web page (in single line), you can wrap them to multiple lines (both horizontally) and vertically.
- **Alignment −** Using Flexbox, you can align the contents of the webpage with respect to their container.
- **Resize −** Using Flexbox, you can increase or decrease the size of the items in the page to fit in available space.

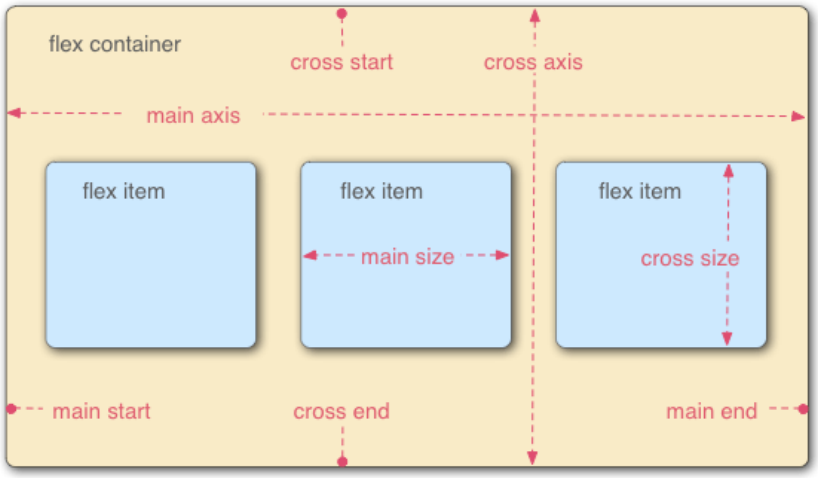
The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate all kinds of display devices and screen sizes)**. A flex container expands items to fill available free space or shrinks them to prevent overflow.**

Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).

Basics and terminology:

Since flexbox is a whole module and not a single property, it involves a lot of things including its whole set of properties. Some of them are meant to be set on the container (parent element, known as "flex container") whereas the others are meant to be set on the children (said "flex items").

When elements are laid out as flex items, they are laid out along two axes:



Items will be laid out following either the main axis (from main-start to main-end) or the cross axis (from cross-start to cross-end).

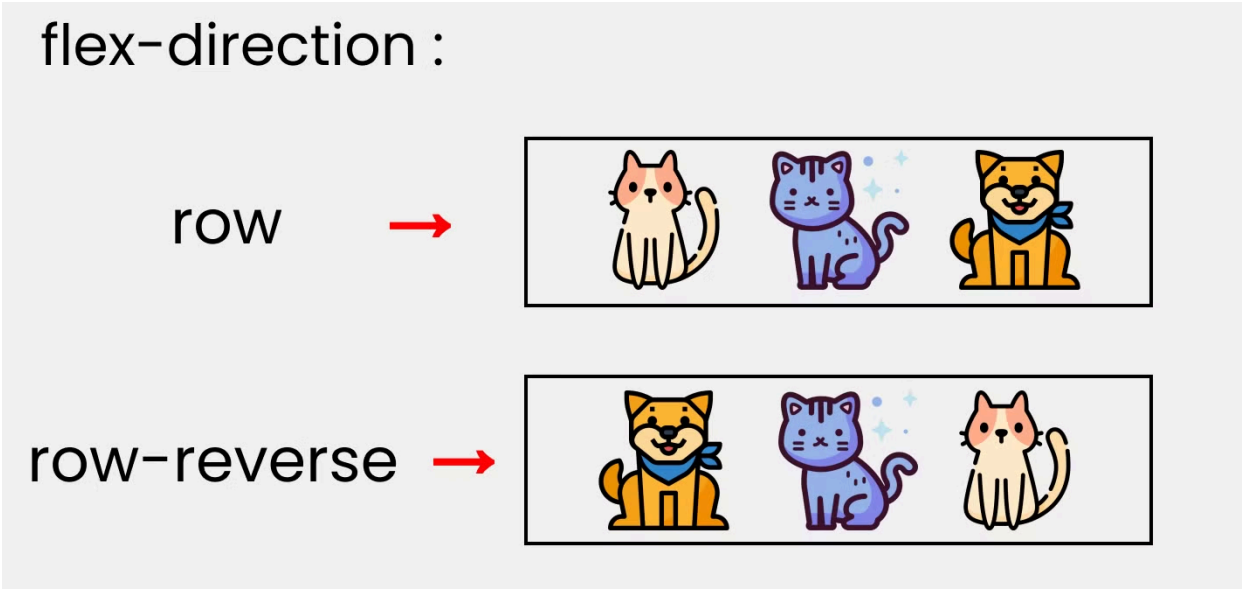
- **main axis** – The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the flex-direction property.
- **main-start | main-end** – The flex items are placed within the container starting from main-start and going to main-end.
- **main size** – A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the 'width' or 'height' property, whichever is in the main dimension.
- **cross axis** – The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.
- **cross-start | cross-end** – Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.
- **cross size** – The width or height of a flex item, whichever is in the cross dimension, is the item's cross size. The cross size property is whichever of 'width' or 'height' that is in the cross dimension.

Pet-Trainer analogy:

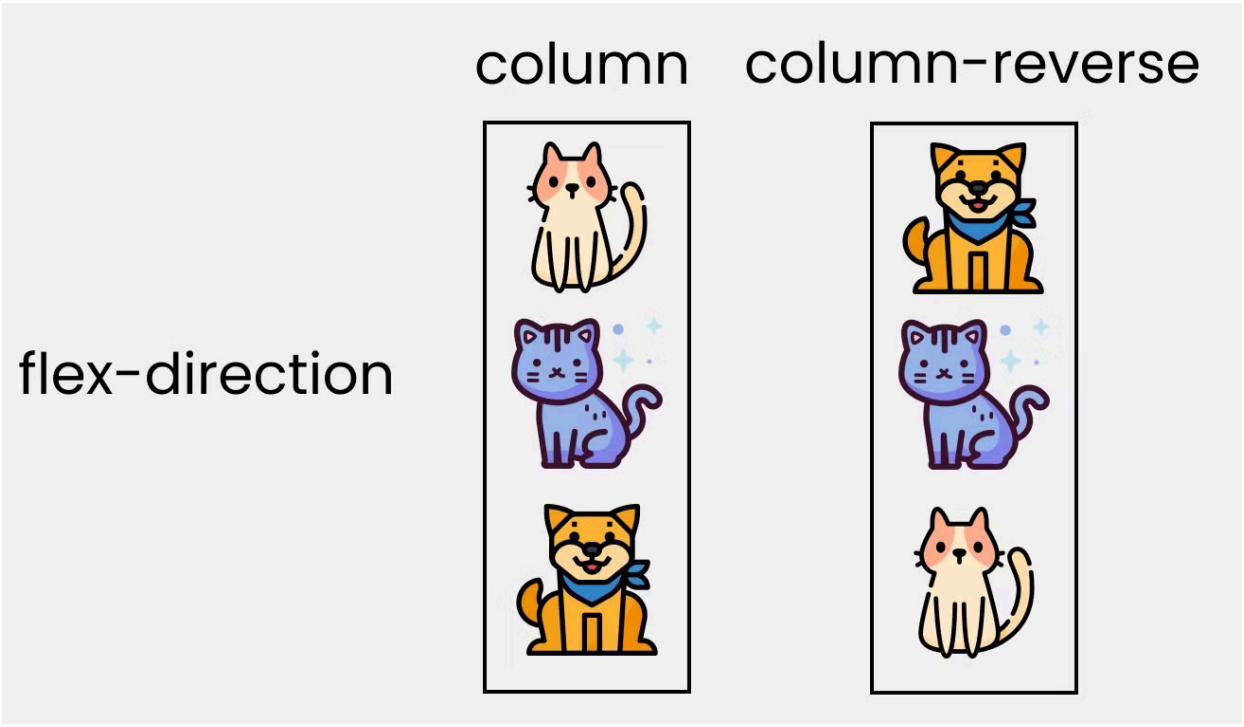
- **In flex-box, the Parent controls child's placement**

- **Pets should follow trainer/owner instructions to stand in a line**
- **For suppose if a trainer wants to make them stand across main axis**

    - he will use flex-direction:row / row-reverse

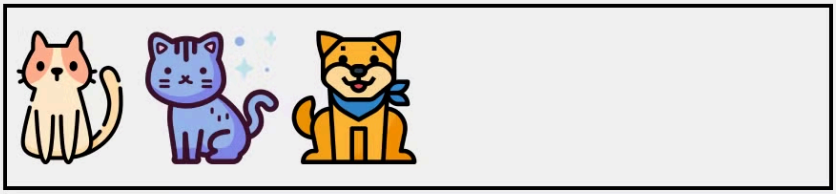


- For suppose if a trainer wants to make them stand across cross axis
    - he will use flex-direction:column

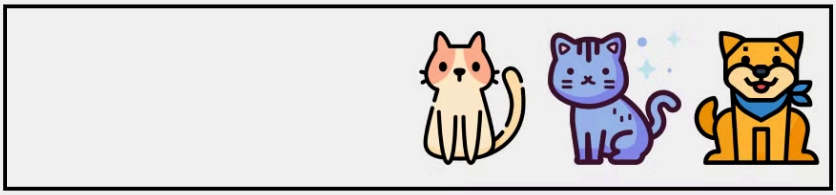


justify-content

flex-start



flex-end



center

justify-content

space-between

space-around

space-evenly

Parent Properties:

- **display:** Defines a flex container.
- **flex-direction:** Defines the main axis direction.
- **flex-wrap:** Allows items to wrap onto multiple lines.
- **flex-flow:** Shorthand for `flex-direction` and `flex-wrap` .
- **justify-content:** Aligns items along the main axis.
- **align-content:** Aligns items along the cross axis.
- **align-items:** Aligns multiple lines of items on the cross axis.

Children/Flex-items Properties:

- **order:** Changes the order of items without altering the source order.
- **flex-grow:** Allows an item to grow to fill available space.
- **flex-shrink:** Allows an item to shrink if there's insufficient space.
- **flex-basis:** Defines the initial size of an item.
- **flex:** Shorthand for `flex-grow` , `flex-shrink` , and `flex-basis` .
- **align-self:** Aligns a single item within the flex container.

Example1: Without Flexbox

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <title>Document</title>
    <style>
        .box {
            background-color: #c3a7a7;
            padding: 10px;
        }

        .box>div {
            padding: 30px;
        }

        .box>div:nth-child(1) {
            background-color: teal;
        }

        .box>div:nth-child(2) {
            background-color: orange;
        }

        .box>div:nth-child(3) {
            background-color: rgb(76, 76, 212);
        }
    </style>
</head>

<body>
    <div class="box">
        <div>1</div>
        <div>2</div>
        <div>3</div>
    </div>
</body>

</html>
```

Output:

With Flex display Property:

First, we have the `display` property. This is what defines the flex container and it's mandatory when working with Flexbox.

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

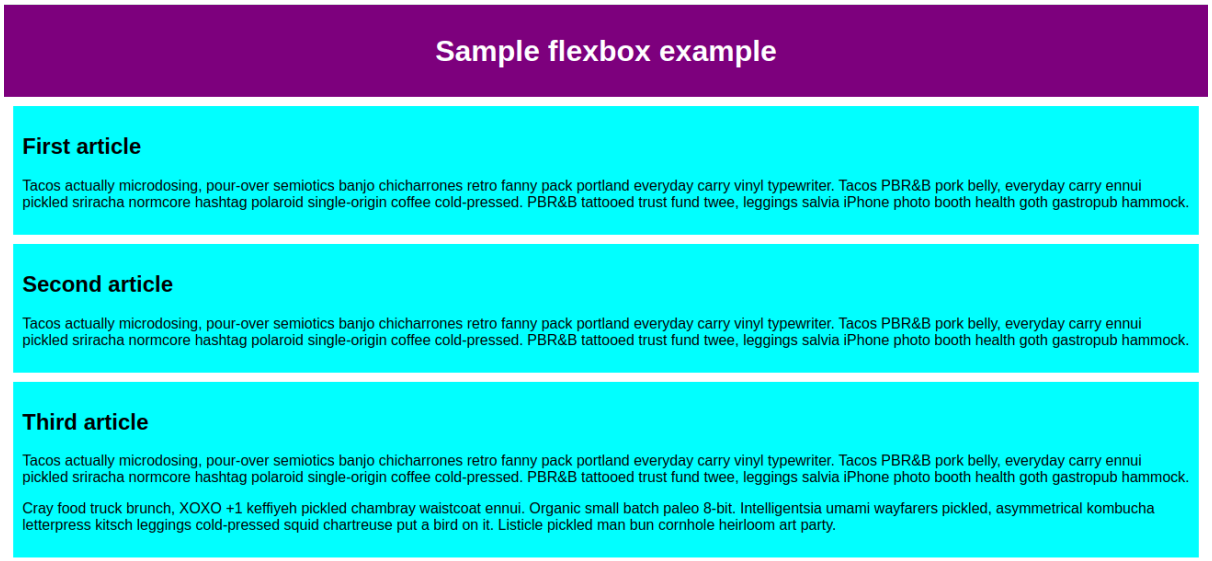
Example2: Normal Layout Without Flex

```
<!DOCTYPE html>
<html lang="en-US">

<head>
```

```
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Flexbox 0 – starting code</title>
    <style>

        body {
            margin: 0;
        }

        header {
            background: purple;
            height: 80px;
        }

        h1 {
            text-align: center;
            color: white;
            margin: 0;
        }

        article {
            padding: 10px;
            margin: 10px;
            background: aqua;
        }
    </style>
</head>

<body>
    <header>
        <h1>Sample flexbox example</h1>
    </header>

    <section>
        <article>
            <h2>First article</h2>

            <p>
            Tacos actually microdosing, pour-over semiotics banjo chicharrones
            retro fanny pack portland everyday
            carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui
            pickled sriracha normcore hashtag
            polaroid single-origin coffee cold-pressed. PBR&B tattooed trust
            fund twee, leggings salvia iPhone photo
            booth health goth gastropub hammock.
            </p>

         </article>
```

```
        <article>
            <h2>Second article</h2>


            <p>
            Tacos actually microdosing, pour-over semiotics banjo chicharrones
            retro fanny pack portland everyday
            carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui
            pickled sriracha normcore hashtag
            polaroid single-origin coffee cold-pressed. PBR&B tattooed trust
            fund twee, leggings salvia iPhone photo
            booth health goth gastropub hammock.
            </p>
        </article>

        <article>
            <h2>Third article</h2>

            <p>
            Tacos actually microdosing, pour-over semiotics banjo chicharrones
            retro fanny pack portland everyday
            carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui
            pickled sriracha normcore hashtag
            polaroid single-origin coffee cold-pressed. PBR&B tattooed trust
            fund twee, leggings salvia iPhone photo
            booth health goth gastropub hammock.
      </p>

            <p>
            Cray food truck brunch, XOXO +1 keffiyeh pickled chambray waistcoat
            ennui. Organic small batch paleo
            8-bit. Intelligentsia umami wayfarers pickled, asymmetrical kombucha
            letterpress kitsch leggings
            cold-pressed squid chartreuse put a bird on it. Listicle pickled man
            bun cornhole heirloom art party.
            </p>

        </article>
    </section>
</body>

</html>
```

Output:

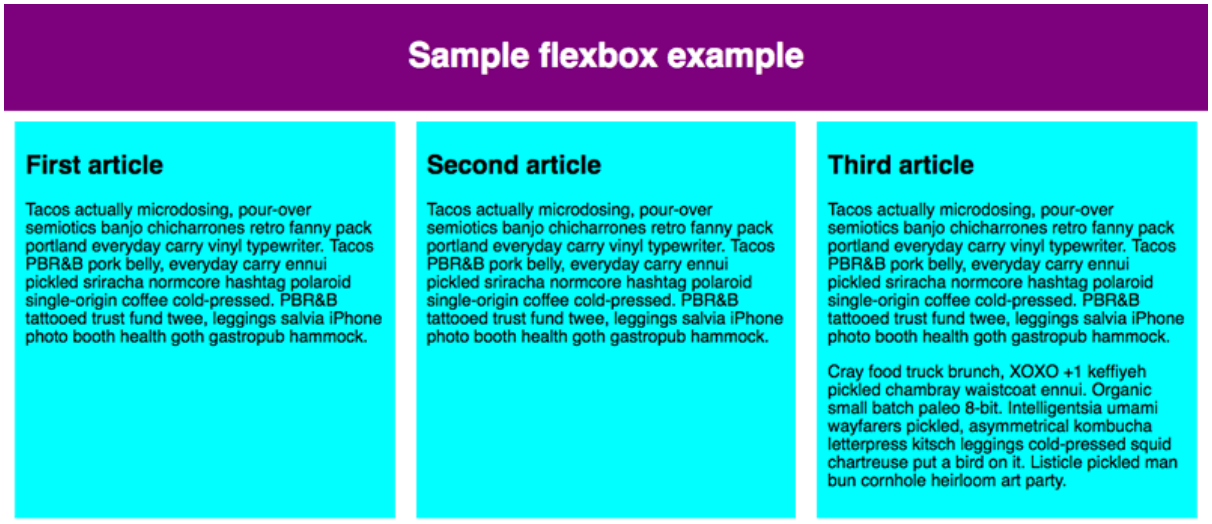


Specifying what elements to lay out as flexible boxes:

To start with, we need to select which elements are to be laid out as flexible boxes. To do this, we set a special value of display on the parent element of the elements you want to affect. In this case we want to lay out the <article> elements, so we set this on the <section>:

```
section {
            display: flex;
        }
```

This causes the <section> element to become a **flex container** and its children become **flex items**. This is what it looks like:



Flexbox Properties:

Properties for the parent(flex container)

1. **display**

display: flex | inline-flex

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

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

- **flex:** It generates a block level flex container. (It occupies the full width of the parent container).
- **Inline-flex**: It generates a inline flex container. (It just takes the place required for the content).

Example: flex

```
<!DOCTYPE html>

<html lang="en">
```

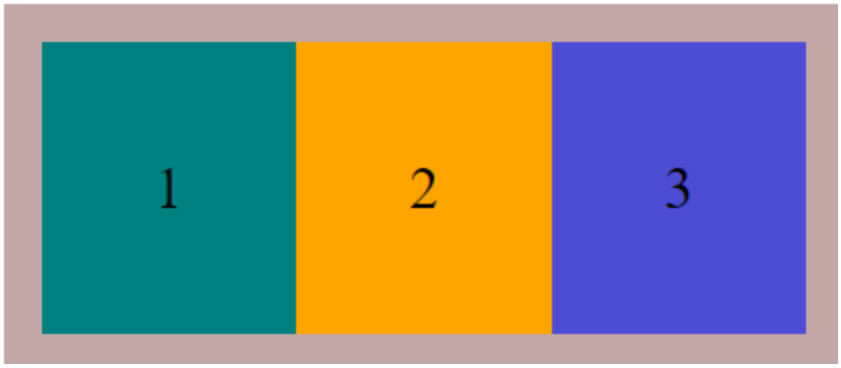
```
<head>
    <title>Document</title>
    <style>
        .box {
            background-color: #c3a7a7;
            padding: 10px;
            display: flex;
        }

        .box>div {
            padding: 30px;
        }

        .box>div:nth-child(1) {
            background-color: teal;
        }

        .box>div:nth-child(2) {
            background-color: orange;
        }

        .box>div:nth-child(3) {
            background-color: rgb(76, 76, 212);
        }
    </style>
```

```
</head>


<body>
    <div class="box">
        <div>1</div>
        <div>2</div>
        <div>3</div>
    </div>
</body>


</html>
```

Output:

Example: inline-flex

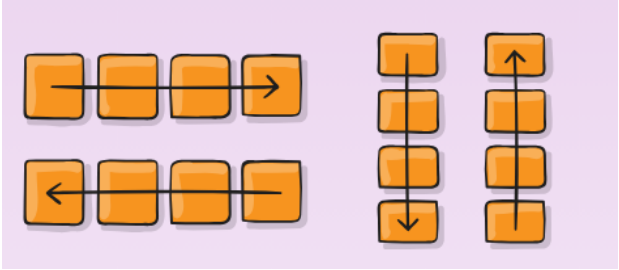
Just change the **display** value as **inline-flex** in the above example and the output will be like as follows:



2. flex-direction

The **flex-direction** property is used to specify the direction in which the elements of flex container (flex-items) are needed to be placed inside the container.

- `flex-direction` can accept one of four values:
  - `row`
  - `row-reverse`
  - `column`
  - `column-reverse`



```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
```

```
}
```

- **row** – It is the default value, Arranges the elements of the container horizontally from left to right.
- **row-reverse** – Arranges the elements of the container horizontally from right to left.
- **column** – Arranges the elements of the container vertically from left to right.
- **column-reverse** – Arranges the elements of the container vertically from right to left.

flex-direction: row

The first value is a row that is the default value of the flex-direction. It doesn't make any changes by default. It's placed on the main axis from left to the right.

Example:

```
.box {
            background-color: #c3a7a7;
            padding: 10px;
            display: flex;
            flex-direction: row;
        }
```

flex-direction: row-reverse

- This sets the main access direction from right to left, which results in the flex items being placed from right to left. In the example below, you can see that the items are now placed in the reverse order. Item 1 is placed to the right:

```
.box {
```

```
        background-color: #c3a7a7;
        padding: 10px;
        display: flex;
        flex-direction: row-reverse;
    }
```

Output:

flex-direction: `column`

When you set `flex-direction` to `column`, the main axis goes from top to bottom, so the items are now stacked on top of each other:

```
.box {
        background-color: #c3a7a7;
        padding: 10px;
        display: flex;
        flex-direction: column;
    }
```

flex-direction: `column-reverse`

We also have `column-reverse`, which stacks the items in the reverse order. Look at the example below. You can see Item 1 is at the bottom and Item 3 is at the top:

```
.box {
    background-color: #c3a7a7;
    padding: 10px;
    display: flex;
    flex-direction: column-reverse;
}
```

Output:

flex-wrap:

It is used to control the wrapping of items within a container. If we reduce the browser width, we lose some items for the browser width. The behaviour changes with the `flex-wrap` property. It can accept three possible values:

- **`nowrap` (default value)**
- **`wrap`**
- **`wrap-reverse`**

flex-wrap: `nowrap`

- This is the `flex-wrap` property default value. If you set the property value to `nowrap`, there are no changes.

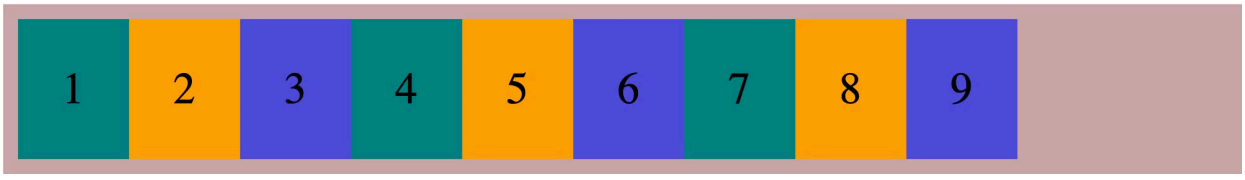
```
<!DOCTYPE html>

<html lang="en">


<head>
```

```
<title>Document</title>
<style>
    .box {
        display: flex;
        background-color: #c3a7a7;
        padding: 10px;
        flex-wrap: nowrap
    }

    .box>div {
        padding: 30px;
        font-size: 30px;
    }

    .box>div:nth-child(1) {
        background-color: teal;
    }

    .box>div:nth-child(2) {
        background-color: orange;
    }

    .box>div:nth-child(3) {
        background-color: rgb(76, 76, 212);
    }
```

```
        .box>div:nth-child(4) {
            background-color: teal;
        }

        .box>div:nth-child(5) {
            background-color: orange;
        }

        .box>div:nth-child(6) {
            background-color: rgb(76, 76, 212);
        }

        .box>div:nth-child(7) {
            background-color: teal;
        }

        .box>div:nth-child(8) {
            background-color: orange;
        }

        .box>div:nth-child(9) {
            background-color: rgb(76, 76, 212);
        }
    </style>
</head>
```

```
<body>
    <div class="box">
        <div>1</div>
        <div>2</div>
        <div>3</div>
        <div>4</div>
        <div>5</div>
        <div>6</div>
        <div>7</div>
        <div>8</div>
        <div>9</div>
    </div>
</body>

</html>
```

Output:



Here if we decrease the window size, we can see the container size will be decreased but all the items will be in the same line.

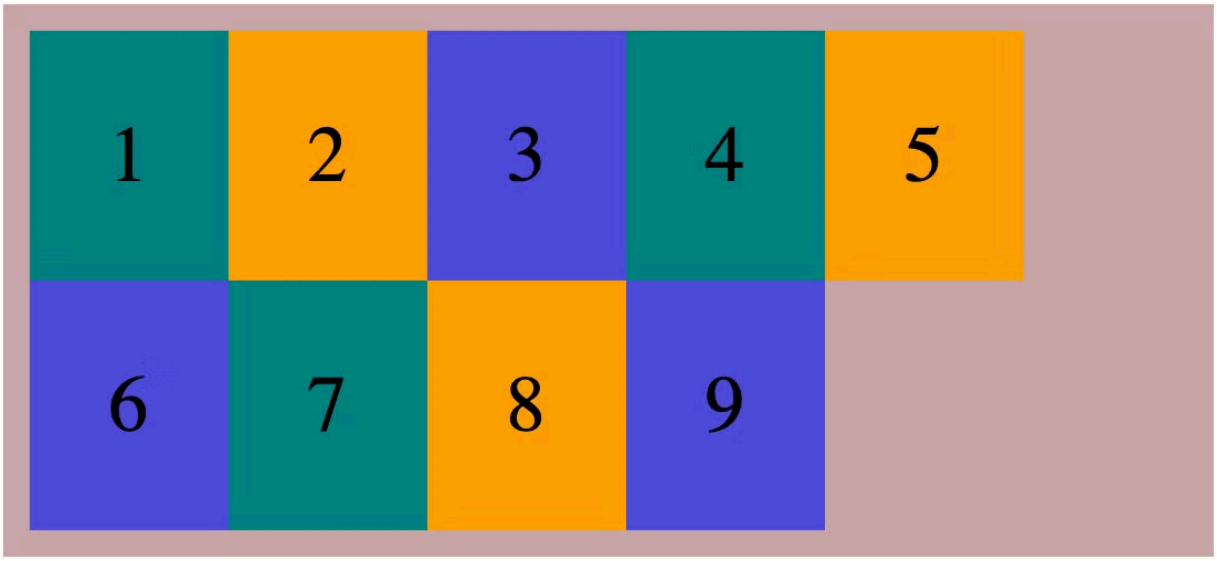
flex-wrap: wrap

- **When you set the `flex-wrap` property to `wrap`, you reduce the browser width that the items have wrapped in the container:**

```
.box {
        display: flex;
        background-color: #c3a7a7;
        padding: 10px;
        flex-wrap: wrap
    }
```

Here we can see the effect by decreasing the window size.

Output:

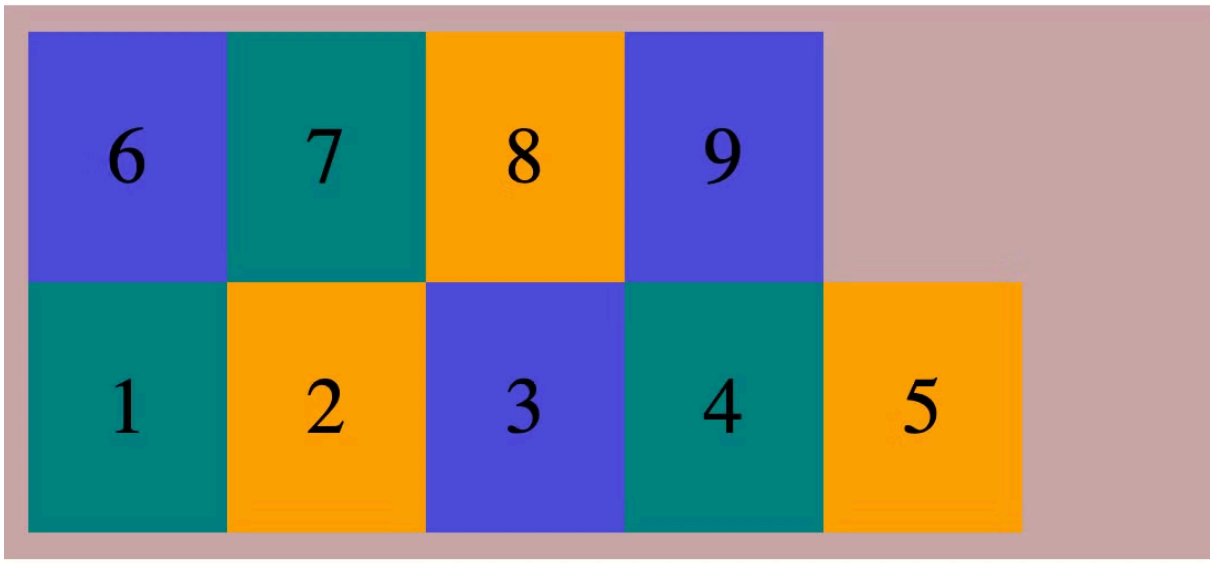


flex-wrap: `wrap-reverse`

- **Instead of items falling into the row below, they climb into the row above. Wrapping always occurs from the last item. `wrap-reverse` pushes the last item above instead of below:**

```
.box {
        display: flex;
        background-color: #c3a7a7;
        padding: 10px;
        flex-wrap: wrap-reverse;
    }
```

Output:



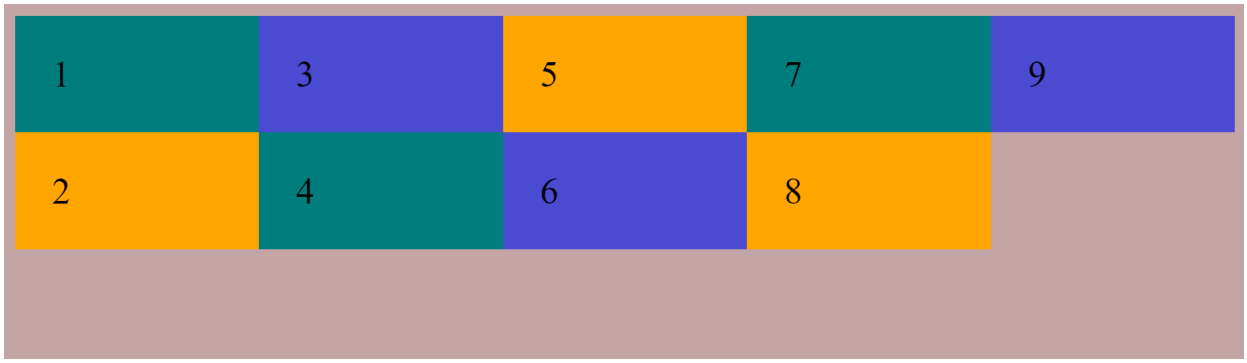
Wrapping the column:

- To wrap the items column wise, we need to set the **height** of the container and the direction should be **column**.

```
.box {
    display: flex;
```

```
        background-color: #c3a7a7;
        padding: 10px;
        flex-direction: column;
        height: 50vh;
        flex-wrap: wrap;
    }
```

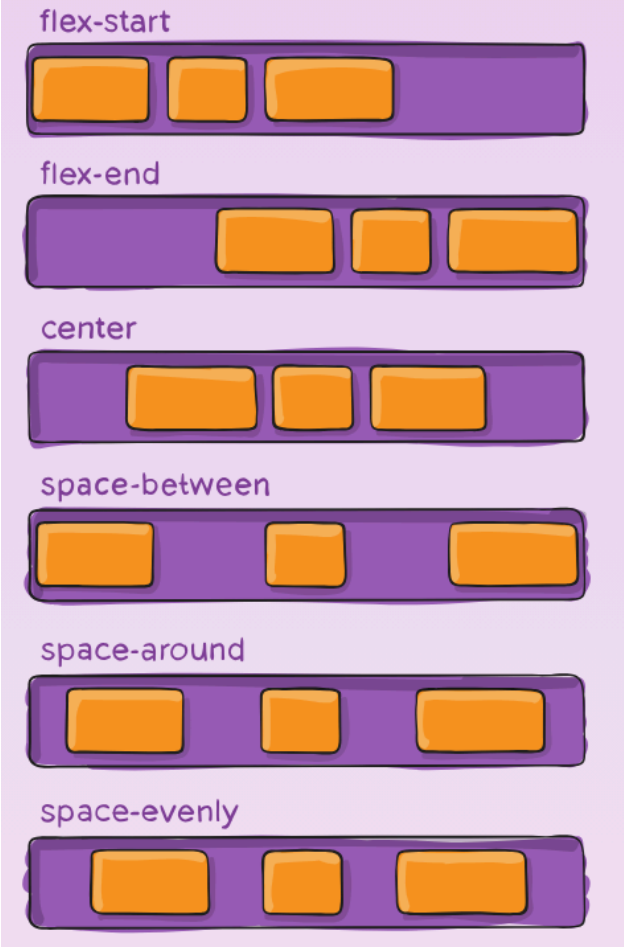
Output:



justify-content:

- `justify-content` defines the alignment of the items along the main axis. It helps distribute extra free space leftover. There are six possible values for the `justify-content` property:

    - **`flex-start`**
    - **`flex-end`**
    - **`center`**
    - **`space-between`**
    - **`space-around`**
    - **`space-evenly`**



justify-content: `flex-start`

- (default value) Items are placed towards the start of the flex-direction(main axis).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <style>
        .box {
            display: flex;
```

```
            background-color: #c3a7a7;
            padding: 10px;
            justify-content: flex-start;
        }
        .box>div {
            padding: 30px;
            font-size: 30px;
        }
        .box>div:nth-child(1) {
            background-color: teal;
        }
        .box>div:nth-child(2) {
            background-color: orange;
        }
        .box>div:nth-child(3) {
            background-color: rgb(76, 76, 212);
        }
    </style>
</head>
<body>
    <div class="box">
        <div>1</div>
        <div>2</div>
        <div>3</div>
    </div>
</body>
```

```
</html>
```

Output:

justify-content: flex-end

- This aligns the items to be placed at the end of the main axis:

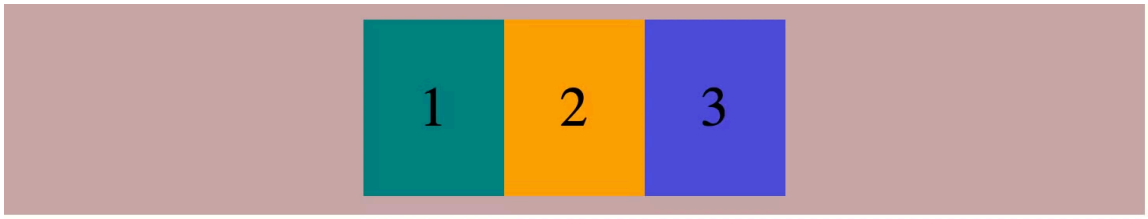
```
.box {
        display: flex;
        background-color: #c3a7a7;
        padding: 10px;
        justify-content: flex-end;
    }
```

Output:

justify-content: center

```
.box {
        display: flex;
        background-color: #c3a7a7;
        padding: 10px;
        justify-content: center;
    }
```

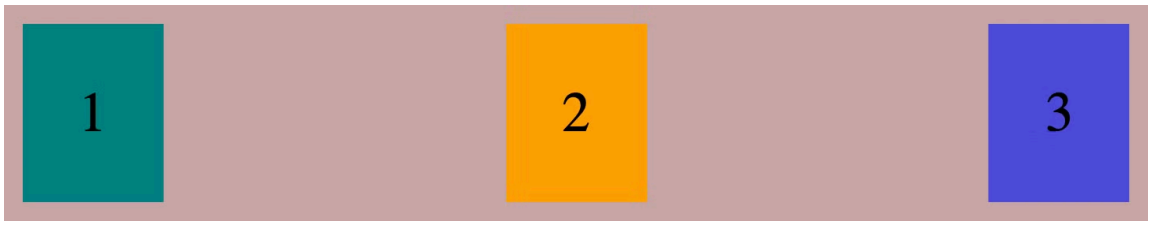
Output:



justify-content: space-between

- This value helps to evenly split extra space and add it in between the flex items

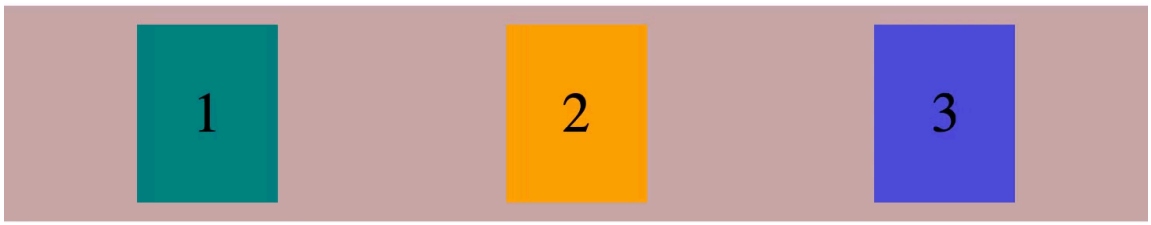
Output:



justify-content: space-around

- This value splits the extra space at the beginning and the end. The space in question is equal to half of the space between the flex items:

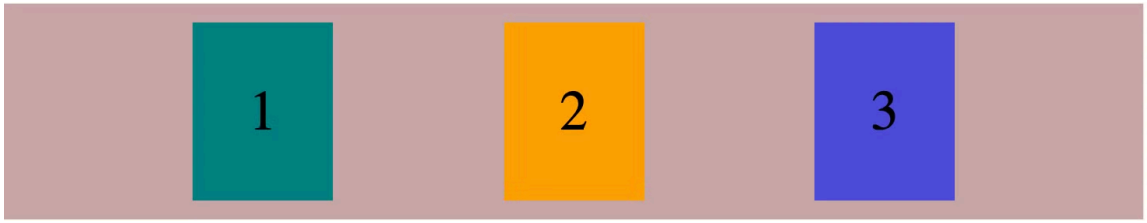
Output:



justify-content: space-evenly

- This value distributes the extra space in the container:

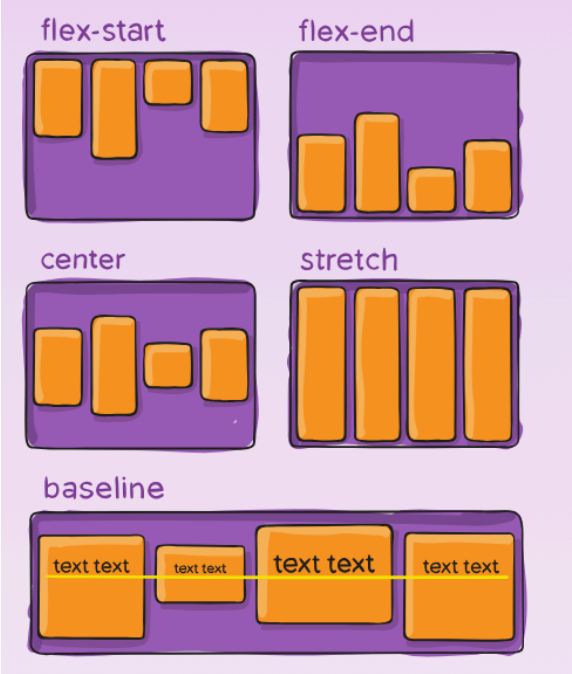
Output:



align-items:

This defines the default behavior for how flex items are laid out along the **cross axis** on the current line. Think of it as the justify-content version for the cross-axis (perpendicular to the main-axis).

- align-items: stretch
- align-items: flex-start
- align-items: flex-end
- align-items: center
- align-items: baseline



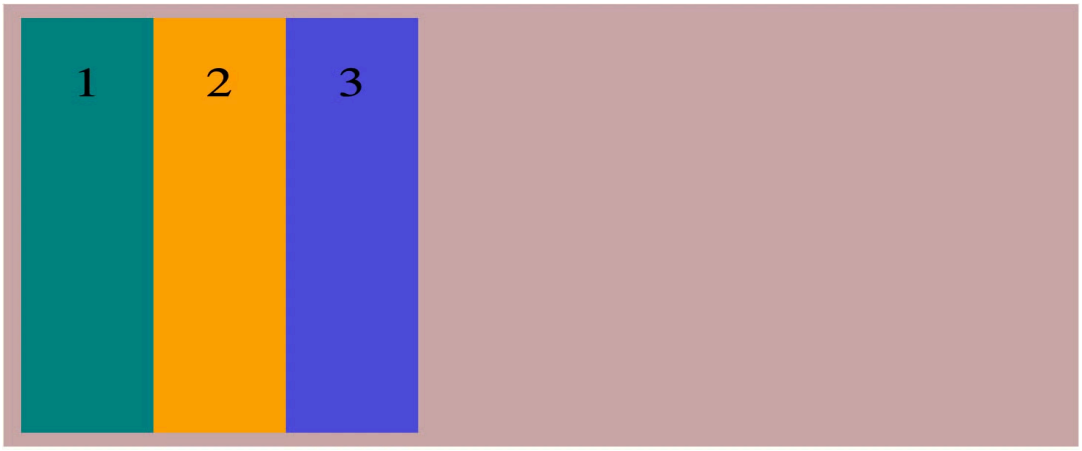
align-items: `stretch`

- The items stretch the entire length of the cross axis and `stretch` is the `default` value:

Example:

```
.box {
        display: flex;
        background-color: #c3a7a7;
        padding: 10px;
        height: 300px;
        align-items: stretch;
    }
```

Output:



align-items: `flex-start`

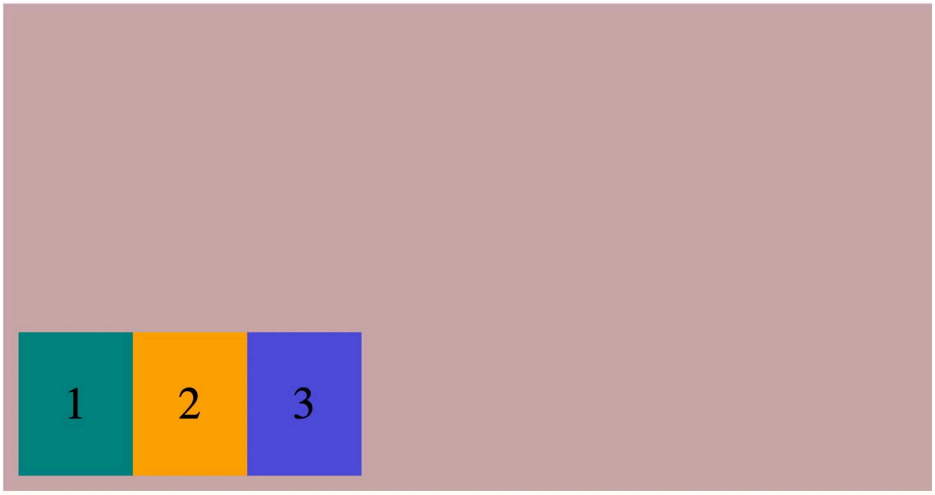
- This is the starting point of the cross axis. The cross axis flows from top to bottom. The item doesn’t stretch and is aligned with the cross start of the line:

Output:

align-items: `flex-end`

- This value pushes the items to the end of the cross axis:

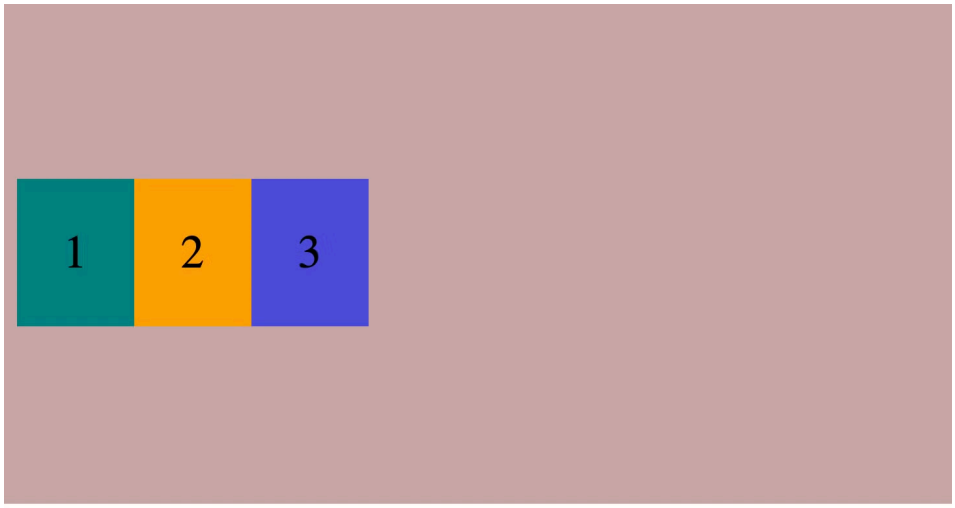
Output:



align-items: `center`

- This centers the content along the cross axis:

Output:



align-items: `baseline`

- aligns the text of all the divs based on their baseline (the invisible line where the letters sit).

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Baseline Example</title>
    <style>
        .box {
            display: flex;
            background-color: #c3a7a7;
            padding: 10px;
            height: 300px;

            /* Aligns flex items by their text baselines */
            align-items: baseline;
        }

        .box > div {
            padding: 30px;
            font-size: 20px;
            border: 1px solid black;
        }

        .box > div:nth-child(1) {
            background-color: teal;
            font-size: 40px; /* Larger font to show baseline alignment */
        }

        .box > div:nth-child(2) {
            background-color: orange;
            font-size: 30px; /* Medium font */
        }

        .box > div:nth-child(3) {
            background-color: rgb(76, 76, 212);
```

```
            font-size: 20px; /* Small font */
        }
    </style>
</head>

<body>
    <div class="box">
        <div>Large Text</div>
        <div>Medium Text</div>
        <div>Small Text</div>
    </div>
</body>

</html>
```

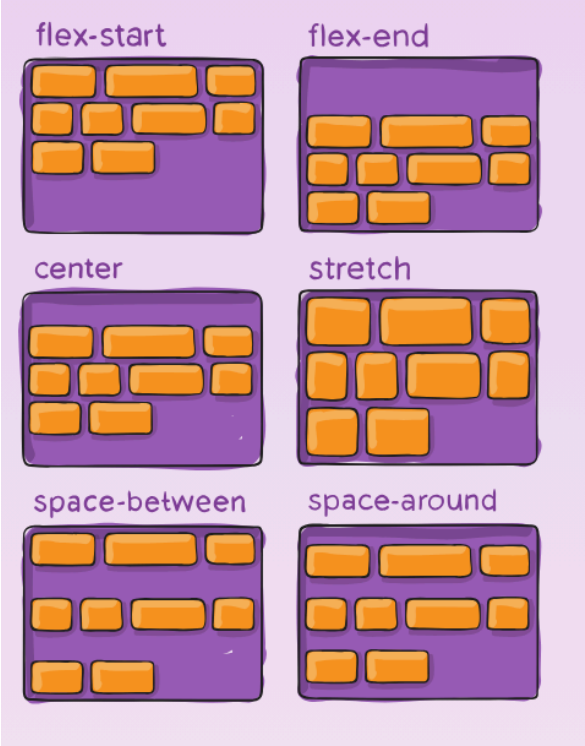
align-content:

In case the flex-container has multiple lines (when, flex-wrap: wrap), the align-content property defines the alignment of each line within the container.

Note: This property only takes effect on multi-line flexible containers, where flex-wrap is set to either wrap or wrap-reverse). A single-line flexible container (i.e. where flex-wrap is set to its default value, no-wrap) will not reflect align-content.

The container must have a height and also wrapping.

- `align-content: stretch`
- `align-content: flex-start`
- `align-content: flex-end`
- `align-content: center`
- `align-content: space-between`
- `align-content: space-around`



align-content: `stretch`

- This is the `align-content` default value. It has no effect on alignment.

Example:

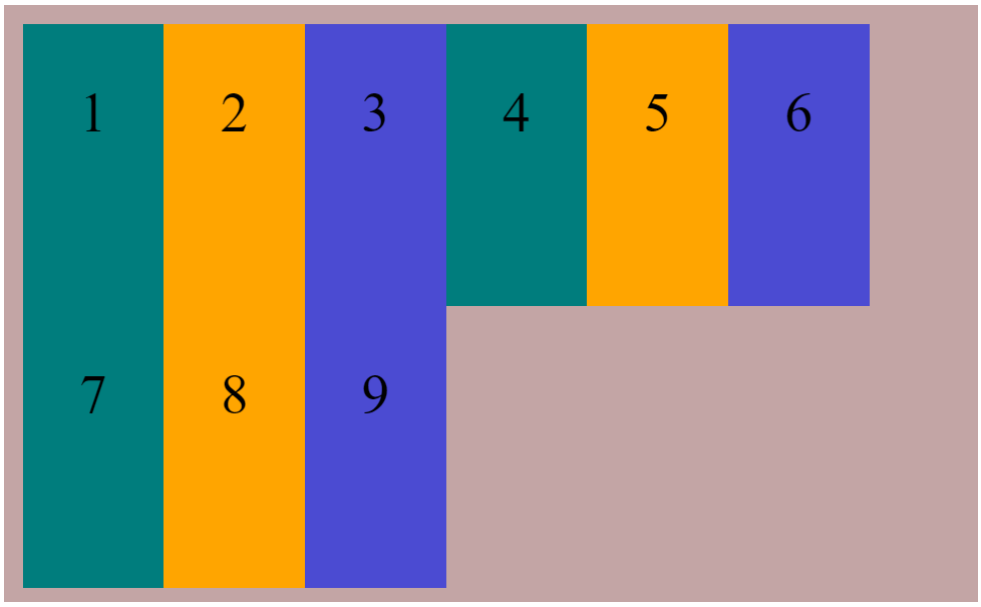
```
<!DOCTYPE html>

<html lang="en">


<head>

    <title>Document</title>

    <style>

        .box {

            display: flex;

            background-color: #c3a7a7;
```

```
    padding: 10px;
    flex-wrap: wrap;
    height: 300px;
    /* Without width we need to resize the window to see the effect */
    width: 500px;
    align-content: stretch;
}
.box>div {
    padding: 30px;
    font-size: 30px;
}
.box1 {
    background-color: teal;
}
.box2 {
    background-color: orange;
}
.box3 {
    background-color: rgb(76, 76, 212);
}
.box4 {
    background-color: teal;
}
.box5 {
    background-color: orange;
}
```

```
        .box6 {
            background-color: rgb(76, 76, 212);
        }
        .box7 {
            background-color: teal;
        }
        .box8 {
            background-color: orange;
        }
        .box9 {
            background-color: rgb(76, 76, 212);
        }
    </style>
</head>
<body>
    <h3>Change values of align-content to see different outputs</h3>
    <div class="box">
        <div class="box1">1</div>
        <div class="box2">2</div>
        <div class="box3">3</div>
        <div class="box4">4</div>
        <div class="box5">5</div>
        <div class="box6">6</div>
        <div class="box7">7</div>
        <div class="box8">8</div>
        <div class="box9">9</div>
```

```
    </div>
</body>
</html>
```

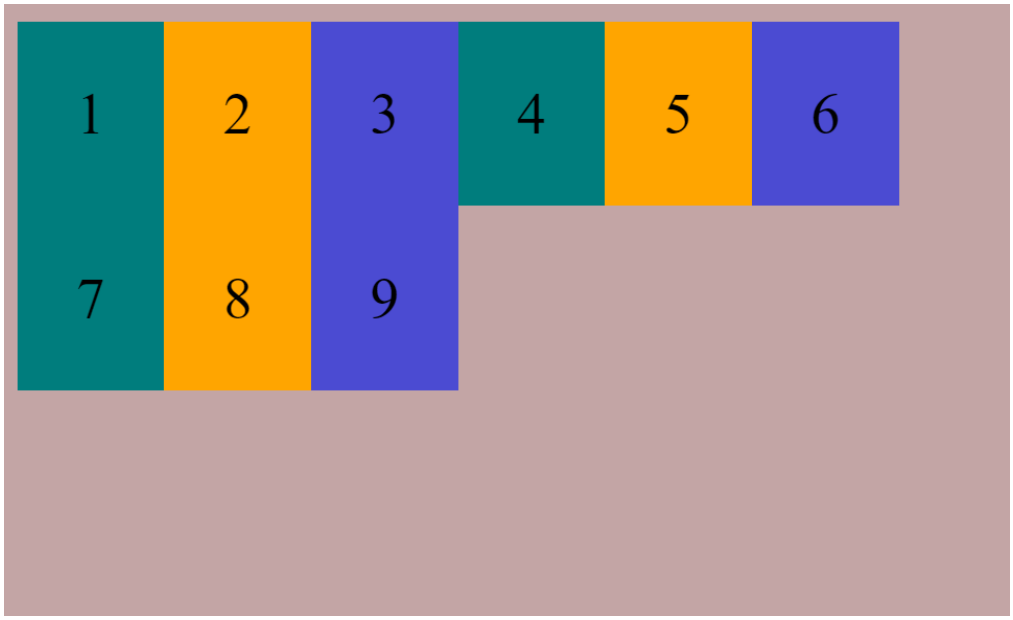
Output:



align-content: `flex-start`

- **This value pulls the lines to the beginning of the cross axis:**

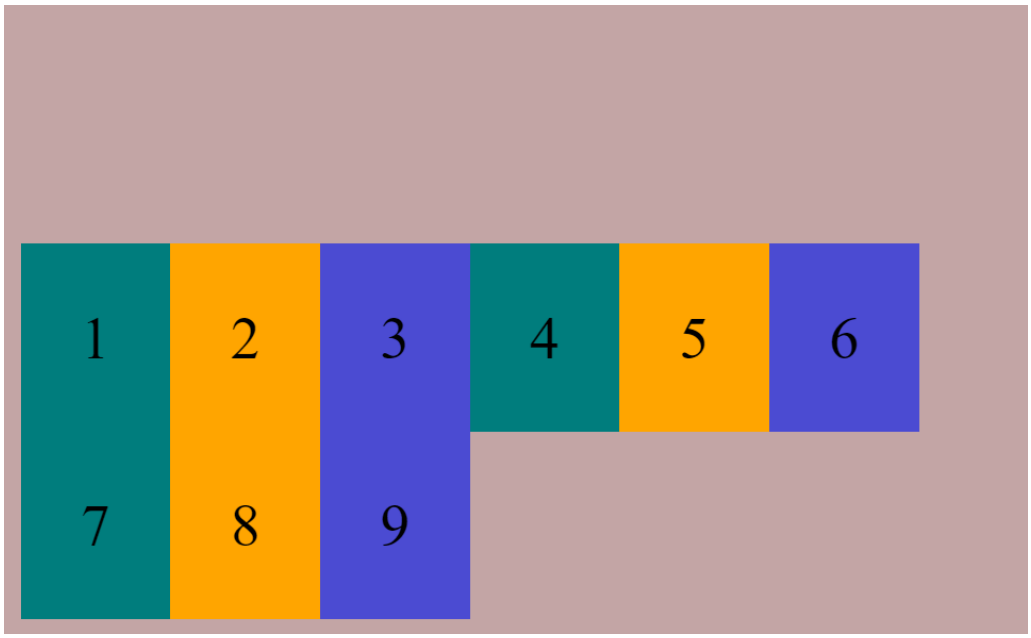
Output:



align-content: `flex-end`

- This value pushes the lines to the end of the cross axis:

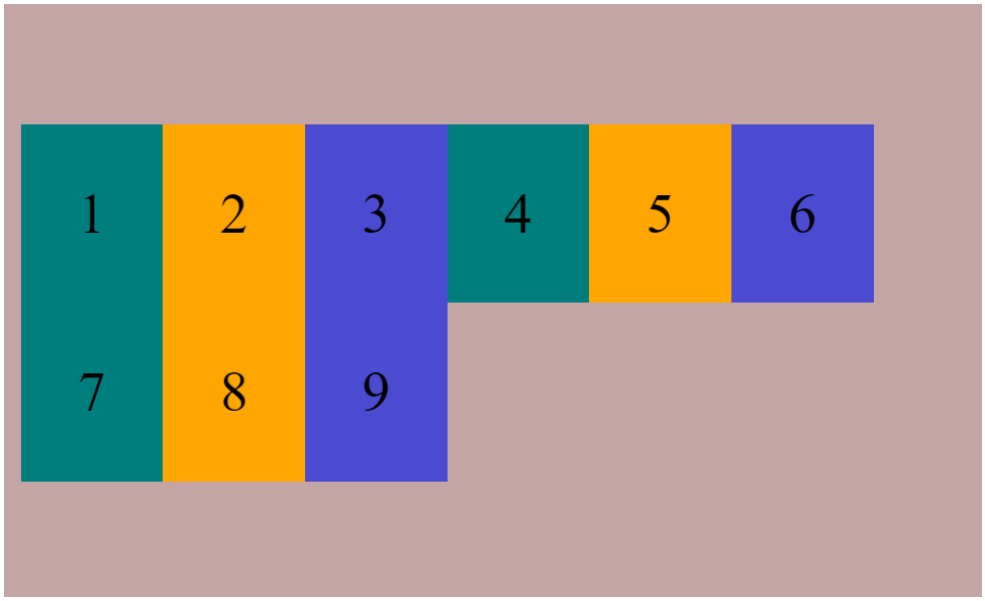
Output:



align-content: `center`

- This value pushes the lines to the center of the cross axis:

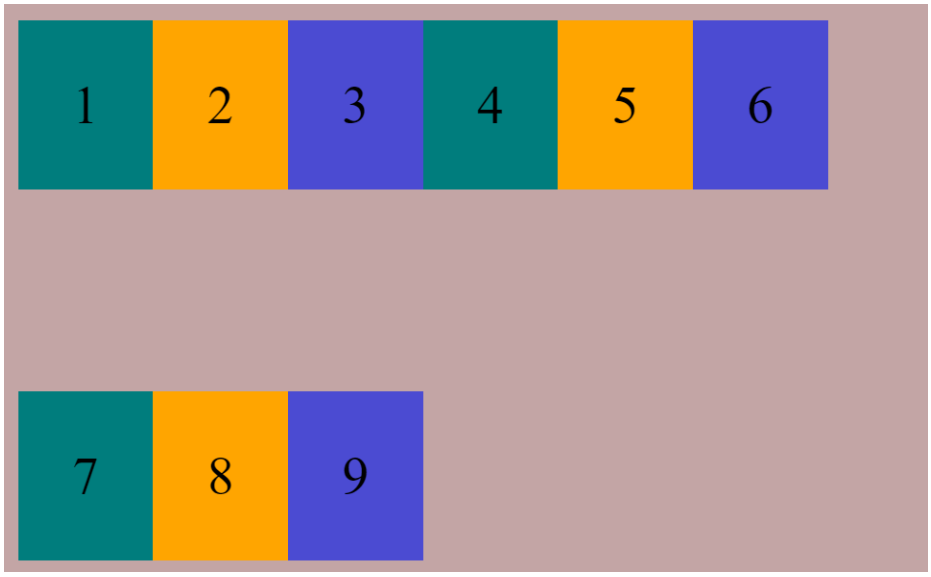
Output:



align-content: `space-between`

- This value takes all the extra space and puts it in between the lines:

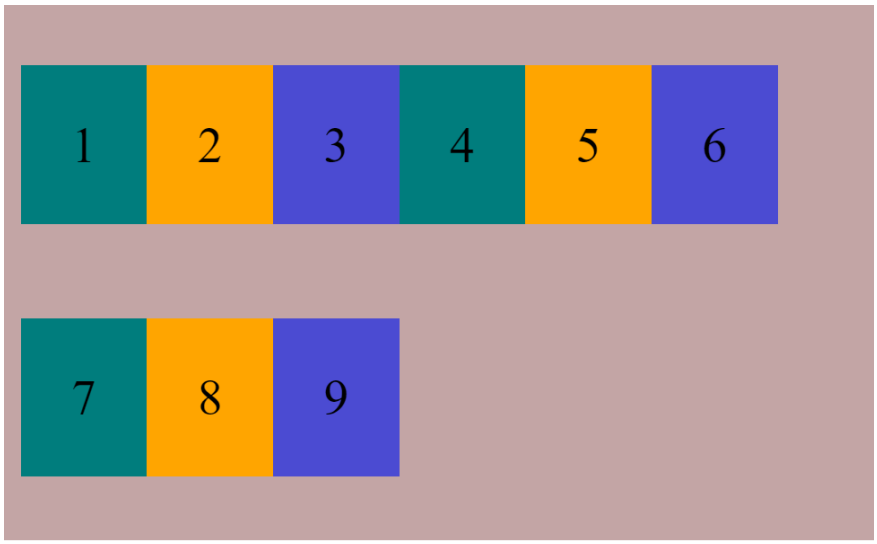
Output:



align-content: `space-around`

- This distributes the space around the lines:

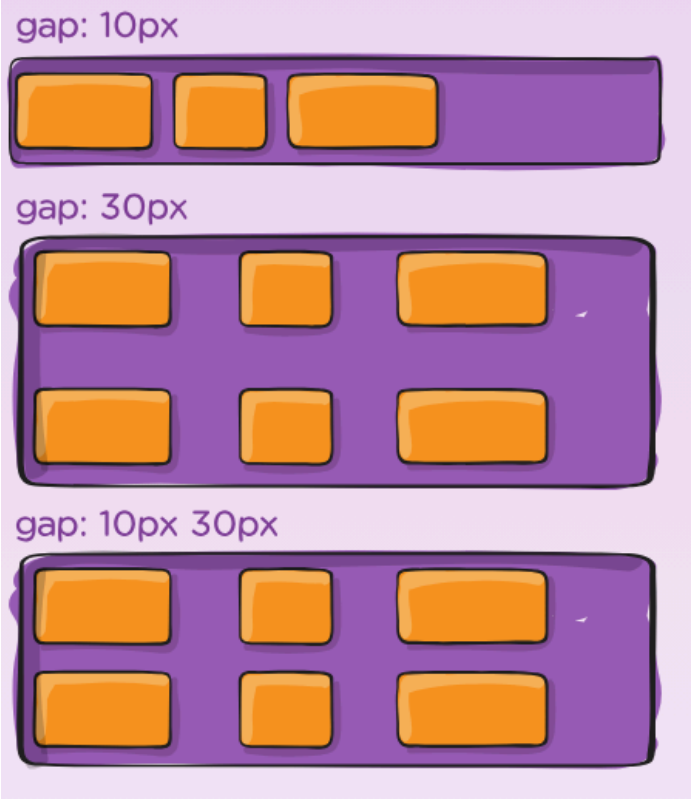
Output:



gap, row-gap, column-gap:

The **gap** property explicitly controls the space between flex items. It applies that spacing only between items not on the outer edges.

Note: gap is not exclusively for flexbox, gap works in grid and multi-column layout as well.



Syntax:

```
.container {
  display: flex;
  gap: 10px; /* gap between row and columns */
  gap: 10px 20px; /* row-gap column gap */
  row-gap: 10px;
  column-gap: 20px;
}
```

Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
    }

    .box2 {
        background: blue;
    }

    .box3 {
        background: red;
    }

    .box4 {
        background: magenta;
    }

    .box5 {
        background: yellow;
    }

    .box6 {
        background: pink;
    }

    .box {
```

```
        font-size: 25px;
        padding: 15px;


    }


    .container {
        box-sizing: border-box;
        border: 2px solid black;
        display: flex;
        width: 200px;
        flex-wrap: wrap;
        /* gap: 20px; */
        /* row-gap: 20px; */
        /* gap: 10px 20px; */

    }
</style>


<body>
    <div class="container">
        <div class="box box1">1</div>
        <div class="box box2">2</div>
        <div class="box box3">3</div>
        <div class="box box4">4</div>
        <div class="box box5">5</div>
        <div class="box box6">6</div>
    </div>
</body>


</html>
```

Putting a div to the center of a page using flex by using parent properties.

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body {
            display: flex;

            justify-content: center;
            align-items: center;
            height: 100vh;
            /* Full viewport height */

        }

        .center-box {
            width: 300px;
            height: 300px;
            background-color: lightblue;
        }
    </style>
    <title>Center Div with Flexbox</title>
</head>

<body>
    <div class="center-box"></div>
</body>

</html>
```

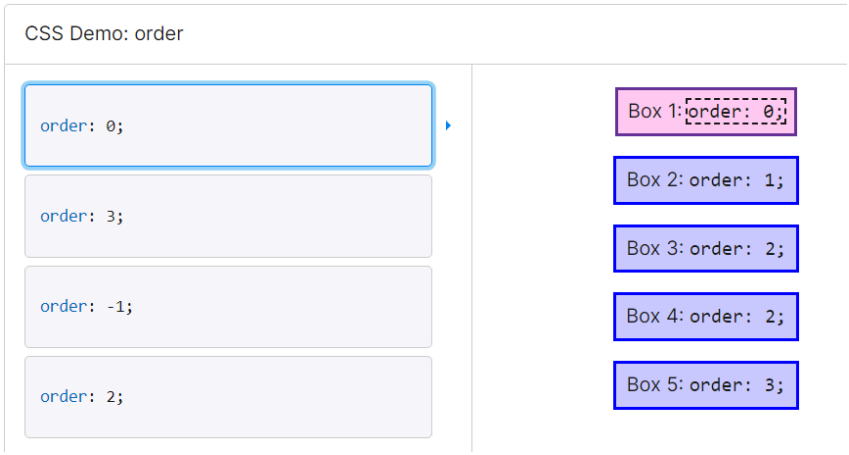
Properties for the Children (flex items):

- **order:** Changes the order of items without altering the source order.
- **flex-grow:** Allows an item to grow to fill available space.
- **flex-shrink:** Allows an item to shrink if there's insufficient space.
- **flex-basis:** Defines the initial size of an item.
- **flex:** Shorthand for `flex-grow` , `flex-shrink` , and `flex-basis` .
- **align-self:** Aligns a single item within the flex container.

1. order:

- The `order` property specifies the order of a flexible item relative to the rest of the flexible items inside the same container.
- The items are placed in the visual order according to that integer, lowest values first. If more than one item has the same integer value, then within that group the items are laid out as per source order.
- Flex items have a default `order` value of `0`.

Note: If the element is not a flexible item, the `order` property has no effect.



Refer: flexbox order mdn

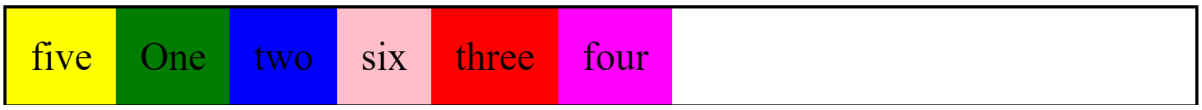
https://developer.mozilla.org/en-US/docs/Web/CSS/order

Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
    }

    .box2 {
        background: blue;
    }

    .box3 {
        background: red;
        order: 1
    }

    .box4 {
        background: magenta;
        order: 2;
    }

    .box5 {
        background: yellow;
        order: -1;
```

```
    }

    .box6 {
        background: pink;
    }

    .box {
        font-size: 25px;
        padding: 15px;

    }

    .container {
        border: 2px solid black;
        display: flex;

    }
</style>

<body>
    <div class="container">
        <div class="box box1">One</div>
        <div class="box box2">two</div>
        <div class="box box3">three</div>
        <div class="box box4">four</div>
        <div class="box box5">five</div>
        <div class="box box6">six</div>
    </div>
</body>

</html>
```

Output:



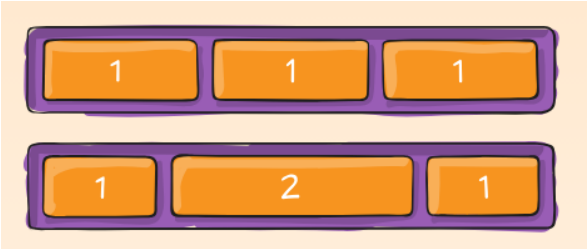
2. flex-grow:

This property specifies how much of the remaining space in the flex container should be assigned to the item (the flex grow factor).

The remaining space is the size of the flex container minus the size of all flex items' sizes together. If all sibling items have the same flex grow factor, then all items will receive the same share of remaining space, otherwise, it is distributed according to the ratio defined by the different flex-grow factors.

If all items have flex-grow set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, that child would take up twice as much of the space either one of the others.

The default flex-grow value for each item is 0.

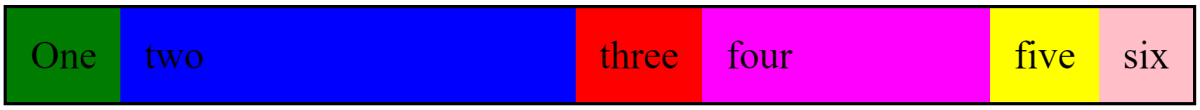


Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
    }

    .box2 {
        background: blue;
        flex-grow: 2;
    }

    .box3 {
        background: red;
    }

    .box4 {
        background: magenta;
        flex-grow: 1;
    }

    .box5 {
        background: yellow;
    }
```

```
    .box6 {
        background: pink;
    }

    .box {
        font-size: 25px;
        padding: 15px;

    }

    .container {
        border: 2px solid black;
        display: flex;

    }
</style>

<body>
    <div class="container">
        <div class="box box1">One</div>
        <div class="box box2">two</div>
        <div class="box box3">three</div>
        <div class="box box4">four</div>
        <div class="box box5">five</div>
        <div class="box box6">six</div>
    </div>
</body>

</html>
```

Output:



3. flex-shrink:

The **flex-shrink** property specifies how the item will shrink relative to the rest of the flexible items inside the same container.

Flex-shrink: 1 (default value) all elements will shrink parallelly (responsive)

Flex-shrink: 0 element size will be fixed (responsive)

Example: Let the second flex-item shrink three times more than the rest:

```
div:nth-of-type(2) {
      flex-shrink: 3;
}
```

Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
        flex-shrink: 0;
    }

    .box2 {
        background: blue;
        flex-shrink: 2;
    }

    .box3 {
        background: red;
    }

    .box4 {
        background: magenta;

    }

    .box5 {
        background: yellow;
    }

    .box6 {
        background: pink;
    }
```

```
    .box {
        width: 100px;
        font-size: 25px;
        padding: 15px;

    }

    .container {
        border: 2px solid black;
        display: flex;

    }
</style>

<body>
    <div class="container">
        <div class="box box1">One</div>
        <div class="box box2">two</div>
        <div class="box box3">three</div>
        <div class="box box4">four</div>
        <div class="box box5">five</div>
        <div class="box box6">six</div>
    </div>
</body>

</html>
```

Output:

4. flex-basis:

The flex-basis property specifies the initial length of a flexible item. This defines the default size of an element before the remaining space is distributed.

It is similar to **max-width** property.

Let's say for box2 the flex-basis is 150px then this box will takes the initial size as 150px and if we apply flex-grow1 then the remaining space will be taken by this box with the ratio 1, if we apply the flex-grow 1 to the 5th box then the remaining space will be taken by the box5.

Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
    }

    .box2 {
        background: blue;
        flex-basis: 150px;
        /* flex-grow: 1; */

    }

    .box3 {
        background: red;
    }

    .box4 {
        background: magenta;

    }

    .box5 {
        background: yellow;
    }

    .box6 {
        background: pink;
    }

    .box {

        font-size: 25px;
        padding: 15px;

    }

    .container {
        border: 2px solid black;
        display: flex;

    }
```

```
</style>

<body>
    <div class="container">
        <div class="box box1">One</div>
        <div class="box box2">two</div>
        <div class="box box3">three</div>
        <div class="box box4">four</div>
        <div class="box box5">five</div>
        <div class="box box6">six</div>
    </div>
</body>

</html>
```

Output: we can see the size by minimizing the window size

5. flex:

This is the shorthand for **flex-grow**, **flex-shrink** and **flex-basis** combined. The second and third parameters (flex-shrink and flex-basis) are optional. The default is **0 1 auto**, but if you set it with a single number value, like **flex: 5**;, that changes the **flex-basis to 0%,** so it's like setting **flex-grow: 5**; **flex-shrink: 1**; **flex-basis: 0%;**.

Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
    }

    .box2 {
        background: blue;
        flex: 0 1 200px;
        /* flex: 1 1 200px; it will absorb the extra space */

    }
```

```
    .box3 {
        background: red;
    }

    .box4 {
        background: magenta;

    }

    .box5 {
        background: yellow;
    }

    .box6 {
        background: pink;
    }

    .box {

        font-size: 25px;
        padding: 15px;

    }

    .container {
        border: 2px solid black;
        display: flex;

    }
</style>

<body>
    <div class="container">
        <div class="box box1">One</div>
        <div class="box box2">two</div>
        <div class="box box3">three</div>
        <div class="box box4">four</div>
        <div class="box box5">five</div>
        <div class="box box6">six</div>
    </div>
</body>

</html>
```

Output:

6. align-self:

It makes possible to override the **align-items** value for specific flex items.

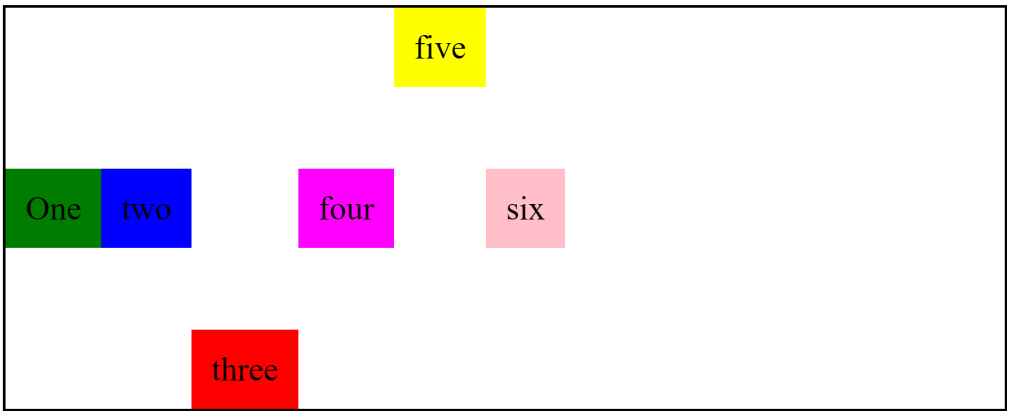
Value	Description
auto	The element inherits its parent container's align-items property, or "stretch" if it has no parent container. This is default.
stretch	The element is positioned to fit the container.
center	The element is positioned at the center of the container
flex-start	The element is positioned at the beginning of the container
flex-end	The element is positioned at the end of the container
baseline	The element is positioned baseline of the container
initial	Sets this property to its default value.
inherit	Inherits this property from its parent element.

Example:

```
<!doctype html>
<html lang="en">
<style>
    .box1 {
        background: green;
    }

    .box2 {
        background: blue;


    }

    .box3 {
        background: red;
        align-self: flex-end;
    }

    .box4 {
        background: magenta;
```

```
    }

    .box5 {
        background: yellow;
        align-self: flex-start;
    }

    .box6 {
        background: pink;
    }

    .box {

        font-size: 25px;
        padding: 15px;

    }

    .container {
        height: 300px;
        border: 2px solid black;
        display: flex;
        align-items: center;
    }
</style>

<body>
    <div class="container">
        <div class="box box1">One</div>
        <div class="box box2">two</div>
        <div class="box box3">three</div>
        <div class="box box4">four</div>
        <div class="box box5">five</div>
        <div class="box box6">six</div>
    </div>
</body>

</html>
```

Output:



Using *margin-auto* property with flex-box:

Setting the margin: auto property on a flex child will take all the available free space accordingly.

Example:

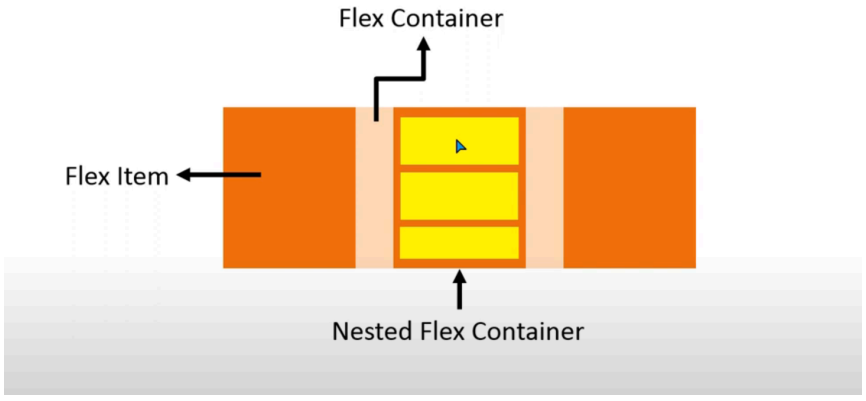
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
```

```
        .container {
            background-color: aquamarine;
            width: 800px;
            height: 400px;
            border: 2px solid red;
            display: flex;
        }

        .box {
            background-color: blueviolet;
            font-size: 20px;
            padding: 10px;

            /* with this the flex-item will come to the center, it will take
margin from all side from all free available space */
/* This way we can put any item to the center of its container by using the child
property */

            /* margin: auto; */

            /* With this, the flex-item will take the margin from left with all
the available free space */
            /* margin-left: auto; */

            /* With this, the flex-item will take the margin from top with all
the available free space */
            /* margin-top: auto;
                margin-left: auto; */
```

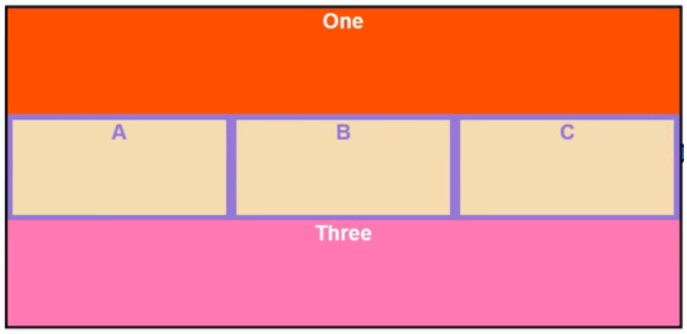
```
        }
    </style>
</head>
<body>



    <div class="container">
        <div class="box">Item</div>
    </div>
</body>
</html>
```

Nested Flex-box:

It's possible to create some pretty complex layouts with flexbox. It's perfectly OK to set a flex item to also be a flex container, so that its children are also laid out like flexible boxes.



Student task: Create the following layout using nested flex-box:



Solution:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        .container {
            width: 600px;
            display: flex;
            flex-direction: column;
            border: 2px solid;
        }
```

```css
.child-one {
    height: 100px;
    background-color: orange;
    border: 2px solid;
    text-align: center;
}

.child-two {
    height: 100px;
    background-color: beige;
    border: 2px solid;
    display: flex;
}


.child-three {
    height: 100px;
    background-color: pink;
    border: 2px solid;
    text-align: center;
}


.grand-child {
    /* width: 200px; */
    flex-grow: 1;
```

```
            border: 5px solid purple;
            text-align: center;
        }
    </style>


</head>

<body>


    <div class="container">
        <div class="child-one">One</div>
        <div class="child-two">
            <div class="grand-child">A</div>
            <div class="grand-child">B</div>
            <div class="grand-child">C</div>
        </div>
        <div class="child-three">Three</div>
    </div>
</body>

</html>
```

Example: Using Nested Flex-box:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        /* General Reset: Applies the box-sizing model to all elements */
        * {
            box-sizing: border-box;
        }

        /* Body Styles */
        body {
            margin: 0;
            /* Remove default margins */
            padding: 0;
            /* Remove default padding */
            height: 100vh;
            /* Viewport height to ensure the body takes full height */
```

```
}

/* Header Styles */
header {
    width: 100%;
    /* Full width of the viewport */
    height: 10vh;
    /* Takes 10% of the viewport height */
    background-color: #3c425a;
    /* Dark background color */
    display: flex;
    /* Flexbox for aligning elements */
    justify-content: space-between;
    /* Space between logo and nav */
    align-items: center;
    /* Center items vertically */
}

/* Logo Styles in the Header */
header img {
    padding: 20px;
    /* Padding around the logo */
    height: 100%;
    /* Makes the logo fit within the header height */
}
```

```css
/* Navigation links in the Header */
#links {
    display: flex;
    /* Flexbox for aligning the links */
    justify-content: space-evenly;
    /* Even space between the links */
    gap: 30px;
    /* Gap between each link */
    color: white;
    /* White text color */
    font-size: 20px;
    /* Text size for links */
    text-transform: uppercase;
    /* Uppercase text */
}

#links>p:hover {
    color: rgb(233, 64, 64);
    cursor: pointer;
}

/* Main Section (Gallery Container) */
main {
    height: 80vh;
    /* Takes 80% of the viewport height */
    width: 100%;
```

```
    /* Full width of the page */
    background-color: aquamarine;
    /* Background color of the main section */
    display: flex;
    /* Flexbox layout for the gallery images */
    flex-wrap: wrap;
    /* Wrap images into multiple rows if needed */
    justify-content: center;
     align-items: center;
     /* Putting all the images in the center of its container */
    gap: 10px;
    /* Small gap between images */
}

/* Gallery Images Styling */
#gallery>img {
    width: 30%;
    /* Image width set to 30% of the container */
    height: 35vh;
    /* Image height set to 35% of viewport height */
    border-radius: 20px;
    /* Rounded corners for the images */
}

/* Footer Section */
footer {
```

```
            height: 10vh;
            /* Footer takes 10% of the viewport height */
            width: 100%;
            /* Full width of the viewport */
            background-color: cadetblue;
            /* Background color for the footer */
            display: flex;
            /* Flexbox for centering content */
            justify-content: center;
            /* Center content horizontally */
            align-items: center;
            /* Center content vertically */
        }

        /* Footer Text Styling */
        footer p {
            color: aliceblue;
            /* Light blue text color */
            font-size: 32px;
            /* Large font size */
        }
    </style>

</head>

<body>
```

```
    <!-- Header Section: Contains logo and navigation links -->
    <header>
        <!-- Logo on the left side -->
        <img src="https://www.chitkara.edu.in/chitkara-university-logo.png"
alt="Chitkara University Logo" />

        <!-- Navigation links on the right side -->
        <nav id="links">
            <p>Home</p>
            <p>Gallery</p>
            <p>About</p>
        </nav>
    </header>

    <!-- Main Section: Contains the image gallery -->
    <main id="gallery">
        <!-- Each image in the gallery -->
        <img src="https://preview.colorlib.com/theme/pentax/img/project/1.jpg"
alt="Gallery Image 1" />
        <img src="https://preview.colorlib.com/theme/pentax/img/blog/2.jpg"
alt="Gallery Image 2" />
        <img src="https://preview.colorlib.com/theme/pentax/img/blog/3.jpg"
alt="Gallery Image 3" />
        <img src="https://preview.colorlib.com/theme/pentax/img/blog/3.jpg"
alt="Gallery Image 4" />
        <img src="https://preview.colorlib.com/theme/pentax/img/blog/2.jpg"
alt="Gallery Image 5" />
```

```
        <img src="https://preview.colorlib.com/theme/pentax/img/project/1.jpg"
alt="Gallery Image 6" />
    </main>

    <!-- Footer Section -->
    <footer>
        <p>This is the footer part</p>
    </footer>
</body>
</html>
```

Output: