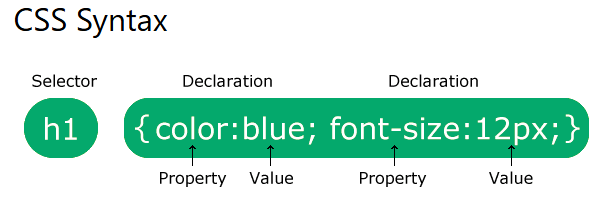
CSS



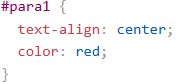
All HTML elements can be considered as boxes.

**NOTE:** CSS selectors are used to "find" (or select) the HTML elements you want to style.

How the CSS rule will be applied to the HTML element with id="para1":

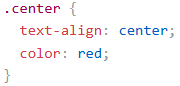


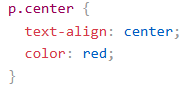
**Note:** An id name cannot start with a number!



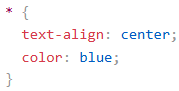
How the CSS rule will be applied to the HTML elements with class="center":

  
**Note:** A class name cannot start with a number!

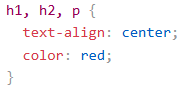




The CSS universal selector (\*) selects all HTML elements on the page.

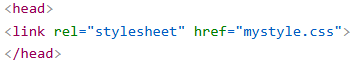


How to group the selectors having same style to minimize the code:



There are three ways of inserting a style sheet:

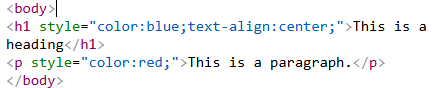
1. **External CSS**



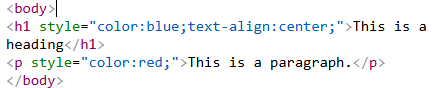
1. **Internal CSS**



1. **Inline CSS:** styles are defined within the “**style**” attribute of the relevant element



Cascading Order - What style will be used when there is **External**, **Internal** and **Inline** styles specified for an HTML element (say **<body>**)



1. **Inline** style will have the highest priority and will override external and internal styles and browser defaults.
2. When only **External** and **Internal** specified **-Internal** will have next priority.
3. External will have least priority.

In CSS, **colors** can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

## RGB Value:

## rgb(red, green, blue) - Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255. For example, rgb(255, 0, 0) is displayed as red.

## To display black, set all color parameters to 0, like this: rgb(0, 0, 0).

## To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

## Shades of gray are often defined using equal values for all the 3 light sources:

## rgb(60, 60, 60)

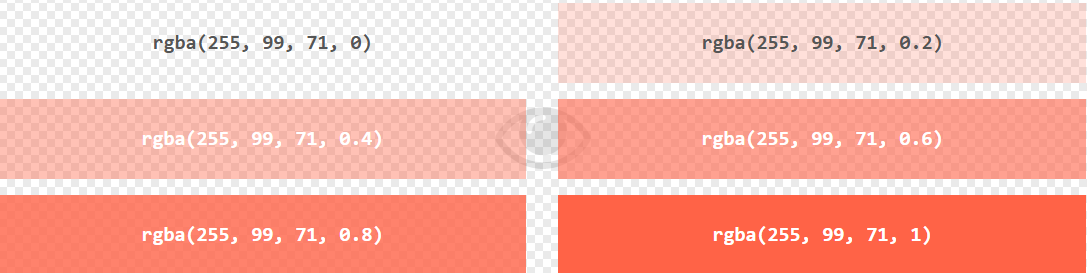
## rgb(120, 120, 120)

## rgb(210, 210, 210)

## RGBA Value

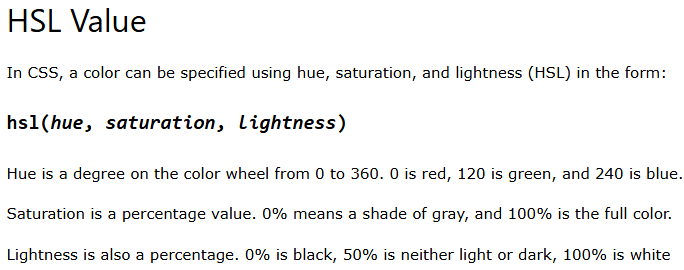
## rgba(red, green, blue, alpha)

## The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all)



## HEX Value

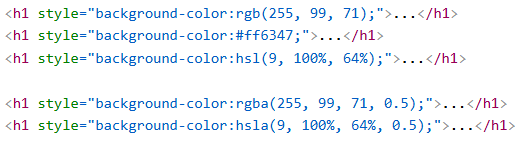
A hexadecimal color is specified with: **#rrggbb**



## HSLA Value

An HSLA color value is specified with: **hsla(hue, saturation, lightness, alpha)**

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all)



CSS properties:

# **CSS Colors**

* color: navy;– defines element text color
* opacity: 0.3;

The opacity property specifies the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent



* text-align: center;- align element text in center of page
* margin-left: 20px;
* border: 2px solid Tomato;

# **CSS Backgrounds**

* background-color: lightblue;
* opacity: 0.3;

The opacity property specifies the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent

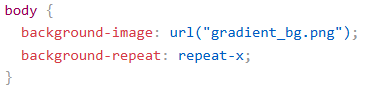


* background-image: url("paper.gif");

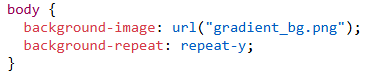
NOTE: By default, the background-image property repeats an image both horizontally and vertically.



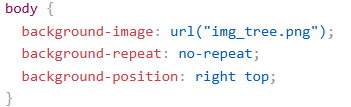
* To repeat the image Horizontally:



* To repeat the image vertically:



* background-position: right top; - Position the bg image in the top-right corner.

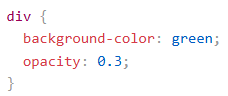


background-attachment: fixed; - Background image should be fixed in a page

background-attachment: scroll; - Bg should scroll with the page.

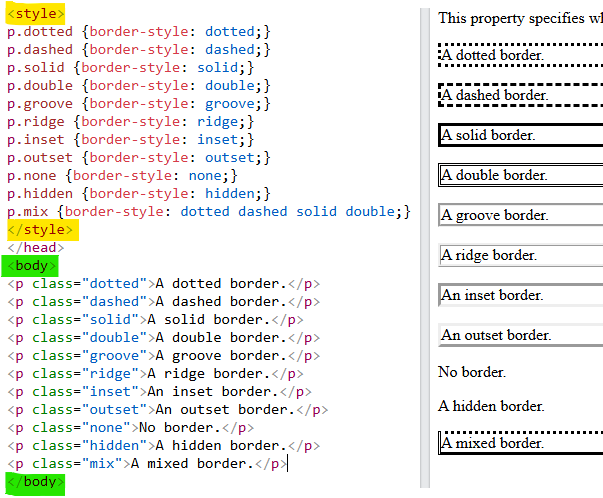
**NOTE:** shorthand property background:





# **CSS Borders**

* **border-style:**



**NOTE:**

/\* **top**-**right**-**bottom**-**left** \*/

border-style: dotted solid double dashed;

/\* **top** – dotted, **right** **&** **left** – solid, **bottom** – double\*/

border-style: dotted solid double;

/\* **top** **&** **bottom** – dotted, **right** **&** **left** – solid \*/

border-style: dotted solid;

* **border-width:**

border-width: 5px;

border-width: 20px 5px; /\* 20px top and bottom, 5px on the sides \*/

border-width: 25px 10px 4px 35px; /\* 25px top, 10px right, 4px botm and 35px left \*/

border-width: medium;

border-width: thick;

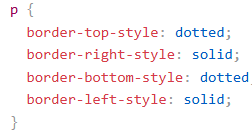
* **border-color:**

border-color: red;

border-color: red green blue yellow;

* **border-radius:**

border-radius: 5px;

****

**NOTE:** shorthand property border:

border: 5px solid red; /\* width, style, color \*/

border-top: 6px solid red; /\* width, style, color \*/

border-right: 6px solid red;

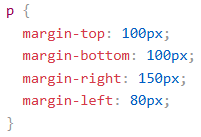
border-bottom: 6px solid red;

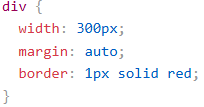
border-left: 6px solid red;

# **CSS Margins**

The CSS margin properties are used to create space around elements, **outside** of any defined borders.

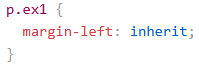
**All the margin properties can have the following values:**

* auto - the browser calculates the margin
* *length* - specifies a margin in px, pt, cm, etc.
* *%* - specifies a margin in % of the width of the containing element
* inherit - specifies that the margin should be inherited from the parent element
* **Note:** Negative values are allowed.
* **margin: auto;**

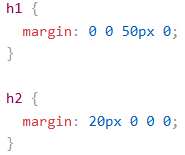


You can set the margin property to auto to horizontally center the element within its container.

* **margin: inherit;**

****

* **Margin Collapse**

****

**Important:**

The <h1> element has a bottom margin of 50px and the <h2> element has a top margin set to 20px.

Common sense would seem to suggest that the vertical margin between the <h1> and the <h2> would be a total of 70px (50px + 20px). But due to margin collapse, the actual margin ends up being 50px.

**NOTE:** shorthand property margin:

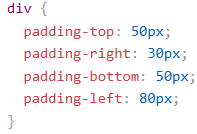
margin: 25px 50px 75px;

# **CSS Padding**

Padding is used to create space around an element's content, inside of any defined borders.

**All the padding properties can have the following values:**

* ***length*** - specifies a **padding** in px, pt, cm, etc.
* ***%*** - specifies a **padding** in % of the width of the containing element
* **inherit** - specifies that the **padding** should be inherited from the parent element



**Note:** Negative values are not allowed.

**NOTE:** shorthand property padding:

padding: 25px 50px 75px;

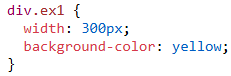
* **Padding and Element Width:**

The CSS width property specifies the width of the element's content area.

****

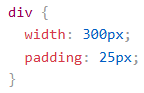
**Output:**

****



**Output:**

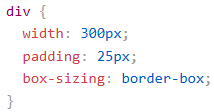


****

Here, the **width** of **<div>** is **300px** but, the actual **width** of the **<div>** will become **350px** (i.e., **300px** + **25px** of **left & right padding**)

* **box-sizing: border-box;**

To keep the width at 300px, no matter the amount of padding, you can use the box-sizing property.



# **CSS Height, Width and Max-width**

The CSS height and width properties are used to set the height and width of an element.

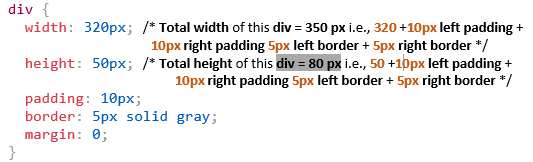
The height and width properties may have the following values:

* auto - This is default. The browser calculates the height and width
* length - Defines the height/width in px, cm, etc.
* % - Defines the height/width in percent of the containing block
* initial - Sets the height/width to its default value
* inherit - The height/width will be inherited from its parent value

**Important:**

When you set the **width** and **height** properties of an element with CSS, you just set the **width** and **height** of the **content area**.

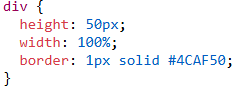
To calculate the total width and height of an element, you must also include the padding and borders.



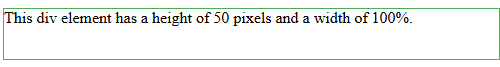
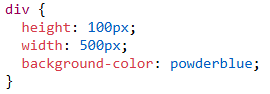


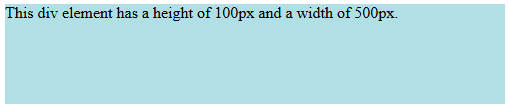
**Output:**

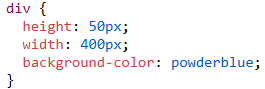


****

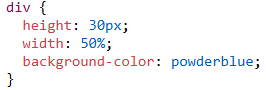
**Output:**

**  
**

****

****

****

****

****

**Note:** Remember that the height and width properties do not include **padding**, **borders**, or **margins**!

* **max-width:**

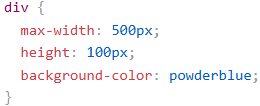
The max-width property is used to **set the maximum width of an element**.

max-width: 500px;

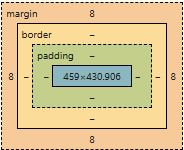
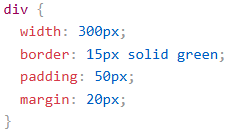
max-height: 500px;

min-width: 500px;

min-height: 500px;

****

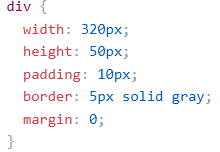
# **CSS Box Model**

  
Demonstration of the box model:  


**Important:** When you set the **width** and **height** properties of an element with CSS, you just set the **width** and **height** of the **content area**.

To calculate the total **width** and **height** of an element, you must also include the **padding** and **borders**.

Example:

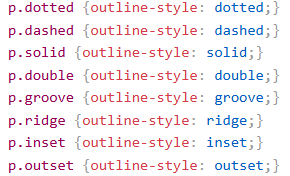
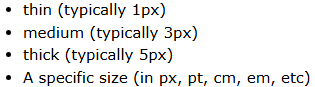
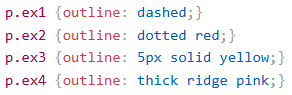
  
**Total element width** = width + left padding + right padding + left border + right border  
**Total element height** = height + top padding + bottom padding + top border + bottom border

**NOTE:** **margin** is not included in the **actual size of the box**. The box's total **width** and **height** **stops** at the border.

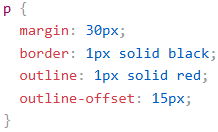
# **CSS Outline**

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the

element "stand out".

* **outline-style:**  
  
* **outline-width:  
  **outline-width: thin;  
  outline-width: medium;  
  outline-width: thick;  
  outline-width: 4px;
* **outline**-**color:**outline-color: red;
* **NOTE:** shorthand property outline:  
  
* **outline-offset:**

The outline-offset property adds space between border of an element and its outline is transparent

****

# **CSS Text Alignment**

The text-align property is used to set the horizontal alignment of a **text**.

A text can be **left or right aligned**, **centered**, or **justified**.

* **text-align:**

text-align: center;

text-align: right;

text-align: left;

text-align: justify; - each line is stretched so that every line has equal width,

and the left and right margins are straight.

* **text-align-last:**

**Ex: <p>** element yokka last line right side undala? Center undala? Or even to page undala? Ani define chesthundi

text-align-last: right;

text-align-last: center;

text-align-last: justify;

* **vertical align:**

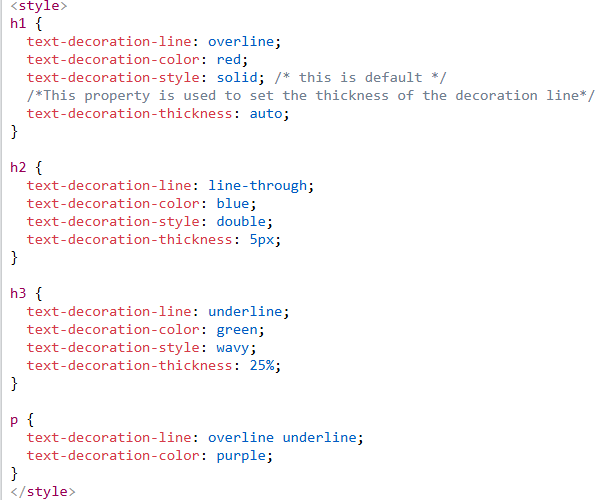


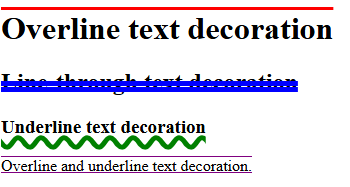
vertical-align: top;

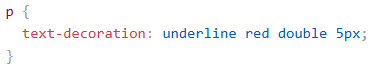
vertical-align: bottom;

vertical-align: middle;

# **CSS Text Decoration** The text-decoration property is used to add a decoration line to text.

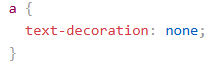
  
**Output:**

  
**NOTE:** shorthand property text-decoration:



**Remember,**

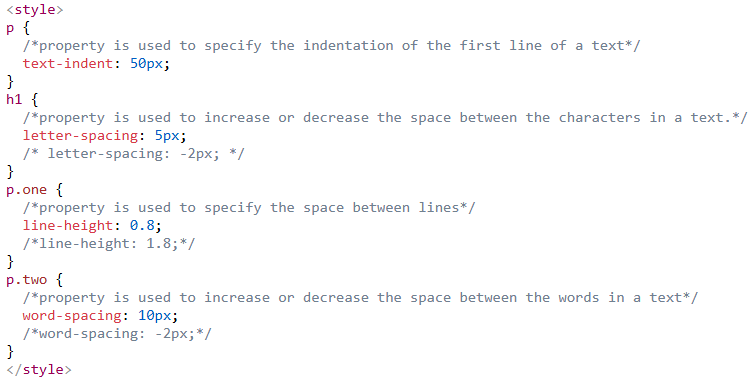
All links in HTML are underlined by default. Sometimes you see that links are styled with no underline. The text-decoration: none; is used to remove the underline from links, like this:



# **CSS Text Transformation** The text-transform property is used to specify uppercase and lowercase letters in a text or capitalize the first letter of each word.

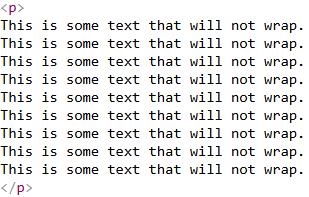
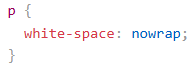
# **CSS Text Spacing**

* text-indent
* letter-spacing
* line-height
* word-spacing
* white-space

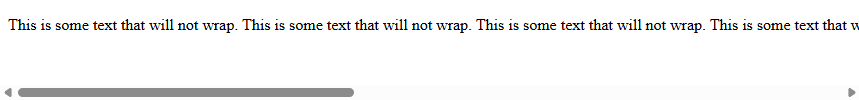
****

The white-space property specifies how to disable text wrapping inside an element.

**Example:**

  
****

**Output:**

****  
**Fig:** window size ni dhati paragraph text unte, ah text next line ki rakunda sameline lone vachela chesthundi scroll-bar add chesi.

# **CSS Text Shadow**

The text-shadow property adds horizontal shadow (2px) and the vertical shadow (2px)to text.

h1 {  
  text-shadow: 2px 2px;  
}

**Add a color (red) to the shadow:**

h1 {  
  text-shadow: 2px 2px red;  
}

**Add a blur effect (5px) to the shadow:**

h1 {  
  text-shadow: 2px 2px 5px red;  
}

**More Text Shadow Examples:**

h1 {  
  color: white;  
  text-shadow: 2px 2px 4px #000000;  
}

**output:**



h1 {  
  text-shadow: 0 0 3px #ff0000;  
}

**output:**



h1 {  
  text-shadow: 0 0 3px #ff0000, 0 0 5px #0000ff;  
}

**output:**



h1 {  
  color: white;  
  text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;  
}

**output:**



# **CSS Fonts**

In CSS, we use the font-family property to specify the font of a text.

**In CSS there are five generic font families:**

* **Serif – (Font Names -Times New Roman, Georgia, Garamond)**
* **Sans-serif – (Font Names -Arial, Verdana, Helvetica)**
* **Monospace – (Font Names -Courier New, Lucida Console, Monaco)**
* **Cursive – (Font Names -Brush Script M7, Lucida handwriting)**
* **Fantasy** **– (Font Names -Copperplate, Papyrus)**

**Note**: If the font name is more than one word, it must be in quotation marks

.p1 {  
  font-family: "Times New Roman", Times, serif;  
}

# **CSS Web Safe Fonts**

Web safe fonts are fonts that are universally installed across all browsers and devices.

The following list are the best web safe fonts for HTML and CSS:

* Arial (sans-serif)
* Verdana (sans-serif)
* Tahoma (sans-serif)
* Trebuchet MS (sans-serif)
* Times New Roman (serif)
* Georgia (serif)
* Garamond (serif)
* Courier New (monospace)
* Brush Script MT (cursive)

MUI – Material UI

* Popular **React UI framework** that provides set of **components and tools** to build modern and responsive web applications based on **Google’s Material Design** guidelines.

### 🔧 What MUI Provides:

#### 1. **Pre-Built UI Components**

MUI offers a huge library of customizable and production-ready UI components, such as:

* ✅ Buttons, TextFields, Checkboxes, Radios
* ✅ Cards, Dialogs, Tooltips, Snackbars
* ✅ DataGrids (tables), Menus, AppBars
* ✅ Tabs, Accordions, Avatars, Chips, etc.

These are all built with Material Design styles and behavior.

#### 2. **Theming & Customization**

* You can fully **customize colors, typography, spacing, and shapes** via MUI's theme system.
* Supports **light/dark mode**, and you can override styles with styled components or sx prop.

#### 3. **Responsiveness**

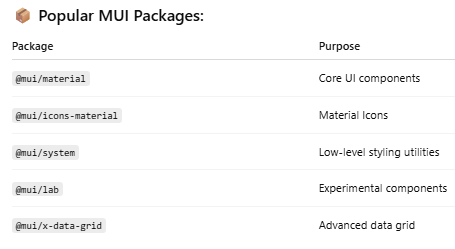
* Built-in support for **responsive design** using breakpoints.
* Components adapt to different screen sizes without much effort.

#### 6. **Styling Solutions**

* You can style components using:
  + sx prop (short, inline styling)
  + styled() API
  + Emotion or styled-components
  + Traditional CSS classes

#### 7. **Icon Library**

* MUI provides @mui/icons-material — an official Material Icons set you can easily use.

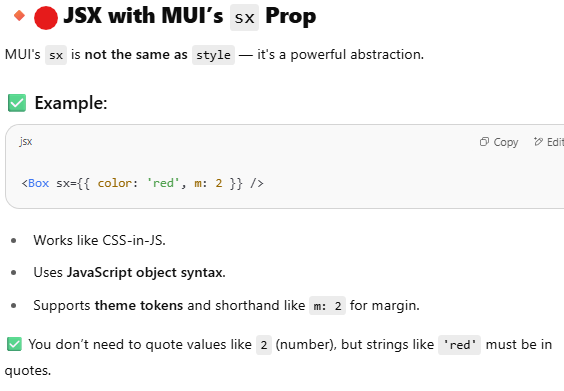


When writing **CSS in HTML, JSX (like React)**, or in **<style>** tags, the syntax differs slightly depending on the context.







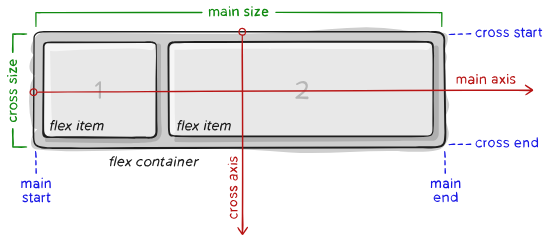


# **Flexbox**

Open - <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

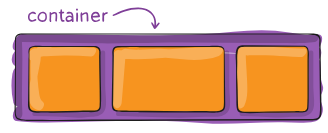
Since flexbox is a whole module and not a single property, it involves a lot of things including its whole set of properties.

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”).



**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).

## Properties for the Parent (flex container):



#### display

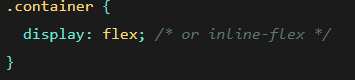


Fig: This defines a **flex container**; inline or block depending on the given value.

* Every HTML element has a default display value, depending on what type of element it is. The **default** **display** **value** for most elements is block or inline.
* The display property is used to change the default display behavior of HTML elements.
* The display property has many values:

**block**: Displays an element as a block element

**inline**: Displays an element as an inline element

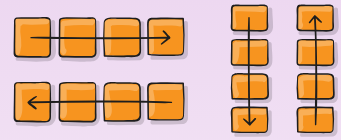
**flex**: Displays an element as a block-level flex container

**grid**: Displays an element as a block-level flex container

Example:



#### flex-direction



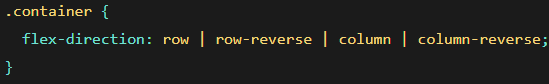
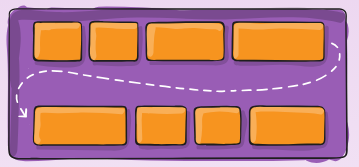


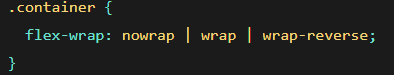
Fig: This establishes the **main-axis**, thus defining the direction flex items are placed in the flex container.

* **row (default):** left to right in ltr; right to left in rtl
* **row-reverse:** right to left in ltr; left to right in rtl
* **column:** same as row but top to bottom
* **column-reverse:** same as row-reverse but bottom to top

#### flex-wrap



**NOTE**: By default, **flex items** will all try to fit onto one line.



* **nowrap (default):** all flex items will be on one line
* **wrap:** flex items will wrap onto multiple lines, from top to bottom.
* **wrap-reverse:** flex items will wrap onto multiple lines from bottom to top.

#### flex-flow

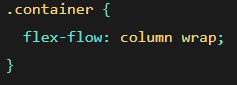
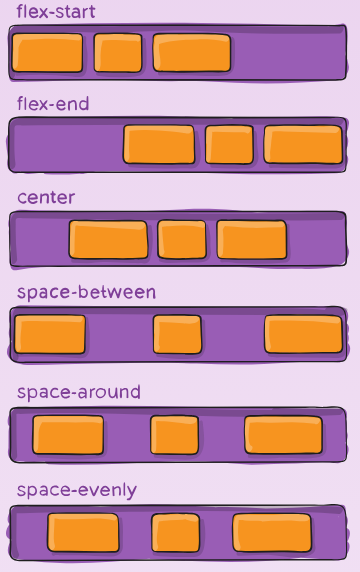


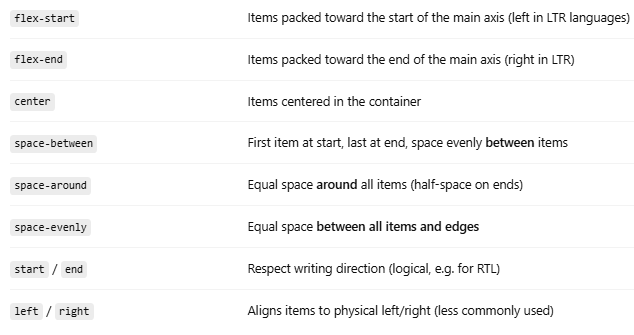
Fig: This is a shorthand for the ***flex-direction*** and ***flex-wrap*** properties

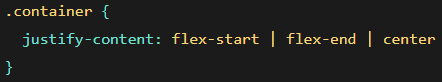
#### justify-content



It **controls how flex items are aligned along the main axis** (usually **horizontal**) inside a flex container.

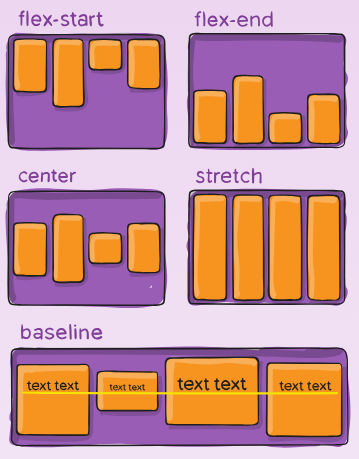
It only works if the container has **display: flex** or **display: grid.**



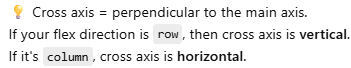
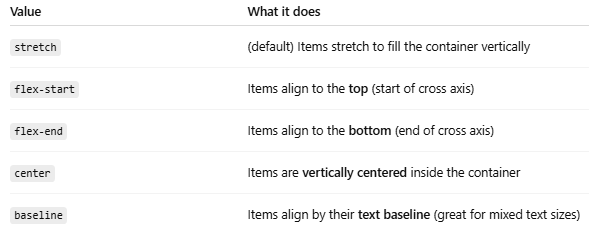


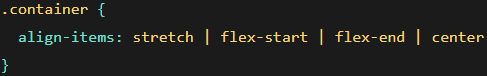
NOTE: The value **space-between** never got support from some versions of **Edge**, and the values **start**/**end**/**left**/**right** aren’t in Chrome yet. The safest values are **flex-start**, **flex-end**, and **center**.

#### align-items



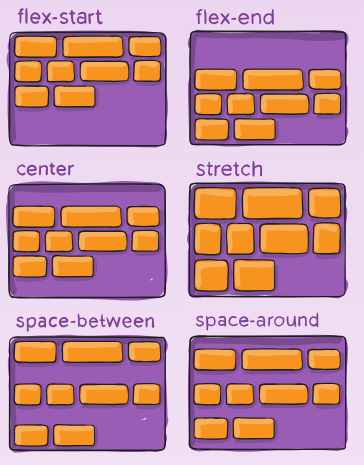
It controls how **flex items are aligned vertically** (if flex-direction: row) **within the container** on the **cross axis**.



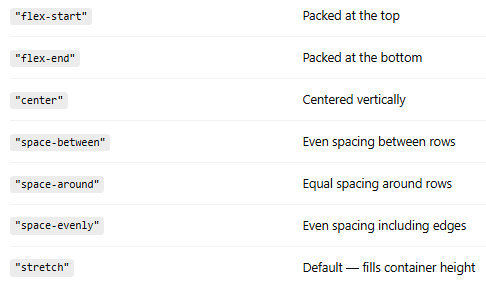
* **stretch (default):** stretch to fill the container (still respect min-width/max-width)
* **flex-start / start / self-start:** items are placed at the start of the cross axis. The difference between these is subtle, and is about respecting the flex-direction rules or the writing-mode rules.
* **flex-end / end / self-end:** items are placed at the end of the cross axis. The difference again is subtle and is about respecting flex-direction rules vs. writing-mode rules.
* **center:** items are centered in the cross-axis
* **baseline:** items are aligned such as their baselines align

#### align-content

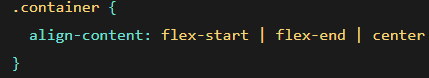


This aligns a flex container’s lines within when there is extra space in the **cross-axis**,

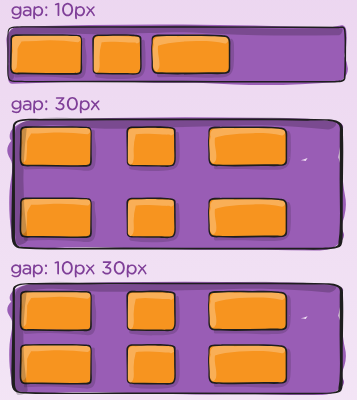
similar to how **justify-content** aligns individual items within the main-axis.



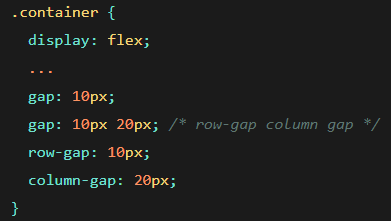
**Note:** This property only takes effect on multi-line flexible containers, where **flex-wrap** is set to either **wrap** or **wrap-reverse**). A single-line flexible container (i.e. where **flex-wrap** is set to its default value, **no-wrap**) will not reflect **align-content**.



#### gap, row-gap, column-gap



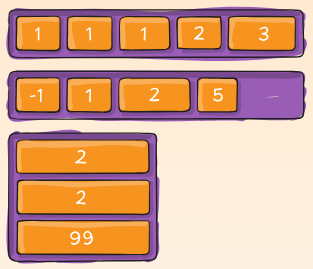
It explicitly applies that spacing only between items not on the outer edges.

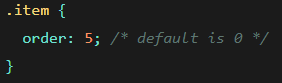


## Properties for the Children (flex items):

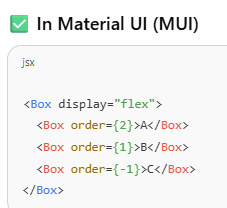
## 

#### order

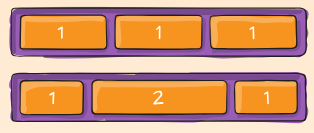


By default, flex items are laid out in the source order.  The order property controls the **visual order** of flex items inside a container **without changing the HTML structure**. 

🧠 All flex items start with order: 0 by default. Items with **lower order values** appear **first**.

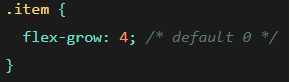


#### flex-grow

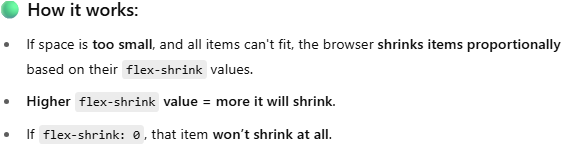
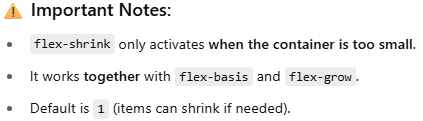
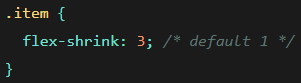


Controls **how much a flex item grows** **relative** to the rest when there’s **extra space**.

**Default: 0** - Item **won’t grow** beyond its content size.



#### flex-shrink

**controls how much a flex item will shrink** relative to the rest of the flex items **when there's not enough space in the container**.  
  
  


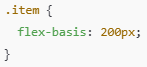
#### flex-basis



Defines the **initial size** of a flex item **before** growing or shrinking.

If set to **0,** the extra space around content isn’t factored in.

Default: auto (uses content size)



Item starts at 200px width (or height if flex-direction: column), then grows/shrinks from there.

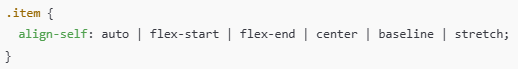
#### flex

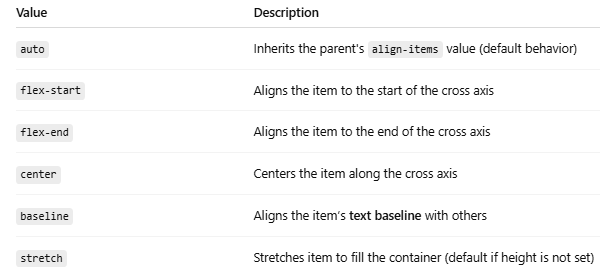


#### 

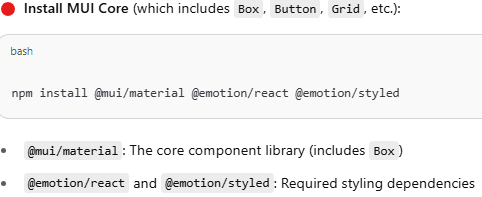
#### align-self

It’s used to **override the parents align-items property** **for a single flex item**. This allows **individual alignment** of a flex child along the **cross axis** (usually vertical).



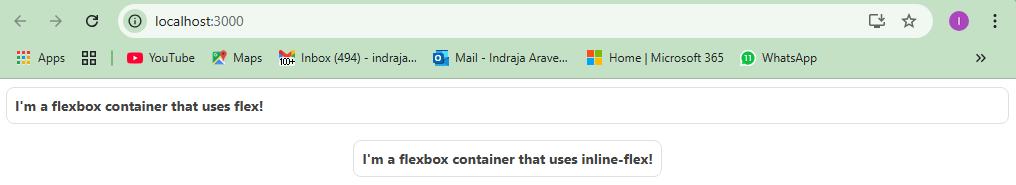


<https://mui.com/system/flexbox/>



Open the page - <https://mui.com/system/flexbox/>

* Code overview on **display** CSS style code:



**Fig:** This is the output we get when we run **Display()** component code.

* You are rendering **two MUI <Box /> components**, each styled as a **Flexbox container**:



* Imports
*   
  **React**: Needed to use JSX and create components.  
  **Box**: A versatile MUI component used for **layout** and **styling**, acting like a **<div>** with extended styling support through the **sx** prop.
* Outer <div style={{ width: '100%' }}>  
  This wraps everything in a full-width container.
* First Box — display: 'flex'  
  display: 'flex' — turns it into a **block-level flex container**theme.applyStyles('dark', {...}) — applies alternate styles in **dark mode**

**This box behaves like a regular flex container**, stretching to fill available space.

**NOTE:**

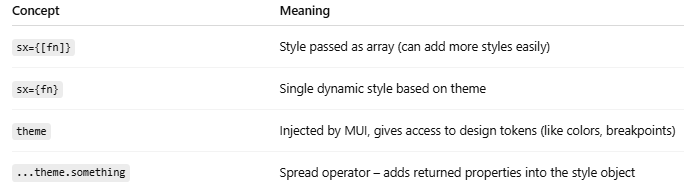
The sx prop **accepts an array** of style objects or functions.  
Inside the array: you passed **one function** that receives theme.  
This setup allows you to later **add more styles** to the array if needed:  
 sx={[style1, style2, style3]}

The theme comes from **MUI’s ThemeProvider**. When you pass a function to sx, MUI will **automatically provide the theme object**. You **didn’t import or define** theme — it is injected by MUI behind the scenes.

. . .theme.applyStyles('dark', {...})- manam pass chesina CSS object ni parent CSS object loki spread/add chesthundi.

* Second Box — display: 'inline-flex'  
  Applies same styles as above  
  **Key Difference**: display: 'inline-flex' makes the box behave like an **inline-level** element while still using **Flexbox layout inside**.  
  That means it **won't break the line** and **can sit beside inline or inline-block elements**.  
  NOTE:

Directly passed a function — not wrapped in an array.  
Perfectly valid — simpler when you're applying only one dynamic style based on the theme.



* Code overview on **flex-direction** CSS style code:

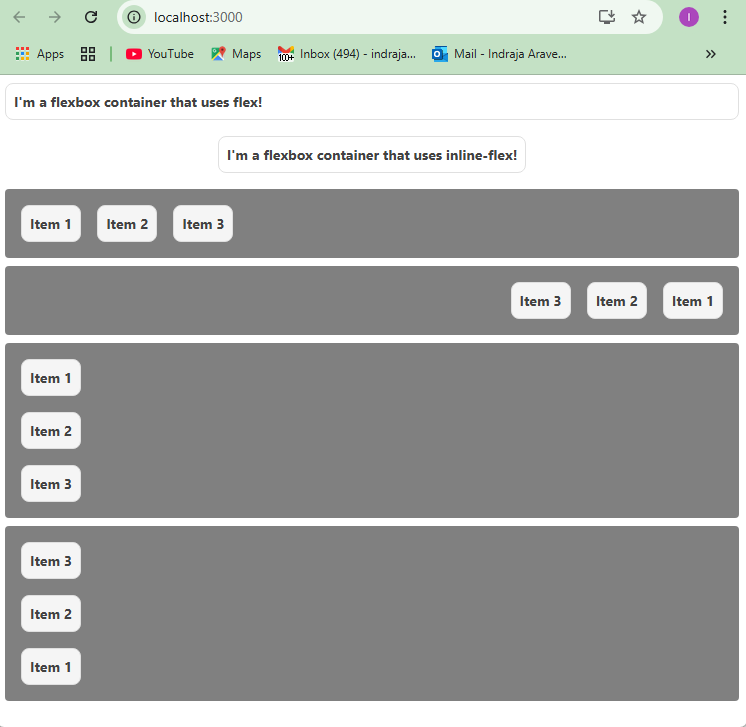
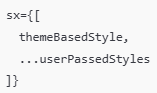
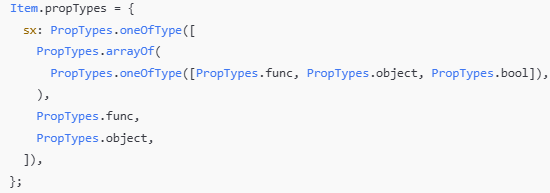


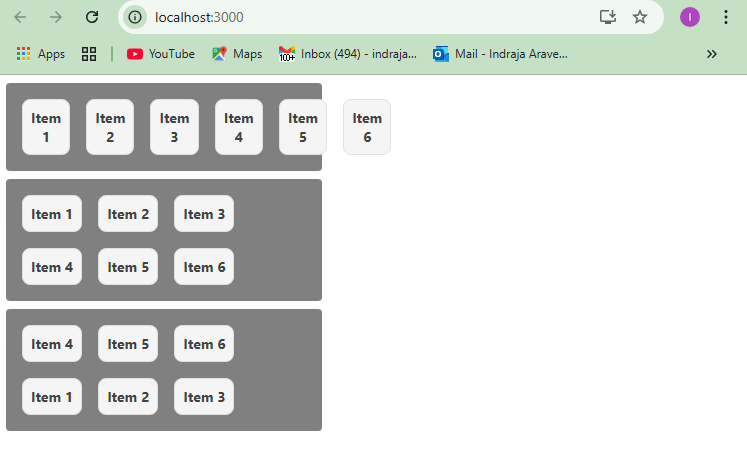
Fig: This is the output we get, when we run **FlexDirection()** component code.

* + ...(Array.isArray(sx) ? sx : [sx]): This line **ensures that sx is always an array**, even if the user passed a single style object or function.
  + Because the **sx** prop in MUI **can take an array**, which allows merging multiple styles.  
    **Let’s say a user passes:**  
     **Now inside your component, you want to merge this with your internal styles:**  
      
    **But here’s the catch:** If sx is an **object**, doing ...sx will throw an error because objects can’t be spread like arrays in an array context.  
    **So we check:** Array.isArray(sx) ? sx : [sx]
  + **{...other}** - It spreads **all remaining props** (other than sx) onto the <Box /> component.  
    Example:  
      
    **sx** gets separated.  
    **id** and **title** go into **other**.
  + **PropTypes** Validation:



Ensures **sx** can be a **style object**, **function**, **array**, or **boolean**.  
Helps during development by catching prop type mismatches.

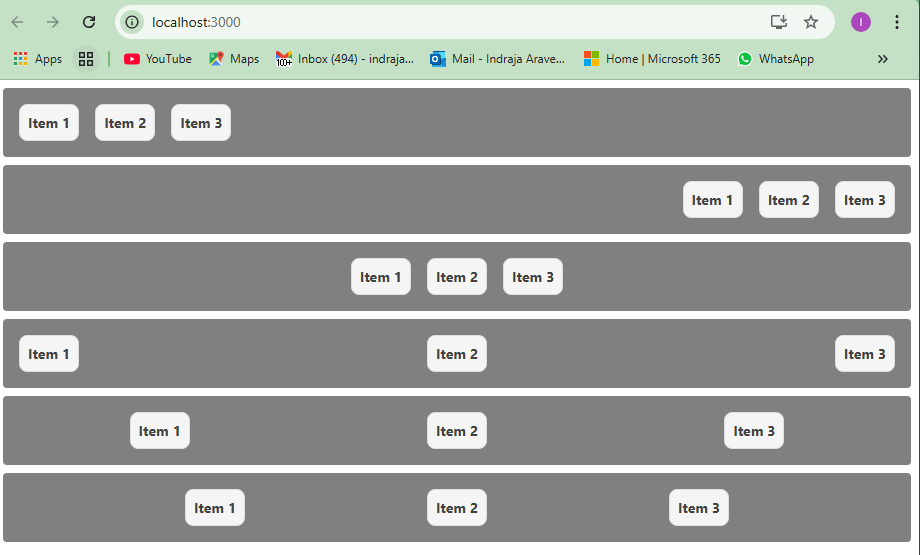
* Main Component: **FlexDirection**Wraps the whole layout in a **<div>** with 100% width.
* Code overview on **flex-wrap** CSS style code:



**Fig:** This is the output we get, when we run **FlexWrap()** component code.

**NOTE:** If you observer the **containers** of **flex-direction** and **flex-wrap**, the **width** of **flex-wrap** **containers** is **fixed** by CSS property **maxWidth: 300**

* Code overview on **justify-content** CSS style code:

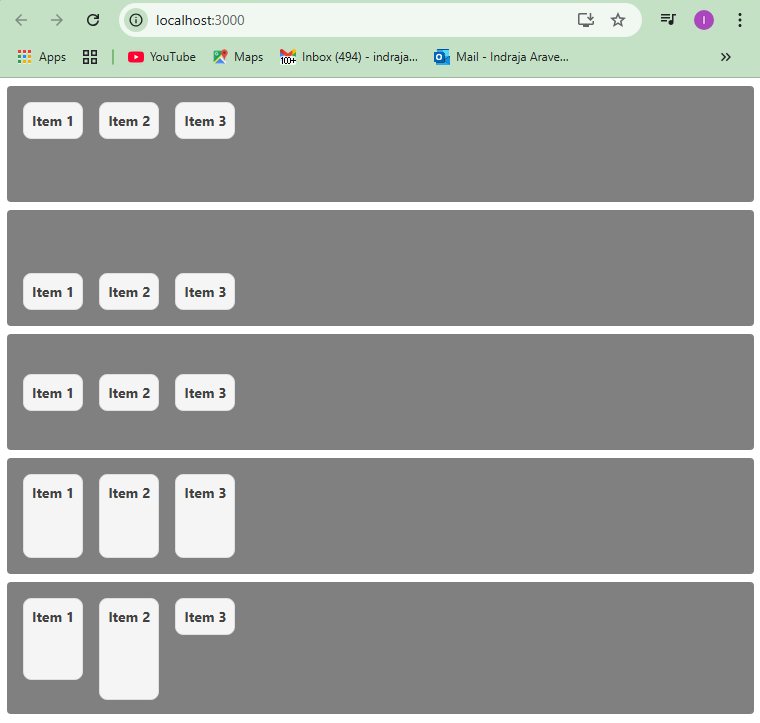


**Fig:** This is the output we get, when we run **JustifyContent()** component code.

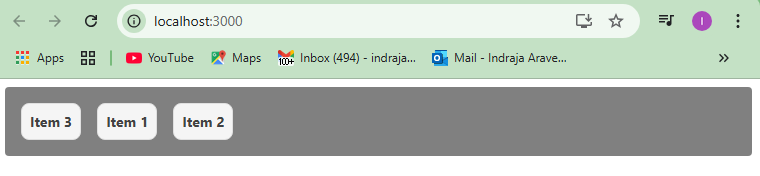
justifyContent: 'space-between' - First item at start, last item at end, space evenly between rest of items

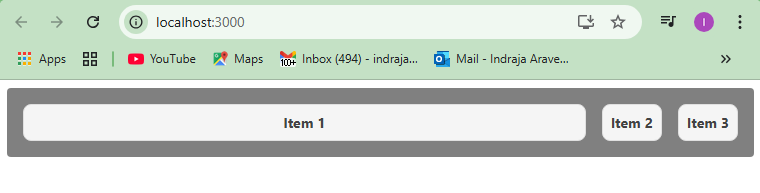
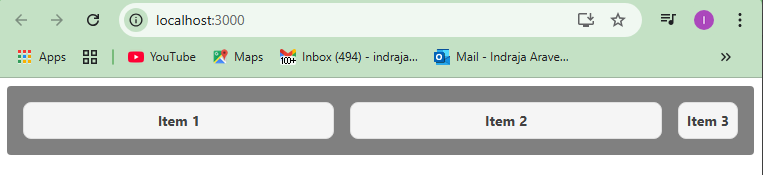
justifyContent: 'space-around' - Equal space around all items (half-space on edges)

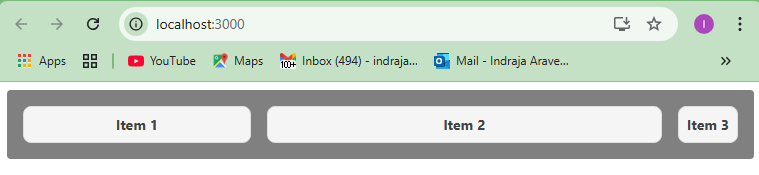
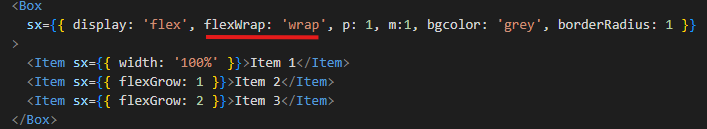
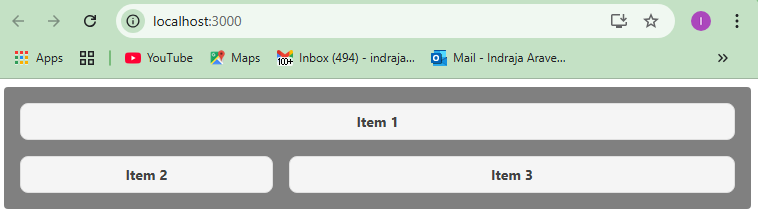
justifyContent: 'space-evenly' - Equal space around all items and edges

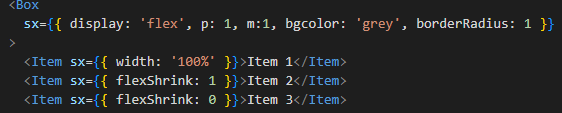
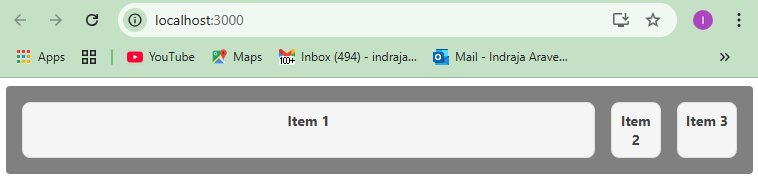
* Code overview on **align-items** CSS style code:  
    
  **Fig:** This is the output we get, when we run **AlignItems()** component code.
* Code overview on **align-content** CSS style code:  
    
  **Fig:** This is the output we get, when we run **AlignContent()** component code.

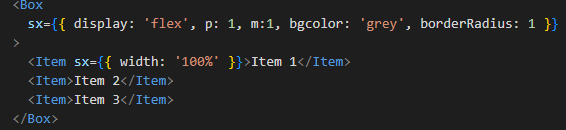
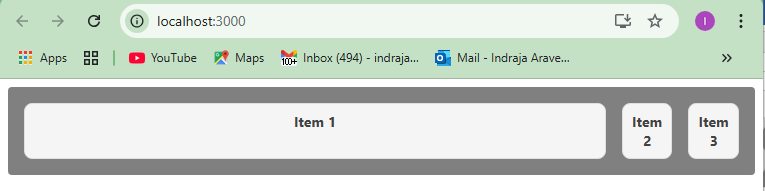
**Note:** ‘**alignContent’** property only takes effect on multi-line flexible containers (i.e., where **flex-wrap** is set to either **wrap** or **wrap-reverse**).   
A single-line flexible container (i.e. where **flex-wrap** is set to its default value, **no-wrap**) will not reflect **align-content**.

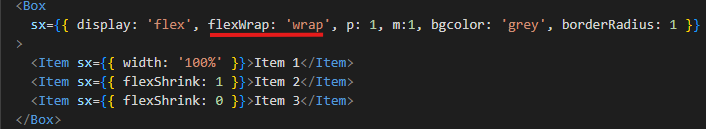
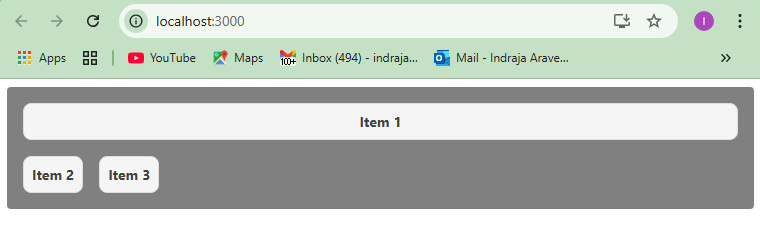
* Code overview on **order** CSS style code:  
    
  **Fig:** This is the output we get, when we run **Order()** component code.
* Code overview on **flex-grow** CSS style code:

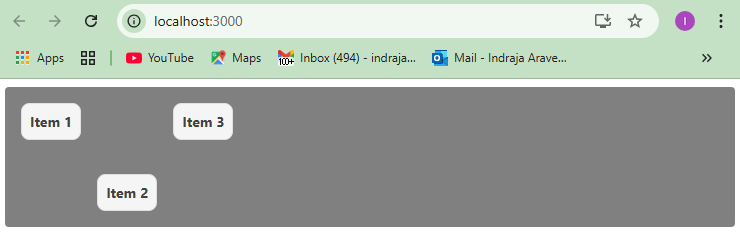
  
  
**Fig-1:** This is the output we get, when we run **FlexGrow()** component code with above JSX + sx prop.  
  
  
**Fig-2:** This is the output we get, when we run **FlexGrow()** component code with above JSX + sx prop.  


  
**Fig-3:** This is the output we get, when we run **FlexGrow()** component code with above JSX + sx prop.  
  
  
**Fig-4:** This is the output we get, when we run **FlexGrow()** component code with above JSX + sx prop.

* Code overview on **flex-shrink** CSS style code:  
    
    
  **Fig:** This is the output we get, when we run **FlexShrink()** component code with above JSX + sx prop.  
  **NOTE:** **Higher flex-shrink value = more it will shrink**. If flex-shrink: 0, that item **won’t shrink at all.**

**Suppose**, when there is no usage of **flexShrink** property.  
  
**Fig:** No **flexShrink** property  
**Remember,** container lo enough space lekuntene item will shrink otherwise it wont shrink

  
  
**Fig:** This is the output we get, when we run **FlexShrink()** component code with above JSX + sx prop.

* Code overview on **align-self** CSS style code:  
    
  **Fig:** This is the output we get, when we run **AlignSelf()** component code.