Theory: Flexibility, growth and contraction ratio

48 minutes 0 / 5 problems solved

Skip this topic Start practicing

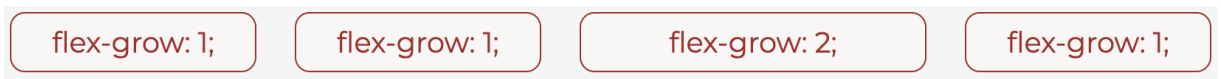
In real life we often need to make a flexible item that scales to a certain width with its content. If we had to define the width of the flex item manually, flexbox would not be such a holy grail that it is. It's possible to manipulate the flex items' width and flexibility just by setting their `flex-grow` or `flex-shrink` properties.

§1. Flex-grow

`Flex-grow` property defines the width of one item in relation to other elements.

It works differently than how you may imagine it. With this property, you don't define how much of the flex-container's width the item takes. Instead, you define how much of the **remaining free space** the item takes.

If the size of the parent container is more than the width of all the items combined, the **remaining space** will be distributed among them in proportion to their `flex-grow` factor. For example, if for all the elements the property is `flex-grow: 1;`, they will all have the same size because each will take the same part of the free space. However, if one of them has **2** as a `flex-grow` value, it will take twice as much free space as the others.



To make this even simpler, let's take a look at the example below. Here you can find the two code snippets with the picture that shows the output:

```
<div class="flex-container">
  <div class="item1">item1</div>
  <div class="item2">item2</div>
  <div class="item3">item3</div>
</div>
```

Current topic:

Flexibility, growth and contraction ratio — Stage 2

Topic depends on:

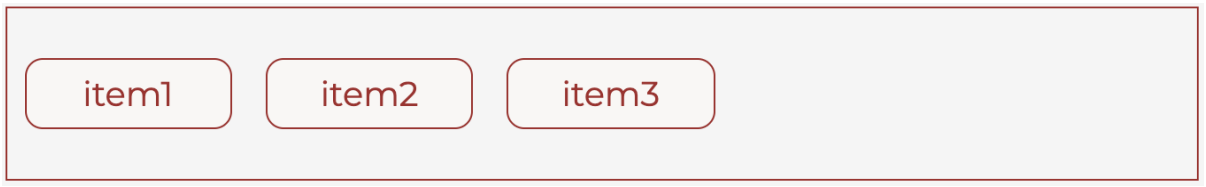
✕ Orientation and display order — Stage 2

Topic is required for:

Axis alignment — Stage 2

Table of contents:

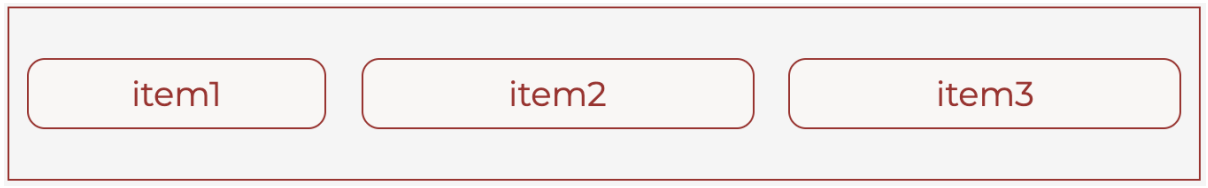
```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;



  flex-basis: 4em;

  flex-grow: initial;

}
```



There is some free space available and the elements are located closer to the left border. As you can see, all the items are of the same size because the `flex-grow` grow property value is 0 by default. Not applying the property at all will provide the same result.

Now let's set some values to this property:

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;


  flex-grow: initial;

}


.item1{

  flex-grow: 1;

}


.item2{

  flex-grow: 2;

}


.item3{

  flex-grow: 2;

}
```



This time the free space is distributed among the items so they fill the container almost evenly. The first item is smaller than the others because its `flex-grow` value is less than that of the others.

The remaining space is divided by the sum of the values, which in this case is 5. The result of the division is the size of **one part**. The other two items will get **two parts** because their flex-grow property value is 2. The first item will get the remaining one part.

§2. Flex-shrink

If the width of all the items combined is more than the size of their parent, you can either apply the `flex-wrap` property to make overflow items move to the next line or use `flex-shrink`, if it is possible to sacrifice some width. The

only difference between `flex-shrink` and `flex-wrap` is that with flex-shrink you have to operate with negative values. That is because the amount of free space is calculated by the following formula: **width of the container minus the width of all the items**. And since the first term is smaller than the second, we get the negative value.

As for the flex-grow, the 'free' space will be distributed between flex items in proportion to their `flex-shrink` values. That means if the flex-shrink values of the items are 1, 2, and 3, respectively, the first item will get 1/6 of the free space, the second will get 2/6, and the last one will get 3/6. But since the remaining space is negative, the width of each container will be **reduced** by those numbers.

This property has a default value of 1, which makes items shrink even if it is not applied. However, if it is applied, it's possible to manage how much each item shrinks individually.

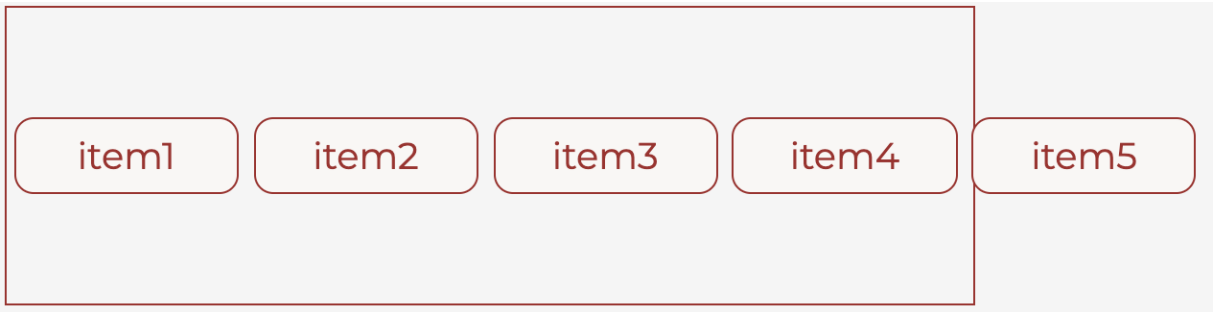
> The bigger the value of the flex-shrink property, the bigger the reduction of the width.

Let's take a look at the example below:

```
<div class="flex-container">
  <div class="flex-item item1">item1</div>
  <div class="flex-item item2">item2</div>
  <div class="flex-item item3">item3</div>
  <div class="flex-item item4">item4</div>
  <div class="flex-item item5">item5</div>
</div>
```

```


.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}



.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;



  min-width: 5em;

}
```

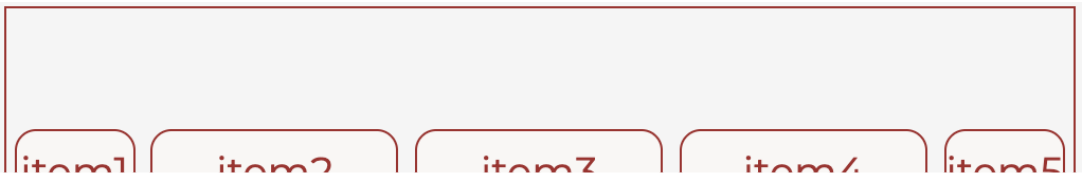
Here all the items have `min-width` property applied, which makes them overflow even though they're not yet compressed to their minimal size:



Let's add the following style changes:

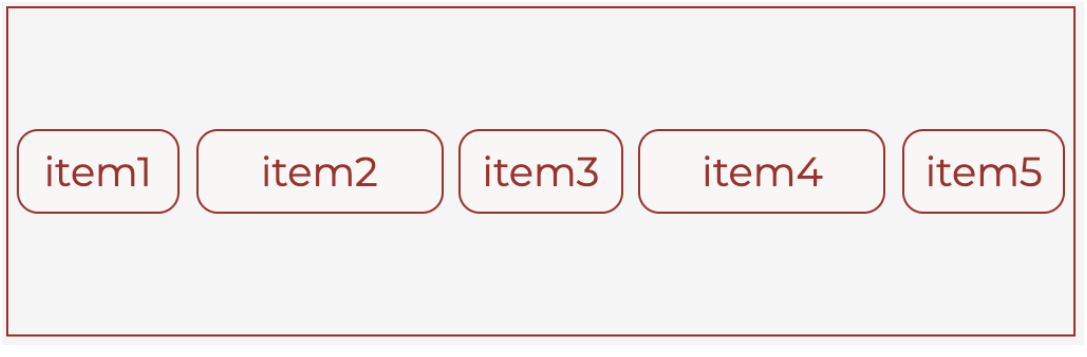
```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;


  min-width: 5em;

}


.item1,

.item5{

  flex-shrink: 1;

}


.item2,

.item3,

.item4{

  flex-shrink: 0;

}
```

In this case items in the middle won't shrink at all since they will get 0 parts of the 'free space', but both the first and the last items will shrink just enough for everyone to fit in the container:



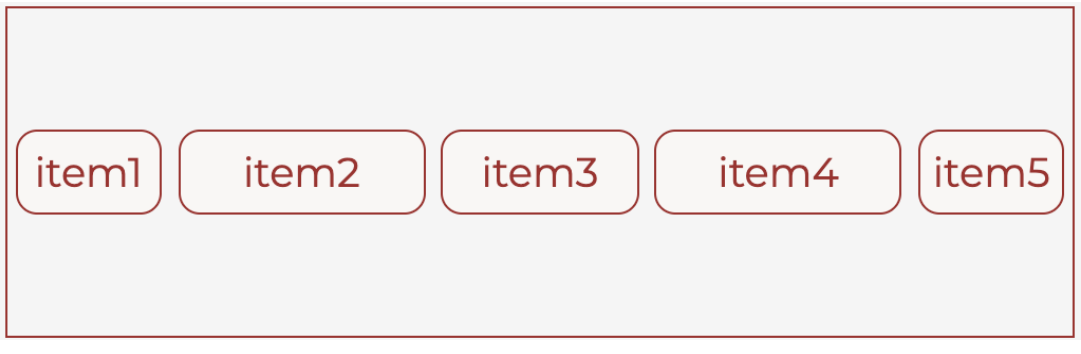
Now let's make the third item shrink too by setting its `flex-shrink` value as 1. As a result, its size will be equal to the size of the items closer to the borders:

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;


  min-width: 5em;

}


.item1,

.item5{

  flex-shrink: 1;

}


.item2,

.item4{

  flex-shrink: 0;

}


.item3{

  flex-shrink: 1;

}
```

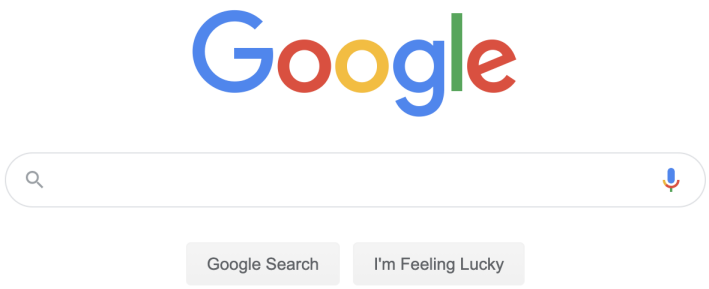


If we want to make the middle item bigger than the items closer to the border, we'll have to raise the border items' `flex-shrink` value. Let's set it at 2. In this case, the border items will get 2 parts of the remaining space and the middle item will get only one part:

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;


  min-width: 5em;

  flex-grow: initial;

}


.item1,

.item5{

  flex-shrink: 2;

}


.item2,

.item4{

  flex-shrink: 0;

}


.item3{

  flex-shrink: 1;

}
```



At first sight, it doesn't seem like both of the properties are used very often or used at all, but, actually, there are a lot of examples of their usage! Even the main page of Google:



If those two buttons were items of the flex container (maybe they are), they would probably look weird without `flex-grow`:

The contents of any item may change, as well as the number of the items, or anything else, which will lead to overflow or, the way around, to items filling the container without overflow. And that implies the change of the CSS properties or even of the whole structure. With `flex-grow` or `flex-shrink`, you don't have to worry about manually changing the width of the items.

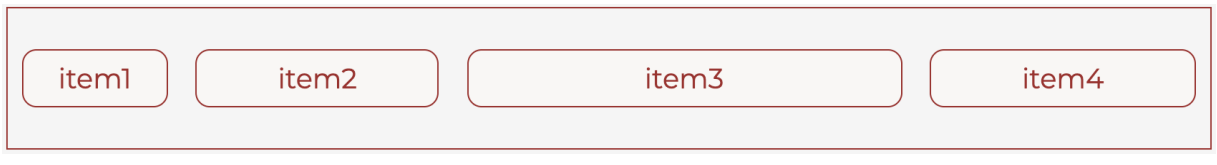
§3. Flex-basis

`Flex-basis` is a simple property which defines the size of the item and behaves like `min-width` (or `min-height` in case of the `flex-direction: column;`) with one exception: `min-width` doesn't allow the width to be reduced automatically for the items to fit the container, unlike `flex-basis`. If the width has to be changed according to `flex-grow` or `flex-shrink` or it's not enough to fit all the elements inside the container, the property will be ignored.

Here's an example:

```
<div class="flex-container">
  <div class="item1">item1</div>
  <div class="item2">item2</div>
  <div class="item3">item3</div>
  <div class="item4">item4</div>
</div>
```

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;


  min-width: 1em;

}


.item1{

  flex-basis: 6vw;

}


.item2{

  flex-basis: 10vw;

}


.item3{

  flex-basis: 18vw;

}


.item4{

  flex-basis: 11vw;

}
```



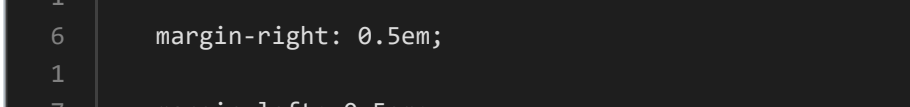
§4. Flex

There also is a shorthand property `flex` which combines all the properties listed above. You should use `flex-grow` as the first value, `flex-shrink` as the second value and eventually `flex-basis` as the third.

If none of the properties are set or only `flex: initial;` is, the default values will be applied. For `flex-grow` it's **0**, for `flex-shrink` it's **1** and for `flex-basis` it's **auto**, which is the size of the item's content. You can also use **none** as the `flex` property value and this will be the equivalent to `flex: 0 0 auto;`.

And as usual, let's take a look at the example. Here all the items fit the container and there is some space available:

```
<div class="flex-container">
  <div class="flex-item">item1</div>
  <div class="flex-item">item2</div>
  <div class="flex-item">item3</div>
  <div class="flex-item">item4</div>
</div>
```



Let's apply `flex-grow` property now:

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;



  flex-grow: 1;

  flex-basis:10vw;

}
```

And this is what we get as a result:

But the content as well as the number of the items might change, which will most likely cause overflow. That's when the `flex-shrink` option comes in handy:

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;



  flex-grow: 1;

  flex-shrink: 1;

  flex-basis:10vw;

}
```

This code, of course, can be shortened by using `flex` property instead of all the three used above:

```
.flex-container{
  border: 1px solid brown;
  color: brown;
  display: flex;
  font-family: 'Montserrat', sans-serif;
  padding: 0.5em 0 0.5em 0;
  width: 30em;
}


.flex-item{

  border: 1px solid brown;

  border-radius: 10px;

  text-align: center;

  height:2em;

  line-height: 2em;

  margin-right: 0.5em;

  margin-left: 0.5em;



  flex: 1 1 10vw;

}
```

In any possible case all the items will have the same size, fill the container, and stay within its borders.

Summary

`Flex-grow` and `flex-shrink` are the properties which are responsible for the width of the items. The size of an item by default is defined by either the width of its content or by the `flex-basis` value. If there is remaining space, it will be divided between the items in proportion to their flex-grow or flex-shrink values.

It turns out that these properties are not easy to use especially when it comes to math and negative values. But the key word here is **proportion** and if you bear it in mind, you'll successfully use these properties every time.

Report a typo

3 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

This content was created 2 months ago and updated 10 days ago. Share your feedback below in comments to help us improve it!

Comments (0) Hints (0) Useful links (0) Show discussion