A closer look at Absolute and Relative flex-items

This is perhaps the shortest article in this interactive course. However, it is important. Remember the goal of this course is to truly *"understand"* the inner workings of the Flexbox model.

Having covered some ground in previous sections, it's important to clarify a few important concepts here too.

What really is the difference between an absolute and relative flex-item?

The major difference between these two is got to do with spacing and how they are computed.

The spacing within a relative flex item is computed based on it's content size. In an absolute flex item, it is based solely on "flex", not content.

> **Example** isn't **another** way to teach, it is the **only** way to teach.
>
> -Albert Einstein

Here's a good example.

Consider the markup below.

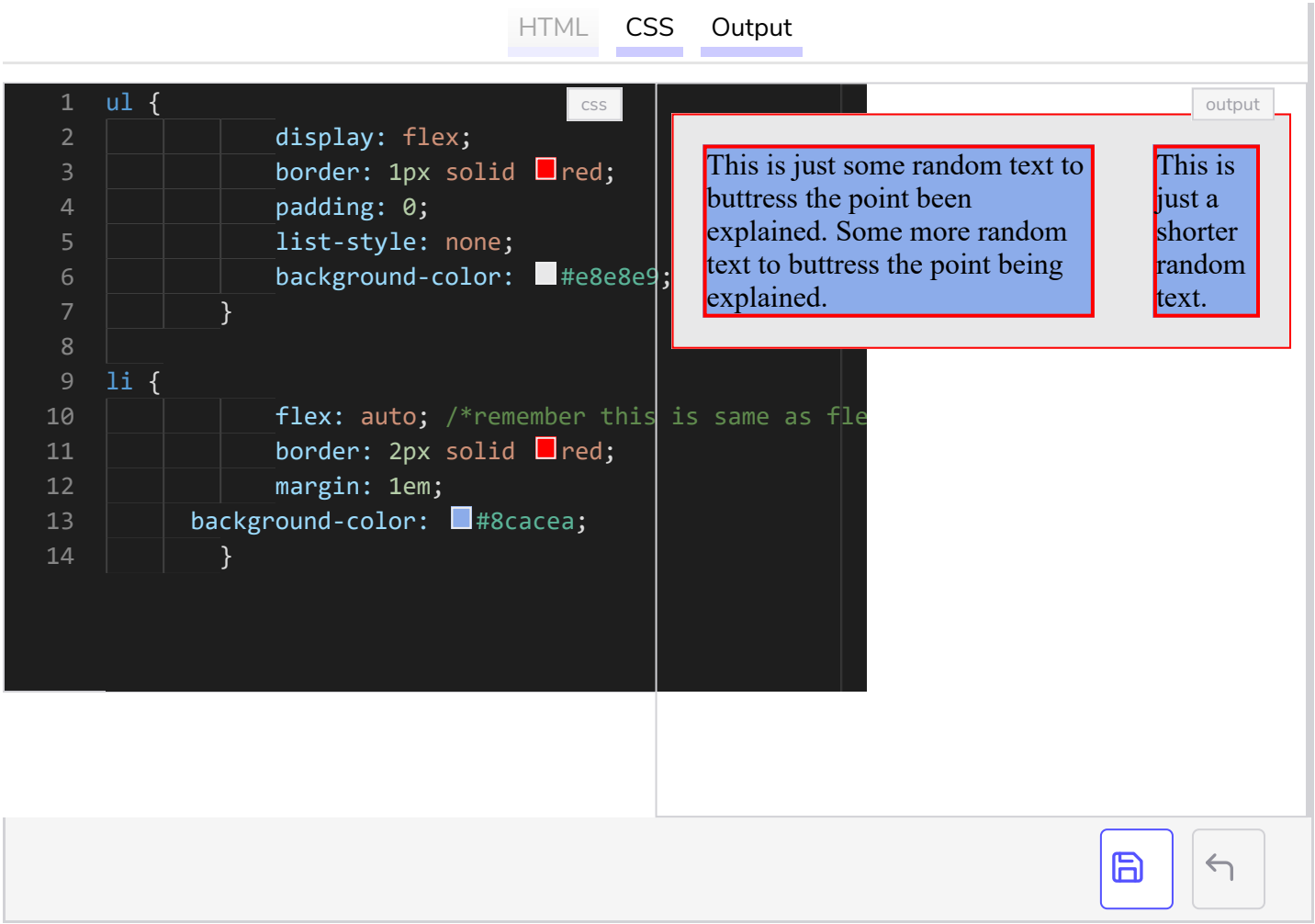
```
<ul>
        <li>
                This is just some random text  to buttress the point been explained.
        Some more random text to buttress the point being explained.
        </li>

        <li>This is just a shorter random text.</li>
</ul>
```

```
ul {
        display: flex; /*flexbox activated*/
}


li {
        flex: auto; /*remember this is same as flex: 1 1 auto;*/
        border: 2px solid red;
        margin: 2em;
}
```

Here's the result:



If you already forgot, `flex: 1 1 auto` is the same as setting: `flex-grow: 1 flex-shrink: 1` and `flex-basis: auto`

Using the framework I established earlier, the initial widths of the flex-items are automatically computed `flex-basis: auto`, and then they "grow" to fit the available space `flex-grow: 1`.

When flex-items have their widths computed automatically, `flex-basis: auto`, it is based on the size of the content contained within the flex-items.

The flex-items in the example above do NOT have contents of the same size. Hence, the sizes of the flex-items would be unequal.

Since the individual widths weren't equal in the first place (it was based off content), when the items grow, the widths also stay unequal.

HTML CSS Output

```
ul {
        display: flex;
        border: 1px solid red;
        padding: 0;
        list-style: none;
        background-color: #e8e8e9;
    }

li {
        flex: auto; /*remember this is same as fle
        border: 2px solid red;
        margin: 1em;
    background-color: #8cacea;
    }
```

output

This is just some random text to buttress the point been explained. Some more random text to buttress the point being explained.

This is just a shorter random text.

The flex-items in the example above are relative flex-items.

Let's make the flex-items absolute—meaning this time their widths should be based on "flex" NOT content size.

A *"one-liner"* does the magic.

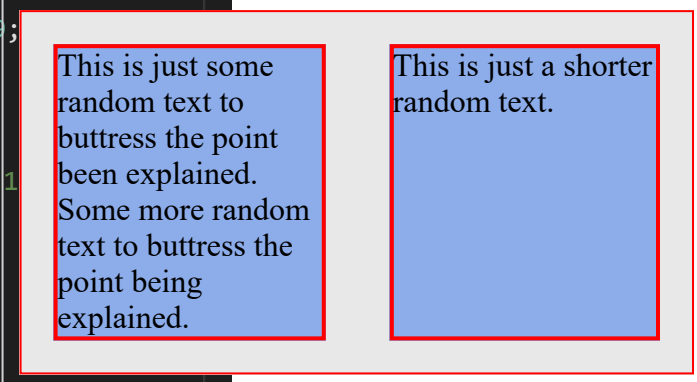
```
li {
        flex: 1 ; /*same as flex: 1 1 0. see flex: 'positive number'*/
}
```

Here's the result:

HTML CSS Output

```
ul {
```

```
2            display: flex;
3            border: 1px solid red;
4            padding: 0;
5            list-style: none;
6            background-color: #e8e8e9;
7        }
8
9    li {
10           flex: 1; /*same as flex: 1 1
11           border: 2px solid red;
12           margin: 1em;
13       background-color: #8cacea;
14       }
```



Do you see both flex-items have the same widths this time?

The initial widths of the flex-items is zero `flex-basis: 0`, and then they *"grow"* to fit the available space.

When there are two or more flex-items with zero based `flex-basis` values, they share the spacing available based on the `flex-grow` values.

I talked about this earlier.

Now the widths aren't computed based on content size. The widths are based on the flex value specified.

So you got that. Right?

Absolute flex-items have their widths based solely on flex, while relative flex items have their widths based on content size.