



**Topic : Flex**

## **1. FLEX**

The Flexbox layout provides efficient way to layout, align and distribute space among items in a container. This is really helpful when the size of the elements in the container is unknown and/or dynamic.

Using the flex in a container, gives the **ability to alter its items width/height and order as well to best fit in the space available**. A flex container expands items to fill available free space, or shrinks them to prevent overflow.

The flex is a value for the display property. It has to be **provided in the container** for the flex to work. Only if it is defined inside the container, flex properties will work. **Flex properties are defined on the child elements**.

To make the container to be flex, add this property in the container: `display: flex;` or `display: inline-flex;` for the inline variation.

## **2. FLEX DIRECTION**

The **flex-direction** property defines in which **direction** the container lays out the flex-items.

It may take 4 values:

- **row** - default
- **column**
- **row-reverse**
- **column-reverse**

The flex-direction property lays out the flex-items either **horizontally, vertically or reversed** in both directions.

The horizontal axis is called **main-axis** and vertical axis is called **cross axis**.

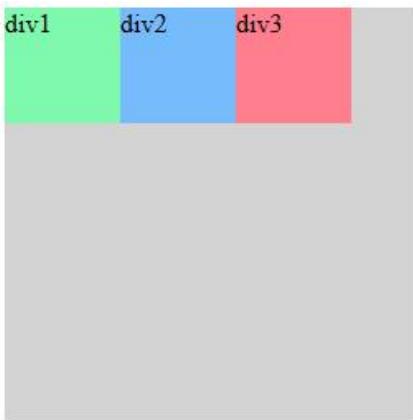
Eg., for the layout like this:

```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

we can use like this:

```
.flex-container { display:flex; width:250px; height:250px; background-color:lightgrey; }
.flex-container div { width:70px; height:70px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; }
```

the layout will look like this:



## 2.1. row

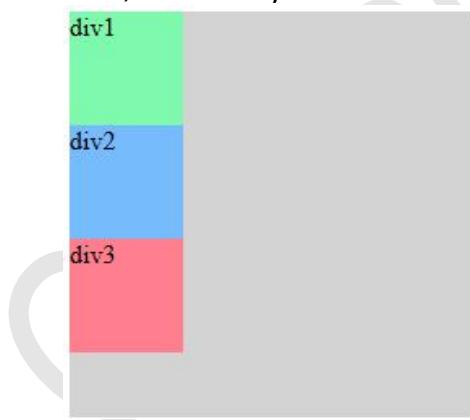
The **row** value stacks the flex items horizontally, from left to right.

For **forward direction**, add the `flex-direction: row;` to the flex-container class. It will not change anything as **this is the default value** and the layout will still look the same.

## 2.2. column

The **column** value stacks the flex items vertically, from top to bottom.

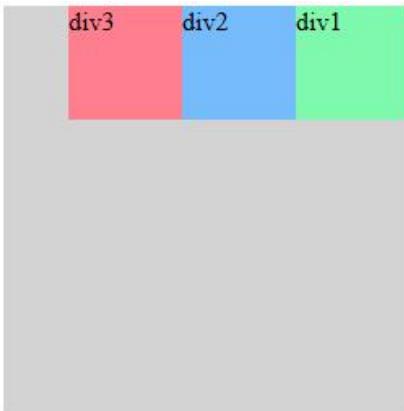
For **downward direction**, add the `flex-direction: column;` to the flex-container class, and the layout will look like this:



## 2.3. row-reverse

The **row-reverse** value stacks the flex items horizontally, from right to left.

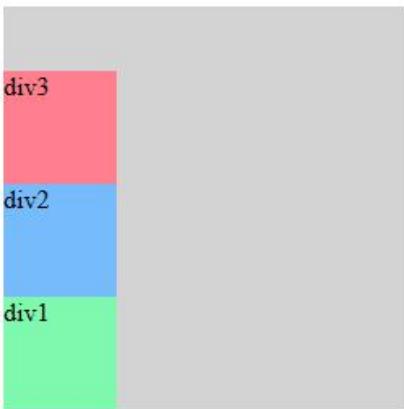
For **backward direction**, add the `flex-direction: row-reverse;` to the flex-container class, and the layout will look like this:



## 2.4. column-reverse

The **column-reverse** value stacks the flex items vertically, from bottom to top.

For **upward direction**, add the `flex-direction: column-reverse;` to the flex-container class, and the layout will look like this:



## 3. ORDER

The **order** property is used to **specify the order of the flex items** in the flex container. This property value must be a whole number. By default, the number is 0(zero).

The higher the number the latter would the flex item appear in the flex container.

The flex items are displayed in the order like:

- first the items **not having order property** or `order:0;` property are displayed in sequence in which they appear in the source order.
- then the items are displayed in **ascending order of the value of the order property.**  
The items having the **same order value** are displayed in the sequence in which they appear in the source order.

Eg., for the layout:

```
<div class="flex-container">
  <div class="div1">div1</div>
  <div class="div2" style="order:0;">div2</div>
  <div class="div3" style="order:3;">div3</div>
```

```

<div class="div1" style="order:3;">div4</div>
<div class="div2">div5</div>
<div class="div3" style="order:2;">div6</div>
</div>

```

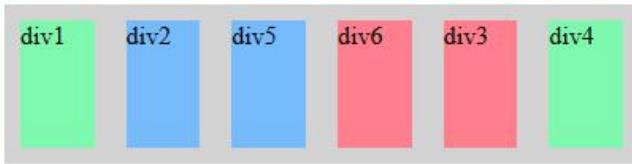
the css is applied to this as:

```

.flex-container { display:flex; width:400px; height:100px; background-color:lightgrey; }
.flex-container div { width:70px; min-height:70px; margin:10px; }
.div1 { background-color:#7ff9ae; }
.div2 { background-color:#76bbfc; }
.div3 { background-color:#ff7f8e; }

```

will display the flex items like:



## 4. FLEX WRAP

The **flex-wrap** property specifies whether the flex items should wrap or not.

By default, flex items try to fit into one line. This property allows you to change that and allow the items to flow into multiple lines as needed with this property.

- **nowrap** - default
- **wrap**
- **wrap-reverse**

For the layout:

```

<div class="flex-container">
  <div class="div1">div1</div>
  <div class="div2">div2</div>
  <div class="div3">div3</div>
  <div class="div1">div4</div>
  <div class="div2">div5</div>
  <div class="div3">div6</div>
</div>

```

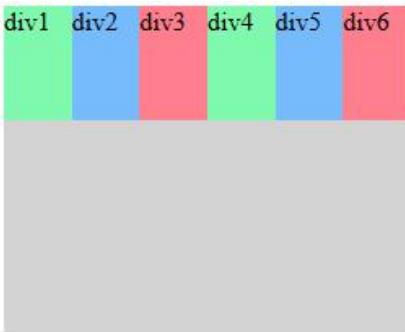
and with the following styles applied to them:

```

.flex-container { display:flex; width:250px; height:200px; background-color:lightgrey; }
.flex-container div { width:70px; height:70px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; }

```

the layout(original container width is 250px and original total div's width is 420px) will look like this:



#### 4.1. nowrap

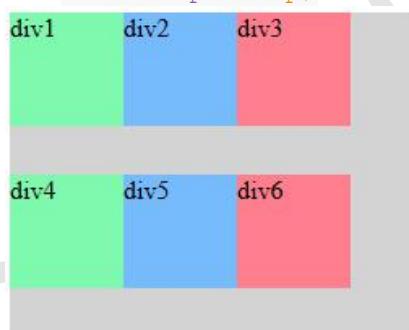
The **nowrap** value specifies that the flex items will not wrap, i.e. it will make all the flex-items appear in a single line, no matter how many items are present inside the flex-container. It is a **default value** but you can use `flex-wrap: nowrap;` as well in the flex-container class.

#### 4.2. wrap

The **wrap** value specifies that the flex items will wrap if necessary, i.e. if the total flex items width is larger than the width of the container, the items will arrange themselves in the next column.

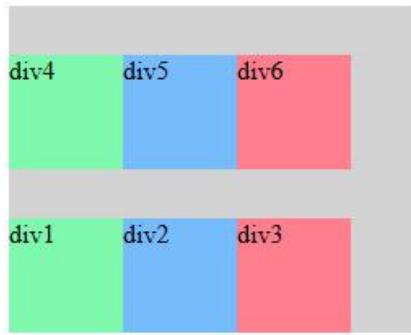
The **vertical spacing of the items will be automatically decided** by this property.

Use `flex-wrap: wrap;` in the flex-container class to wrap the items like this:



#### 4.3. wrap-reverse

The **wrap-reverse** value specifies that the flex items will wrap in reverse order if necessary, i.e. if the total flex items width is larger than the width of the container, the items will arrange themselves in the next column but the items will start from the bottom of the container. Use `flex-wrap: wrap-reverse;` in the flex-container class to wrap the items like this:



## 5. FLEX GROW

The **flex-grow** property specifies the ratio with which a **flex item grows relative to the other flex items**, when there is some extra space available. This controls the extend how much a flex-item grows, with respect to other flex items.

This takes up any value from **0(zero) to any positive number**.

**0(zero)** means that the flex items' width would not change.

Eg., for a simple layout:

```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

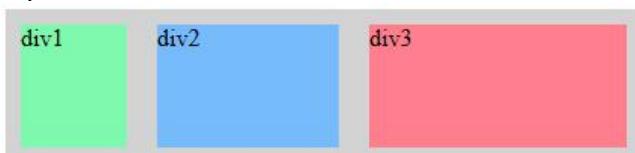
the css applied is:

```
.flex-container { display:flex; width:270px; height:100px; background-color:lightgrey; }
.flex-container div { width:70px; margin:10px; }
#div1 { background-color:#7ff9ae; flex-grow:0; }
#div2 { background-color:#76bbfc; flex-grow:2; }
#div3 { background-color:#ff7f8e; flex-grow:4; }
```

this will look like this:



Now, if the width of the flex-container is increased by 150px to **width:420px**; then the layout will now look like this:



Here, only the div2 and div3 grows, when the width of the container is increased. Also, div3 increases twice as much as div2. Now, width of **div1 is 70px, div2 is 120px and div3 is 170px**.

## 6. FLEX SHRINK

The **flex-grow** property specifies the ratio with which a **flex item shrinks relative to the other flex items**, when there is some extra space available. This is just opposite to flex-grow. Here, also the range of value is from **0(zero) to any positive number**. Zero means width would not change.

Eg., for a simple layout:

```
<div class="flex-container">
  <div id="div1">div1</div>
  <div id="div2">div2</div>
  <div id="div3">div3</div>
</div>
```

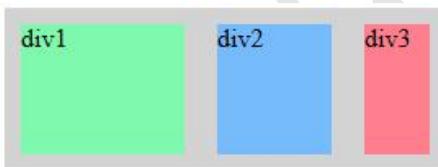
the css applied is:

```
.flex-container { display:flex; width:360px; height:100px; background-color:lightgrey; }
.flex-container div { width:70px; margin:10px; }
#div1 { background-color:#7ff9ae; flex-shrink:0; }
#div2 { background-color:#76bbfc; flex-shrink:2; }
#div3 { background-color:#ff7f8e; flex-shrink:4; }
```

this will look like this:



Now, if the width of the flex-container is decreased by 90px to **width:270px;**, then the layout will now look like this:



Here, only the div2 and div3 shrink, when the width of the container is decreased. Also, div3 decreases twice as much as div2. Now, width of **div1 is 100px, div2 is 70px and div3 is 40px**.

## 7. FLEX - JUSTIFY CONTENT

The **justify-content** property is used to **align the flex items along the main axis**. This defines the alignment along the main axis. This property **distributes extra free space** inside the layout between the elements.

The justify-content property takes on any of the values below:

- **flex-start** - default

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

Eg., for the layout like this:

```
<div class="flex-container">
  <div id="div1">div1</div>
  <div id="div2">div2</div>
  <div id="div3">div3</div>
</div>
```

we apply this css to it:

```
.flex-container { display:flex; width:350px; background-color:lightgrey; }
.flex-container div { width:70px; height:70px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; }
```

the layout will look like this:

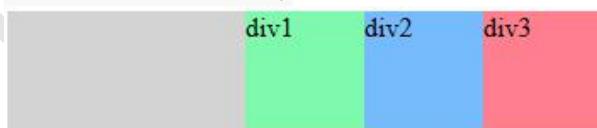


## 7.1. flex-start

The **flex-start** value aligns the items towards the **start of the main axis**. This is the **default value**. Else use `justify-content: flex-start;` in the flex-container class.

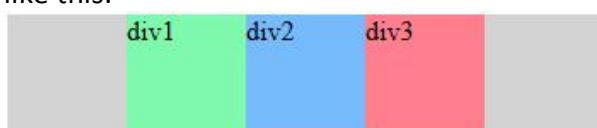
## 7.2. flex-end

The **flex-end** value aligns the items towards the **end of the main axis**. Use `justify-content: flex-end;` in the flex-container class and the flex items will look like this:



## 7.3. center

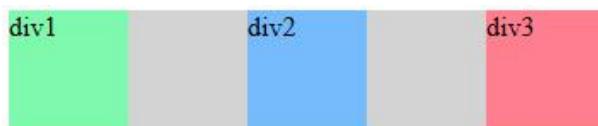
The **center** value aligns the items towards the **center of the main axis**. Use `justify-content: center;` in the flex-container class and the flex items will look like this:



## 7.4. space-between

The **space-between** value distributes the remaining **space between the items evenly along the main axis**. No space is provided towards the start of the first container and end of last container. So, there is same spacing between the items.

Use `justify-content: space-between;` in the flex-container class and the flex items will look like this:



## 7.5. space-around

The **space-around** value distributes the remaining **space around the items evenly along the main axis**.

Visually the spaces does not seem to be equal. This is because all the flex items have equal space on both sides.

Eg., suppose each item gets a spacing of 10px on both sides of each other. So, first item will have 10px on the left side. The first item will have 10px on the right side and second item will have 10px on it's left side. This will total to 20px spacing between them. Similarly, the last item have only 10px spacing on it's right side.

Use `justify-content: space-around;` in the flex-container class and the flex items will look like this:

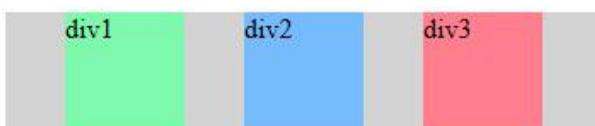


## 7.6. space-evenly

The **space-evenly** value distributes the remaining **space between the items and edges of the container evenly along the main axis**.

This provides a visual look of evenly spread items inside the container.

Use `justify-content: space-evenly;` in the flex-container class and the flex items will look like this:



## 8. FLEX - ALIGN ITEMS

The **align-items** property is used to align the **flex items along the cross-axis**. The cross-axis is perpendicular to the main-axis.

If the main-axis is horizontal, then cross-axis is vertical. If the main-axis is vertical, then cross-axis is horizontal.

Align-items can be set to any of these values:

- **flex-start**
- **flex-end**
- **center**
- **stretch** - default
- **baseline**

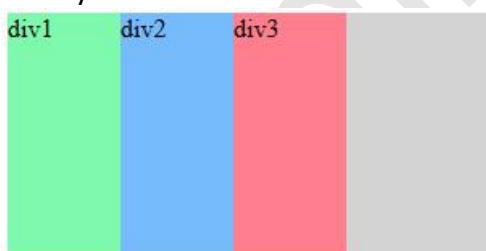
Eg., for the layout like this:

```
<div class="flex-container">
  <div id="div1">div1</div>
  <div id="div2">div2</div>
  <div id="div3">div3</div>
</div>
```

we can use like this:

```
.flex-container { display:flex; width:300px; height:150px; background-color:lightgrey; }
.flex-container div { width:70px; min-height:70px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; }
```

the layout will look like this:



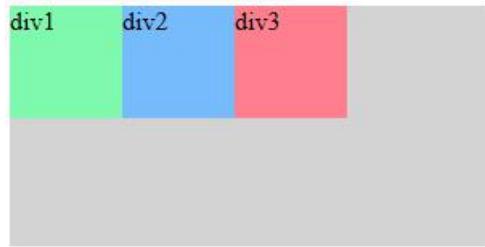
**NOTE:** Remember that the default behaviour will stretch the items only if the **height of the item is not fixed**. But if the size is fixed, all the other values other than the stretch value will work.

### 8.1. **stretch**

The **stretch** value stretches the flex items to **fill the container along the cross axis**. This is the default value. Else use `align-items: stretch;` in the flex-container class.

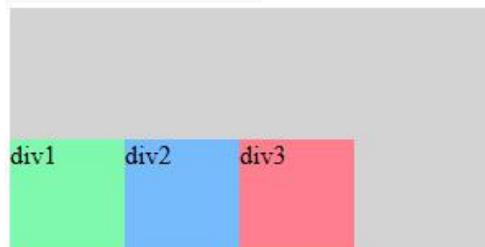
## 8.2. flex-start

The **flex-start** value aligns the items towards the **start of the cross axis**. Use `align-items: flex-start;` in the flex-container class and the flex items will look like this:



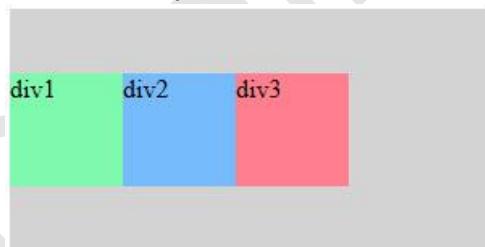
## 8.3. flex-end

The **flex-end** value aligns the items towards the **end of the cross axis**. Use `align-items: flex-end;` in the flex-container class and the flex items will look like this:



## 8.4. center

The **center** value aligns the items towards the **center of the cross axis**. Use `align-items: center;` in the flex-container class and the flex items will look like this:



## 8.5. baseline

The **baseline** value aligns the flex items such as the **baselines of the content inside the items**. This means that the bottom base of the content inside the flex item will be the axis to align the items.

To work with this property, we need to make some variation in the above code. The changes are highlighted:

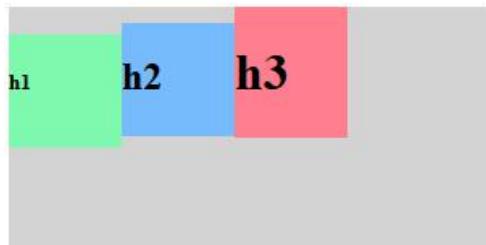
```
<div class="flex-container">
  <div id="div1"><h5>h1</h5></div>
  <div id="div2"><h2>h2</h2></div>
  <div id="div3"><h1>h3</h1></div>
```

```
</div>
```

we can use like this:

```
.flex-container { display:flex; width:300px; height:150px; align-items: baseline; background-color:lightgrey; }
.flex-container div { width:70px; min-height:70px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; }
```

the layout will look like this:



Here, see that the items are aligned based on the contents (i.e. h1, h2 and h3) baseline (or bottom of the text).

## 9. FLEX - ALIGN CONTENT

The **align-content** property is used to **align the flex lines**, i.e., multiple lines of flex items. This property aligns flex lines when there is extra space in the cross-axis, similar to how justify-content aligns individual items within the main-axis.

Align-content can be set to any of these values:

- **flex-start**
- **flex-end**
- **center**
- **space-between**
- **space-around**
- **stretch** - default

Eg., for the layout like this:

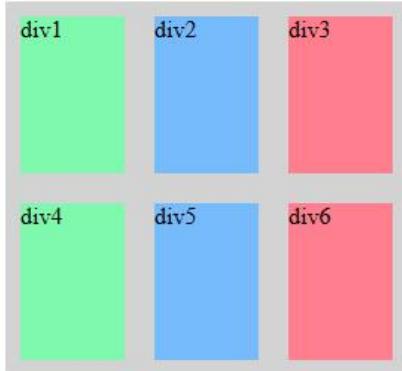
```
<div class="flex-container">
  <div class="div1">div1</div>
  <div class="div2">div2</div>
  <div class="div3">div3</div>
  <div class="div1">div4</div>
  <div class="div2">div5</div>
  <div class="div3">div6</div>
</div>
```

we apply this css to it:

```
.flex-container { display:flex; width:270px; height:250px; flex-wrap:wrap;
  background-color:lightgrey; }
.flex-container div { width:70px; min-height:70px; margin:10px; }
.div1 { background-color:#7ff9ae; }
```

```
.div2 { background-color:#76bbfc; }
.div3 { background-color:#ff7f8e; }
```

the layout will look like this:



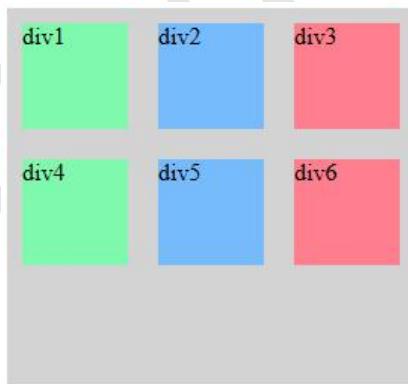
**NOTE:** This property works only when there are multiple lines of flex items. So, it has no effect when there is only one line of flex items.

### 9.1. stretch

The **stretch** value stretches the flex items present in multiple lines to **fill the container equally along the cross axis**. This is the default value. Else use `align-content: stretch;` in the flex-container class.

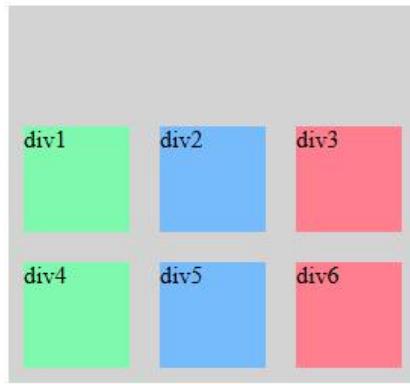
### 9.2. flex-start

The **flex-start** value aligns the multiple lines of flex items towards the **start of the cross axis**. Use `align-content: flex-start;` in the flex-container class and the flex items will look like this:



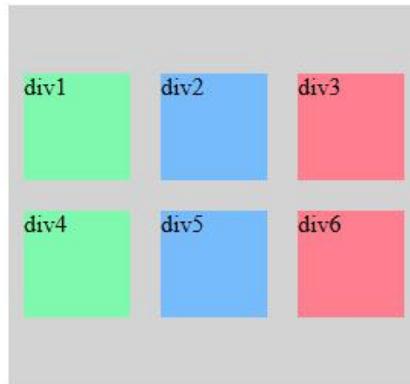
### 9.3. flex-end

The **flex-end** value aligns the multiple lines of flex items towards the **end of the cross axis**. Use `align-content: flex-end;` in the flex-container class and the flex items will look like this:



## 9.4. center

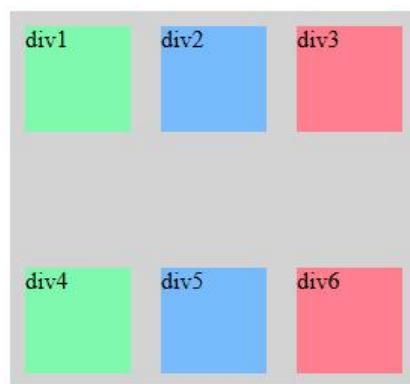
The **center** value aligns the multiple lines of flex items towards the **center of the cross axis**. Use `align-content: center;` in the flex-container class and the flex items will look like this:



## 9.5. space-between

The **space-between** value distributes the remaining **space between the multiple lines of flex items evenly along the cross axis**. No space is provided towards the start of the first container and end of last container. The space at top and bottom of container is because of the margin of the flex items.

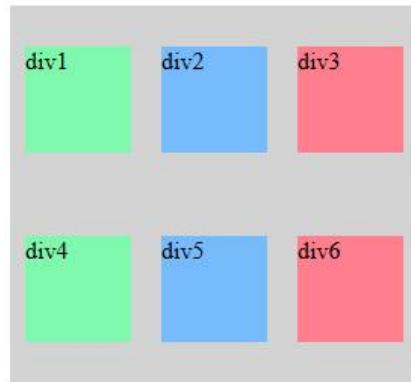
Use `align-content: space-between;` in the flex-container class and the flex items will look like this:



## 9.6. space-around

The **space-around** value distributes the remaining space around the multiple lines of flex items evenly along the main axis.

Use `align-content: space-around;` in the flex-container class and the flex items will look like this:



## 10. FLEX - ALIGN SELF

The **align-self** property is used for the **alignment of selected flex items** inside the flex container. This overrides the default alignment set by the flex container.

Align-self can be set to any of these values:

- **auto** - default
- **flex-start**
- **flex-end**
- **center**
- **stretch**
- **baseline**

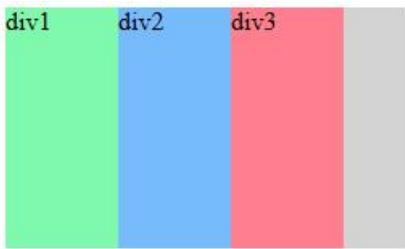
Eg., for this simple layout:

```
<div class="flex-container">
  <div id="div1">div1</div>
  <div id="div2">div2</div>
  <div id="div3">div3</div>
</div>
```

we apply this to it:

```
.flex-container { display:flex; width:250px; height:150px; background-color:lightgrey; }
.flex-container div { width:70px; min-height:70px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; }
```

and it will look like this in the browser:

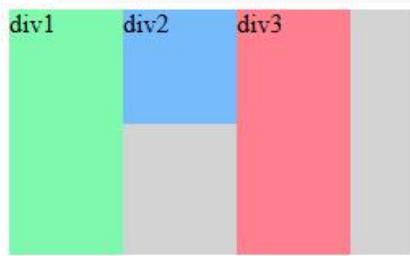


### 10.1. auto

The **auto** value inherits the flex container align-items property, or **by default stretch** is applied if align-items property is not set. This is the default value. Else use `align-self: auto;` in the flex-container class.

### 10.2. flex-start

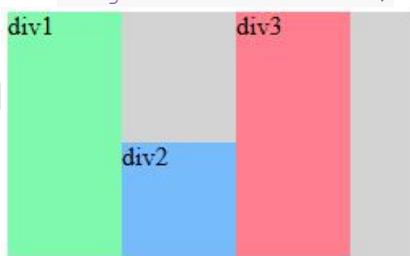
The **flex-start** value aligns the selected flex items towards the **start of the cross axis**. Use `align-self: flex-start;` with the selected flex items and it will look like this:



Here, the property is applied to div2.

### 10.3. flex-end

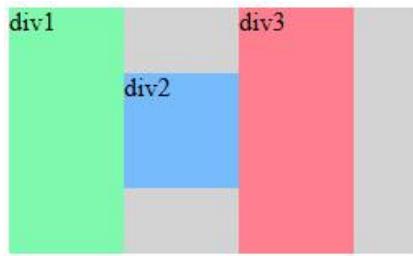
The **flex-end** value aligns the selected flex items towards the **end of the cross axis**. Use `align-self: flex-end;` with the selected flex items and it will look like this:



Here, the property is applied to div2.

### 10.4. center

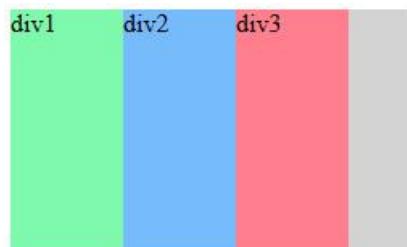
The **center** value aligns the selected flex items towards the **center of the cross axis**. Use `align-self: center;` with the selected flex items and it will look like this:



Here, the property is applied to div2.

## 10.5. stretch

The **stretch** value stretches the selected flex items to **fill the container along the cross axis**. Use `align-self: stretch;` with the selected flex items and it will look like this:



For this example, auto is also set to stretch by default. That's why both auto and stretch value are same.

## 10.6. baseline

The **baseline** value aligns the flex items such as the **baselines of the content inside the items**. This means that the bottom base of the content inside the flex item will be the axis to align the items.

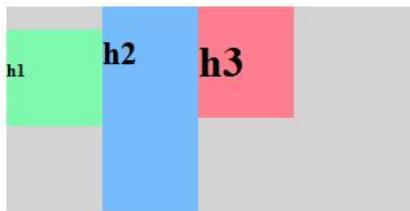
To see this property work, we need to make some variation in the above code. The changes are highlighted:

```
<div class="flex-container">
  <div id="div1"><h5>h1</h5></div>
  <div id="div2"><h2>h2</h2></div>
  <div id="div3"><h1>h3</h1></div>
</div>
```

we can use like this:

```
.flex-container { display:flex; width:300px; height:150px; background-color:lightgrey; }
.flex-container div { width:70px; min-height:70px; }
#div1 { background-color:#7ff9ae; align-self: baseline; }
#div2 { background-color:#76bbfc; }
#div3 { background-color:#ff7f8e; align-self: baseline; }
```

the layout will look like this:



Here, h2 is not given the baseline property. Therefore, it is not aligned and is stretched by default.

## 11. FLEX BASIS

The **flex-basis** is used to specify an **initial length to the flex-item**. The length of the item can be provided in absolute as well as in relative values. It will first apply the length of the item before applying flex-grow or flex-shrink to the items.

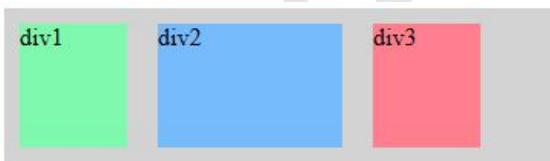
Eg., for a simple layout:

```
<div class="flex-container">
  <div id="div1">div1</div>
  <div id="div2">div2</div>
  <div id="div3">div3</div>
</div>
```

the css applied is:

```
.flex-container { display:flex; width:360px; height:100px; background-color:lightgrey; }
.flex-container div { width:70px; margin:10px; }
#div1 { background-color:#7ff9ae; }
#div2 { background-color:#76bbfc; flex-basis:120px; }
#div3 { background-color:#ff7f8e; }
```

this will look like this:



## 12. SHORTHAND PROPERTIES OF FLEX

Shorthand properties are used to **combine some flex properties together**, so that they can be defined in one line.

### 12.1. Flex Flow

The **flex-flow** property is a shorthand property to combine **flex-direction** and **flex-wrap** properties in one property.

CSS Syntax is - `flex-flow: flex-direction flex-wrap | initial | inherit;`

The meaning of the other 2 values is:

- **initial** - represents flex-flow property as (row nowrap)
- **inherit** - inherits this property from its parent element

## 12.2. Flex

The **flex** property is a shorthand property to combine the flex child properties, i.e. **flex-grow**, **flex-shrink** and **flex-basis**.

CSS Syntax is - `flex: flex-grow flex-shrink flex-basis | auto | initial | none | inherit;`

The meaning of the other 4 values is:

- **auto** - represents flex property as (1 1 auto)
- **initial** - represents flex property as (0 1 auto)
- **none** - represents flex property as (0 0 auto)
- **inherit** - inherits this property from its parent element