Flexbox

It is assumed that you have learned the basics of flexbox layout although we will be doing a quick review here.

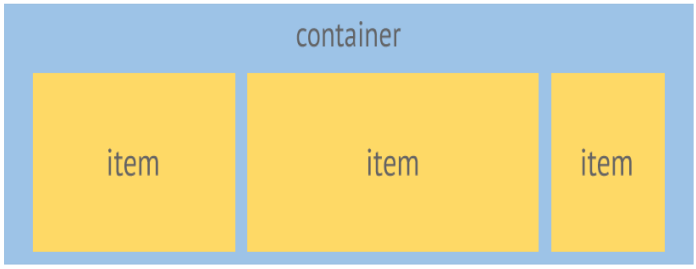
In November/2016 the flexbox was still in draft mode at W3C and the main idea of the proposal was to provide a more efficient way for layout design of web pages (web apps). The flex container expands the items to fill any available space.

Note: Most developers use flexboxes to components of an application and/or small-scale layouts while they prefer to use grid for larger-scale layouts.

Flexbox Terminology

The **container** – the parent also called **flex container**
The **items** – the children also called **flex items**

```
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
```



CSS Properties

It is also possible to change the direction of the ***flex line***. You can change that direction by setting the **direction** property to **rtl** (right-to-left) and the content will be drawn from right to left.

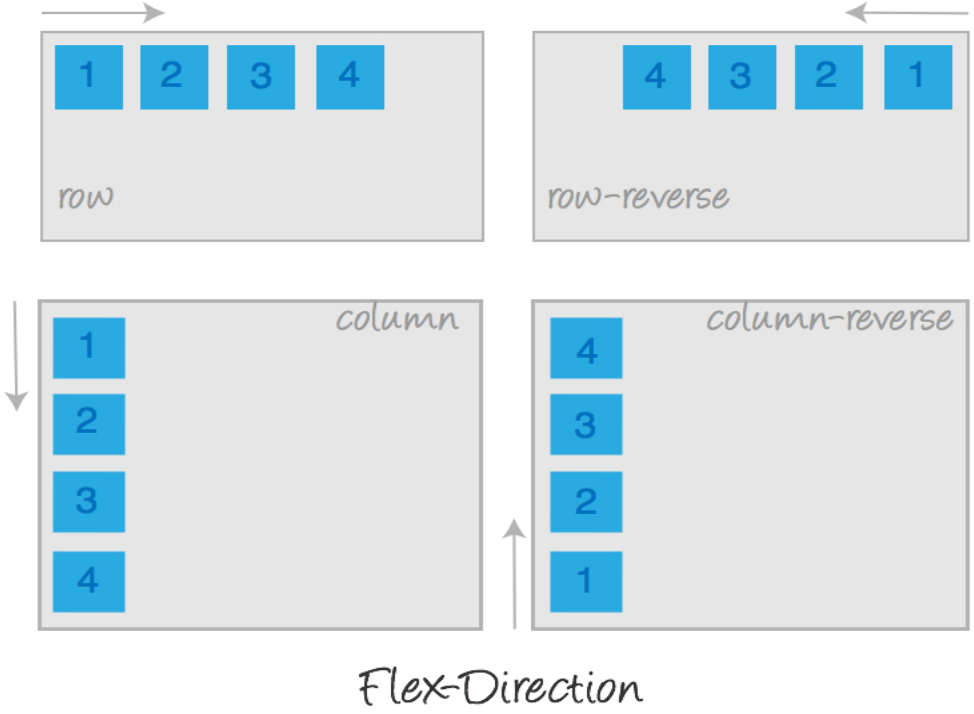
```
body {
        direction: rtl;
}
```

CSS Properties used for flex container

The parent will become a flexbox when you set the **display** property to be **flex**. You can also set the direction the flex items will be positioned within the container (parent) by using the **flex-direction** property that can have the following values:

- **row** - the default value of flex-direction (left-to-right, top-to-bottom)
- **row-reverse**
- **column**
- **column-reverse**

The **flex-direction** property gives the main axis defining the direction that flex items will be placed. The image below represents how the flex items would be displayed depending on the value you use for the **flex-direction** property – the image below assumes you have 4 elements, inside the flex container, that will be flex items:



Flex-Direction

An example of a code would be:

```
.flex-container {
    display: flex;
    flex-direction: row-reverse;
    width: 400px;
    height: 250px;
    background-color: #e3e3e3;
}
```

The **flex-container** class would then be applied to an element of your HTML document that would become the flex container and any children elements of that flex container would automatically become flex items presented in a row-reverse way.

The **flex-wrap** sets the way the flex items will wrap or not within the area of the flex container. The possible values are:

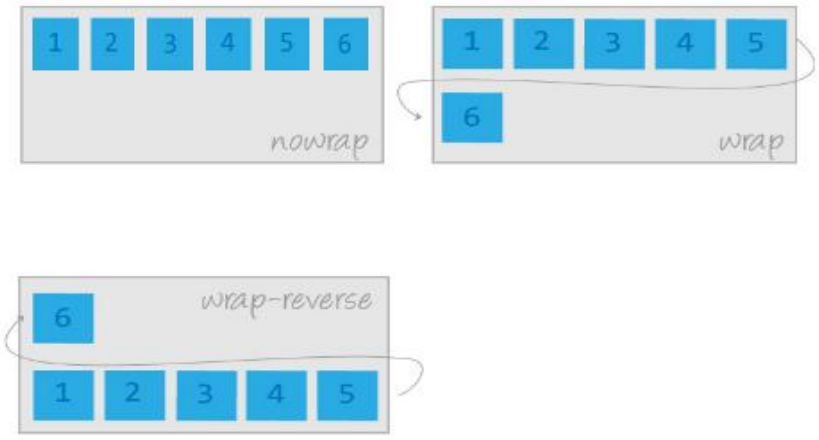
- **nowrap** - the default value
- **wrap**
- **wrap-reverse**

For example, you can have something as:

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

By default, the flex items will always try to fit within the same row no matter how many flex items you have but if you code like the example above, you would have the flex items wrap to another row within the flex container.

The image below shows the values for the **flex-wrap** property and what would be the effect in the flex items within the flex container.



Flex-Wrap

The **flex-flow** is a shorthand for **flex-direction** and **flex-wrap** and the default value for this property would be ***row nowrap***.

The **align-items** property will set how flex items will be laid out across the axis of the container. These are the possible values for this property:

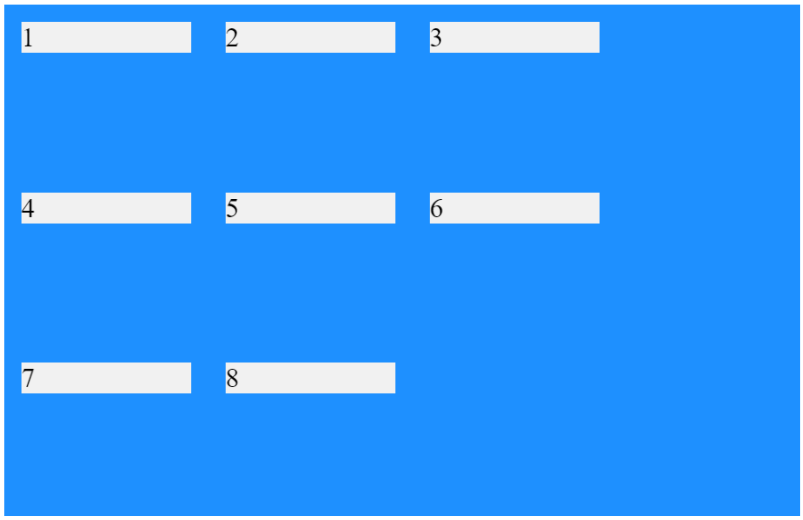
- **stretch** – default
- **flex-start**
- **flex-end**
- **center**
- **baseline**

The example below would have the flex items not wrapping (as there is no **flex-wrap**, neither **flex-flow**), the direction would be the default direction (there is no **flex-direction**, neither **flex-flow**).

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

Now imagine this example below and that each flex item is represented by a div that has the properties you see listed for **.flex-container > div**, the image you see on the right side represents how the 8 div's, inside the flex container would be displayed:

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  background-color: DodgerBlue;
  align-items: flex-start;
  height: 300px;
}

 .flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
}
```



Because the **align-items** has the value of **flex-start**, you see the flex items starting at the top left of the flex container and then continuing their alignment based on the combination of properties set to the **flex-container** class and to the div's (flex items) inside that flex container.

The **justify-content** defines the alignment of the items within the container and it can have the following values:

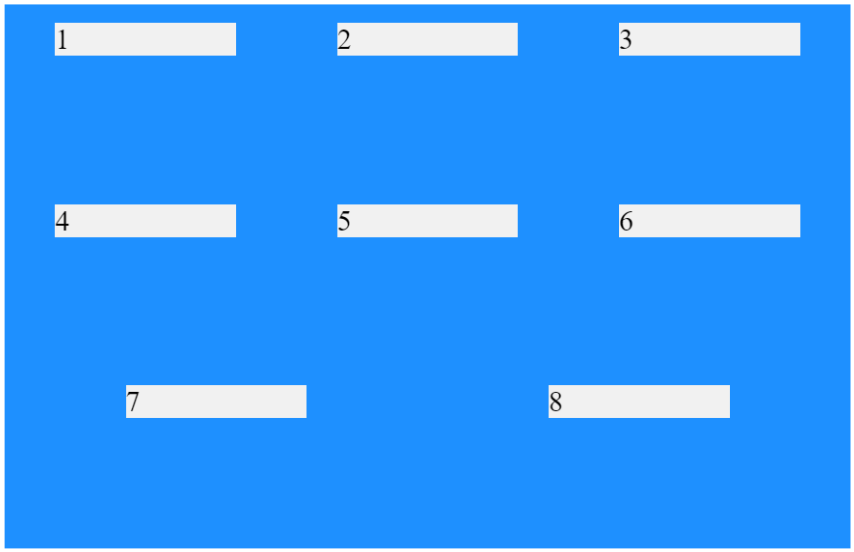
- **flex-start** - default value
- **flex-end**
- **center**
- **space-between**
- **space-around**

If we changed the previous example to be (noticed the modified/added code in red bold color):

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  background-color: DodgerBlue;
  align-items: flex-start;
  justify-content: space-around;
  height: 300px;
}
```

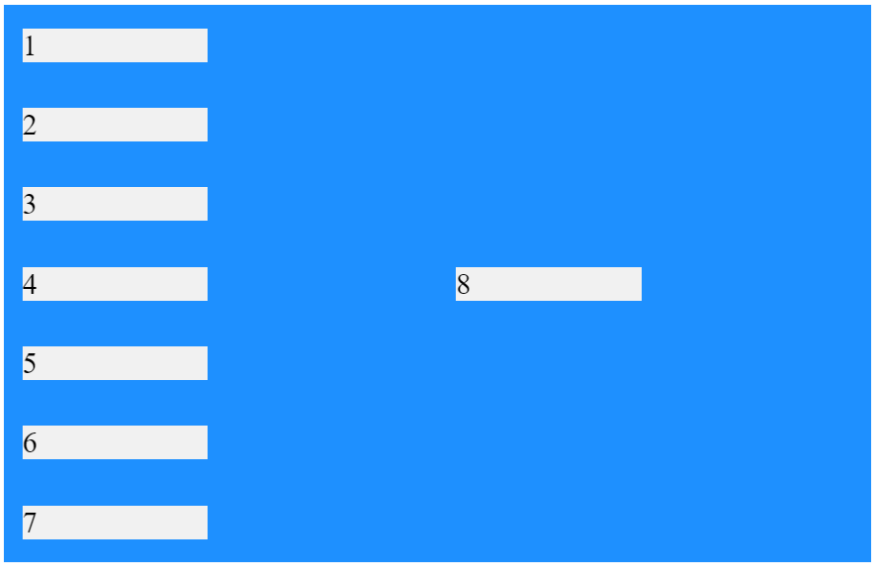
```
.flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
}
```

The image below would represent the new disposition of the flex items within the flex container – notice that the **justify-content: space-around** is applied along the horizontal axis while the **align-items: flex-start** is applied to the vertical axis (the flex items are starting at the top part of the flex-container).



Now, we will add a flex-direction: column to the previous code for the flex-container class (see below the added code in red bold color):

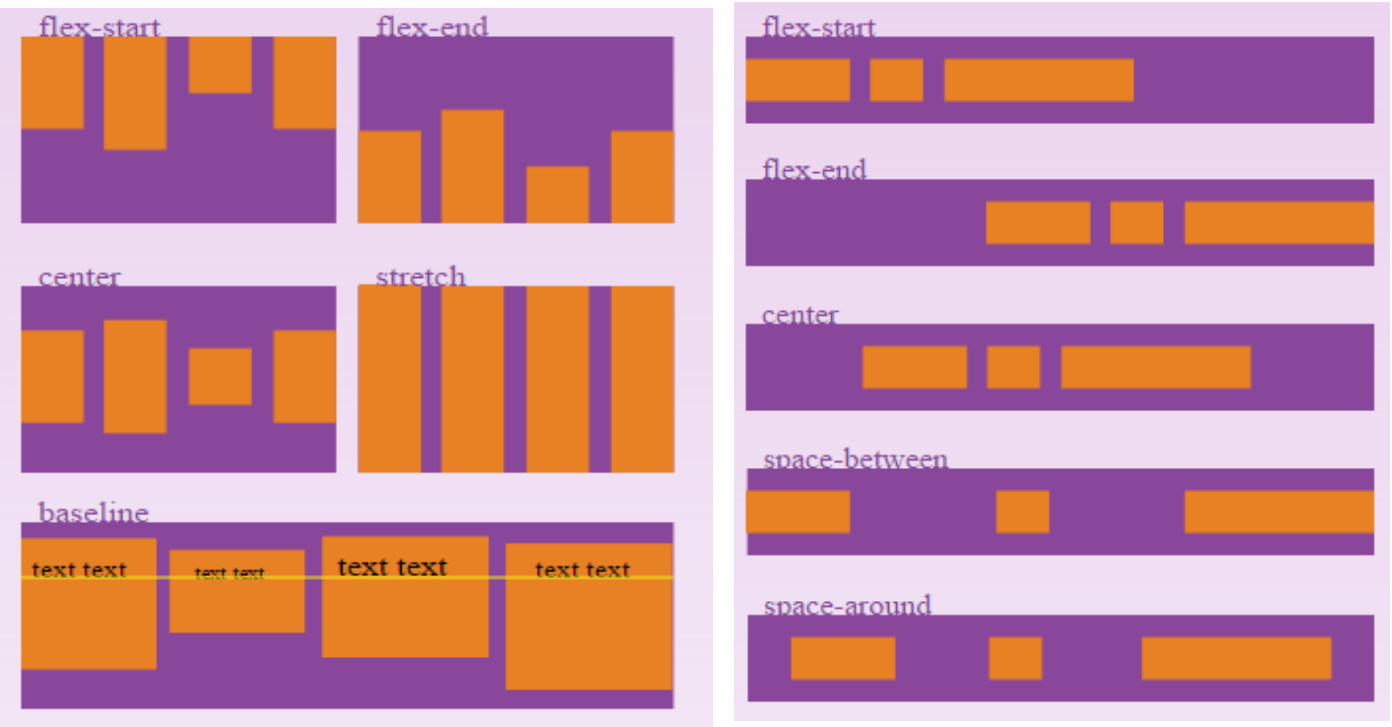
```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  flex-direction: column;
  background-color: DodgerBlue;
  align-items: flex-start;
  justify-content: space-around;
  height: 300px;
}

.flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
}
```



Notice that now, the image on the right shows the disposition of the 8 flex items (div's) and now the **align-items: flex-start** is being applied to the horizontal axis (the flex items are starting on the left side of the flex

container) while the **justify-content: space-around** is being applied to the vertical axis (that is the reason you see the 8th div in the second/right column, vertically positioned in the middle of it).

The image below shows the difference between **align-items** and **justify-content** properties (on the left side you see the different disposition of flex items for the different values of **align-items** – **flex-start**, **flex-end**, **center**, **stretch**, **baseline**; on the right side of the image you see the different disposition of the flex items for the different values of **justify-content** – **flex-start**, **flex-end**, **center**, **space-between**, **space-around**):



The **align-content** defines how it will align the lines in the flex container and it can have the following values:

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

It is similar to the align-items property, but instead of aligning flex items, it aligns flex lines!

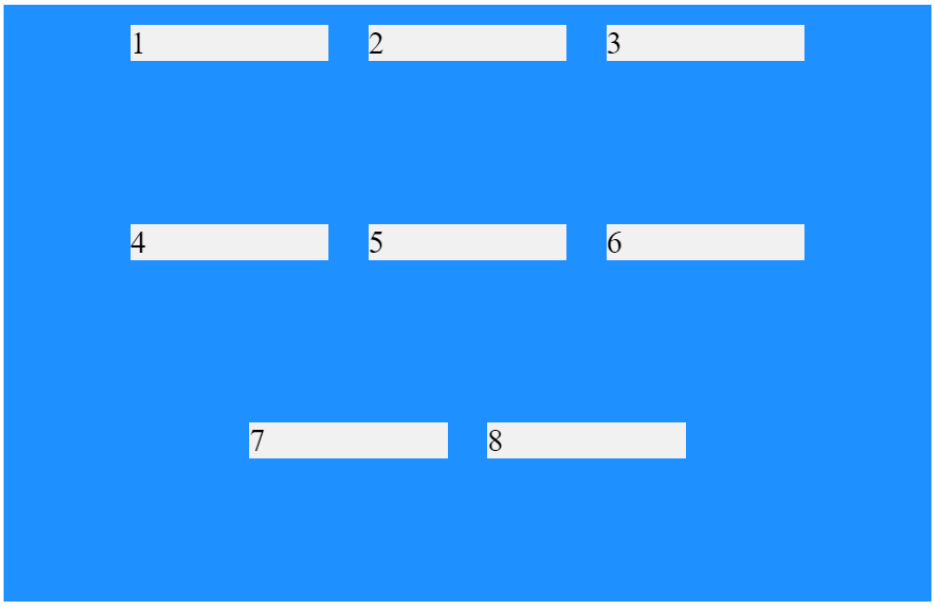
NOTE: This property will have no effect when there is only one line of flex items being displayed

Using the same example as before with some changes in the code as shown below:

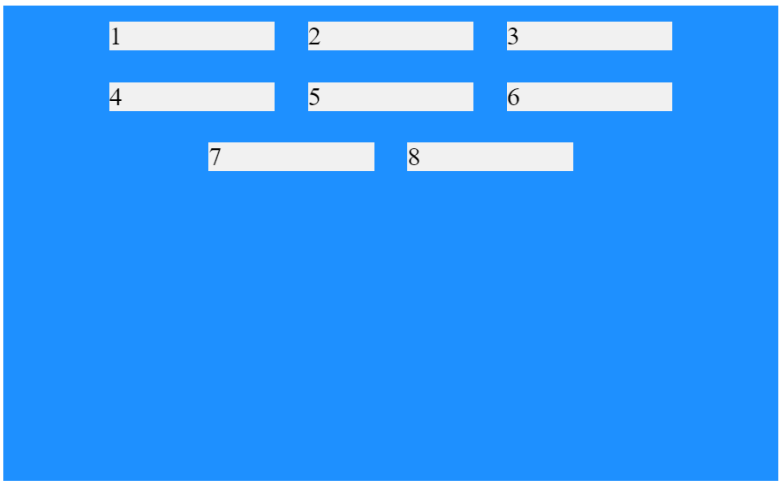
```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  background-color: DodgerBlue;
  align-items: flex-start;
  justify-content: center;
  height: 300px;
```

```
}

 .flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
}
```

Now we have no **flex-direction** which means that it is the default value of **row** and this means that **align-items** will be positioned considering the vertical axis (the value of **flex-start** makes the flex items start at the top) and the **justify-content** will be positioned considering the horizontal axis (the value of **center** makes the flex items be centralized within the horizontal axis of the flex container). This is what you see in the image below:



If I add to the previous code, in the **.flex-container** class, the line with **align-content: flex-start;** the image would change to what you see here on the right side: the flex lines are distributed from the top without that much space, between the first and second lines and the second and third lines, you saw in the previous image.



Differences between align-items, justify-content, and align-content

The **justify-content** aligns the flex items **on the main axis** (by default, if not **flex-direction** is set, it will be the horizontal axis = **row**). If you change the **flex-direction** to be **column**, then the main axis will be the column (the vertical axis).

The **align-items** has the same meaning of **justify-content** but now the alignment of the flex items will happen **on the cross axis** and if you change the **flex-direction** to be **column**, the cross axis will be the **row** (horizontal axis).

The **align-content** defines how the lines in the flex container will be aligned and this one will only be noticed if you have more than one row (or more than one column) of flex items.

NOTE: if you have only one row of flex items, the **align-items: center** will seem like it is doing nothing especially if you do not have any **height** applied to the flex container (the height of the flex container becomes basically the height of the row of flex items). If you apply a different **height** to the flex container, then you will see the **align-items: center** working as expected – of course, this scenario is for the case you do not change the **flex-direction** to be column.

CSS Properties used for flex items

The **order** property sets the order the flex item will be shown. The default value is 0 (zero). The value should always be an integer.

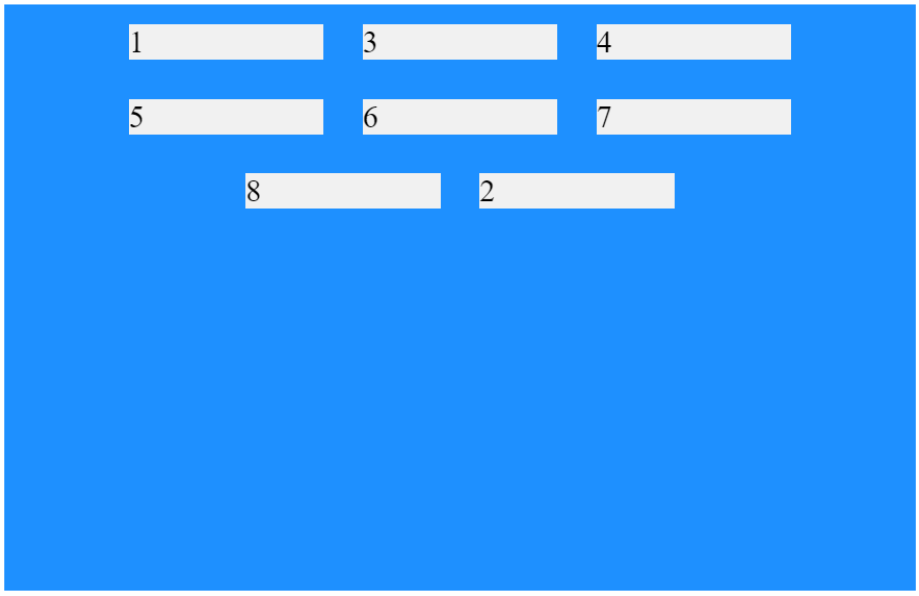
IMPORTANT: If you are using a **reverse** value (for **flex-flow**, or **flex-direction**) or **order** to most or all flex items, you should consider whether you need to change the logical order of the content in the source code as the specification and advice from W3C continues with a warning not to use reordering to fix issues in your source: ***"Authors must not use order or the *-reverse values of flex-flow/flex-direction as a substitute for correct source ordering, as that can ruin the accessibility of the document"***

Let us now get the previous code and add a class called **special** as shown below (the class **special** is shown in the code below with a red bold color):

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  background-color: DodgerBlue;
  align-items: flex-start;
  justify-content: center;
  align-content: flex-start;
  height: 300px;
}

 .flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
}
```

```
.special {
order: 3;
}
```

We will apply this class **special** to the div with the number **2** in it and that is what we will have – the div with the number **2** will be the last one, even after number **8**, because all the other div's have the default value of **order** which is 0 (zero) – this is what is shown in the image below:



If I apply the **special** class to the div with number **8**, then the div with number **2** will be shown before the number **8** and all the others will come before those two as they will continue with the default value of 0 (zero) to the **order** property.

The **flex** property specifies the length of the item relatively to the rest of the flex items inside the same flex container. This **flex** property is a shorthand for the **flex-grow**, **flex-shrink**, **flex-basis** and the default value of the **flex** property is **0 1 auto**.

The **flex-growth** property has the default value of 0 (zero) and specifies how much the item will grow relatively to the rest of the flex items inside the same flex container. The value is always a number.

The **flex-shrink** property sets the ability for the flex item to shrink, if necessary, also relatively to the rest of the flex items inside the same flex container. The default value is 1.

The **flex-basis** property specifies the initial length of a flexible item and the default is auto.

The flex-grow tells the element if or not it can take up additional space within the flex container. So, when we set a flex-grow value of 1 or bigger, we are telling our element to grow to take up the available space within the flex container and the default value of 0 (zero) will tell the element not to grow to take up available space.

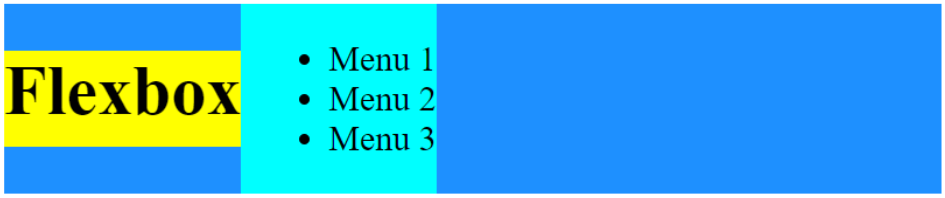
Imagine we have two elements next to one another and one element has a flex-grow of 2 and the other has a flex-grow of 1. This means that the total number of flex-grow units will be 3. Then, the available space in the

flex container will be divided by 3 and distributed accordingly meaning that the first item would get 2 pieces of the available space while the second item would get 1 piece of the available space. This DOES NOT mean that the first item will be twice as big as the second – remember, it is receiving 2/3 of the available space within the flex container and the second item is receiving 1/3 of that available space.

Suppose we have a header element with two children: h1 and nav. We will set the display:flex to the header. Here is what we would have as HTML:

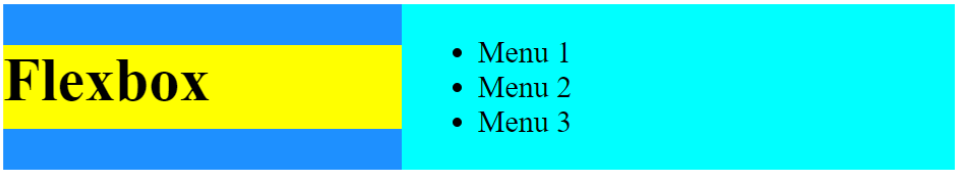
```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Flexbox</title>
<meta charset="utf-8">
<meta name="description" content="playing around with flex-grow, flex-shrink, flex-basis">
<style>
header {
  display: flex;
  background-color: DodgerBlue;
}

h1 { background-color: yellow; }

nav { background-color: cyan; }
</style>
</head>
<body>
<header>
  <h1>Flexbox</h1>
  <nav>
    <ul>
      <li>Menu 1</li>
      <li>Menu 2</li>
      <li>Menu 3</li>
    </ul>
  </nav>
</header>
</body>
</html>
```

This would produce the image you see here below:



Now, if we add **flex-grow: 1;** to the **h1**, we will get the following display in the browser:

The **h1** will grow to take up all the available space. Now, if we go back to the code and add **flex-grow:2;** to the **nav** element, the total **flex-grow** value will be 3 and the **nav** will be taking 2/3 of the available space while the **h1** will take 1/3 of that available space and here is what you would have:



The **flex-shrink** works in a similar way to **flex-grow** but instead of dealing with extra space, it deals with space not needed by an element's content.

An element with a **flex-shrink** value of 0 (zero) will not shrink if the flex container gets smaller but this is only true if there is no **flex-grow** value applied to this element because if there is a **flex-grow** value applied, the element will not shrink to be smaller than its content.

If you have more than one element per line, the total number of **flex-shrink** values will be considered when dividing up how much each element should shrink by.

The **flex-basis** is best used when together with either **flex-grow** or **flex-shrink**. Here are some cases using the **flex** property which is shorthand for those 3 properties: **flex-grow**, **flex-shrink**, and **flex-basis**.

- ***flex: 1 0 200px;*** - an element with this setting means it has **flex-basis** of 200px, **flex-grow** of 1, and **flex-shrink** of 0 (zero) which means that this element will be at minimum 200px wide but it will be allowed to grow (if there is space available for it). Then, you can think of the **flex-basis** as being the **min-width**.
- ***flex: 0 1 200px;*** - Now the **flex-basis** is 200px, **flex-grow** is set to 0 (zero) and **flex-shrink** is set to 1. In this case, this element will be 200px at its largest but can be smaller and, in this case, **flex-basis** is acting like a **max-width**.
- ***flex: 0 0 200px;*** - In this scenario, both **flex-shrink** and **flex-grow** are set to be 0 (zero) and the **flex-basis** is set to be 200px which means that this element will only ever be that size in pixel value. It will not be allowed to shrink, neither to grow. This would be the same of setting a pixel value to the CSS **width** property.
- ***flex: 1 0 auto;*** - The **auto** keyword let us define an element to be the size of its content. This is useful when we do not want to give the element a minimum size, nor a maximum size.
- ***flex: 0 1 auto;*** - This is the default value for the **flex** property.

The **align-self** allows the default alignment, or the one specified by **align-items** to be overridden. It can have one of the following values:

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

The **align-self** property specifies the alignment for the selected item inside the flexible container.

The **margin** property can be used to push flex items into different positions.

ATTENTION!!!

The CSS properties float, clear, and vertical-align have NO effect on a flex item! You can still use those properties inside the flex item (for the content of the flex item).

Look at **flexbox_1.html** and compare the code with **flexbox_2.html** and see how the flexbox layout was from applied. What would you modify in the code of **flexbox_2.html** to:
Show the paragraphs in columns?
Change the order of the flex items?
Make flex item 3 be wider than the other flex items?

Extra Resources

A complete guide to CSS Flexbox (from CSS Tricks)
CSS Flexbox in a practical use case of web accessibility