Gazza Azhari
103 Followers · About Follow

Playing around with Flexbox (CSS)

Gazza Azhari Jan 10, 2019 · 4 min read

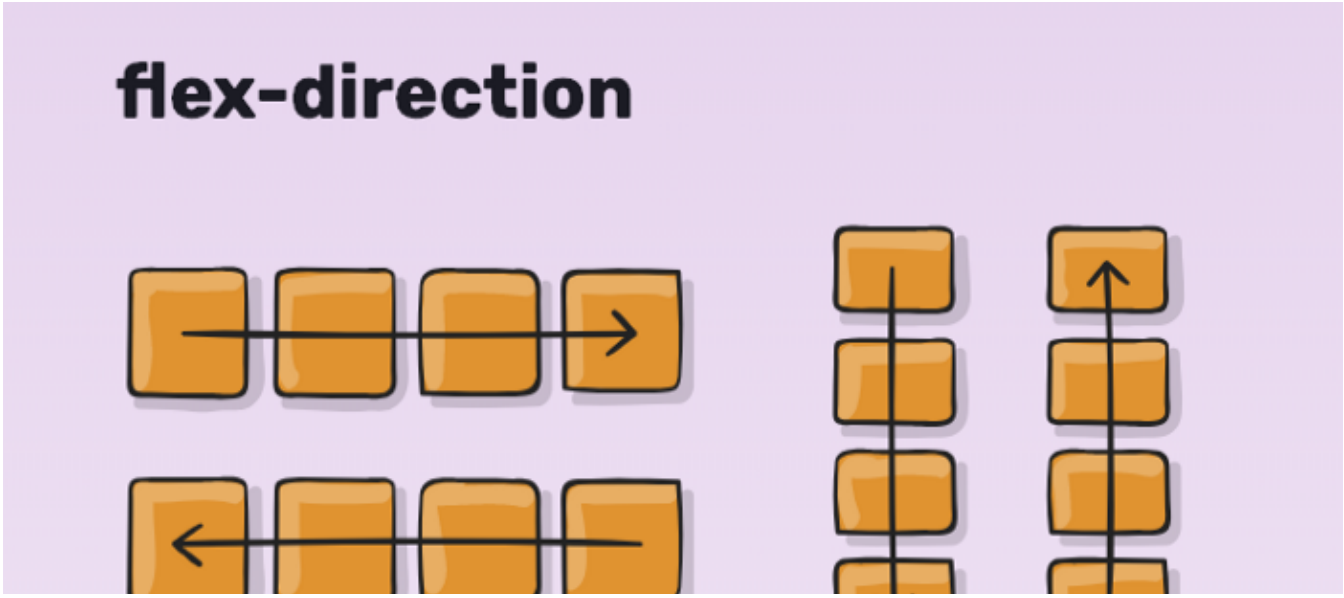
Months ago, I landed my first official job as a front-end developer at one of startup companies in Melbourne. Throughout my time there, I was highly involved in creating the Web User Interface and animation through React JS. From the perspective of a front-end developer who is not a big fan of CSS, I find using Flexbox is a very interesting layout mode. And a bit tricky in a way. In this post, I will try summarise what I understand in using flexbox:

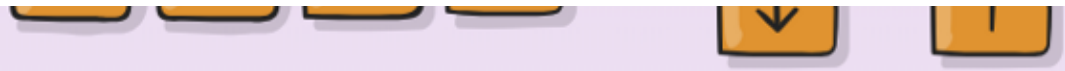
1. First of all, properties for the parent (flex container):

.container {

display: flex;

flex-direction: row/column/row-reverse/column-reverse;





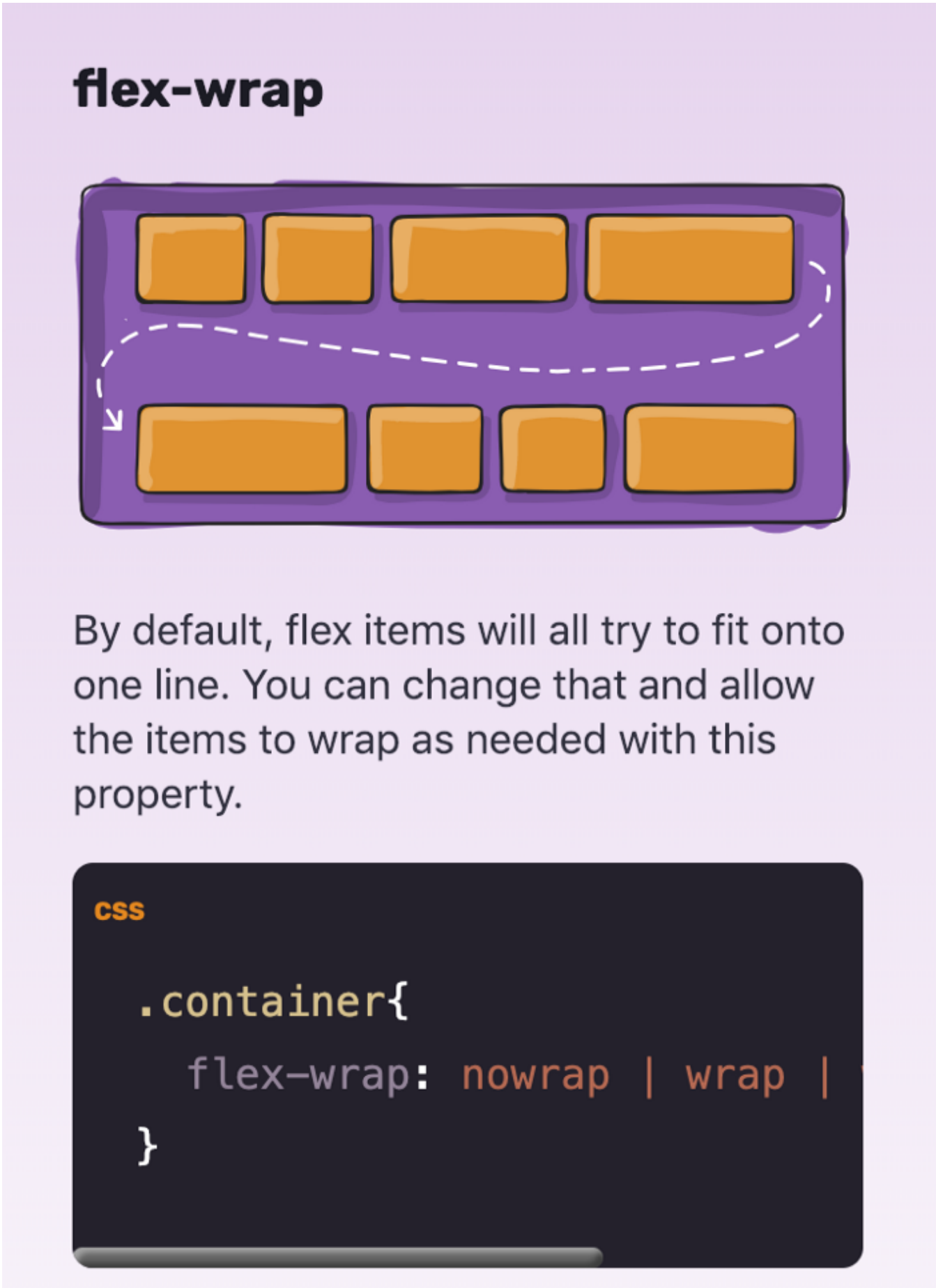
This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

```css
.container {
  flex-direction: row | row-r
}
```

- `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

reverse but bottom to top

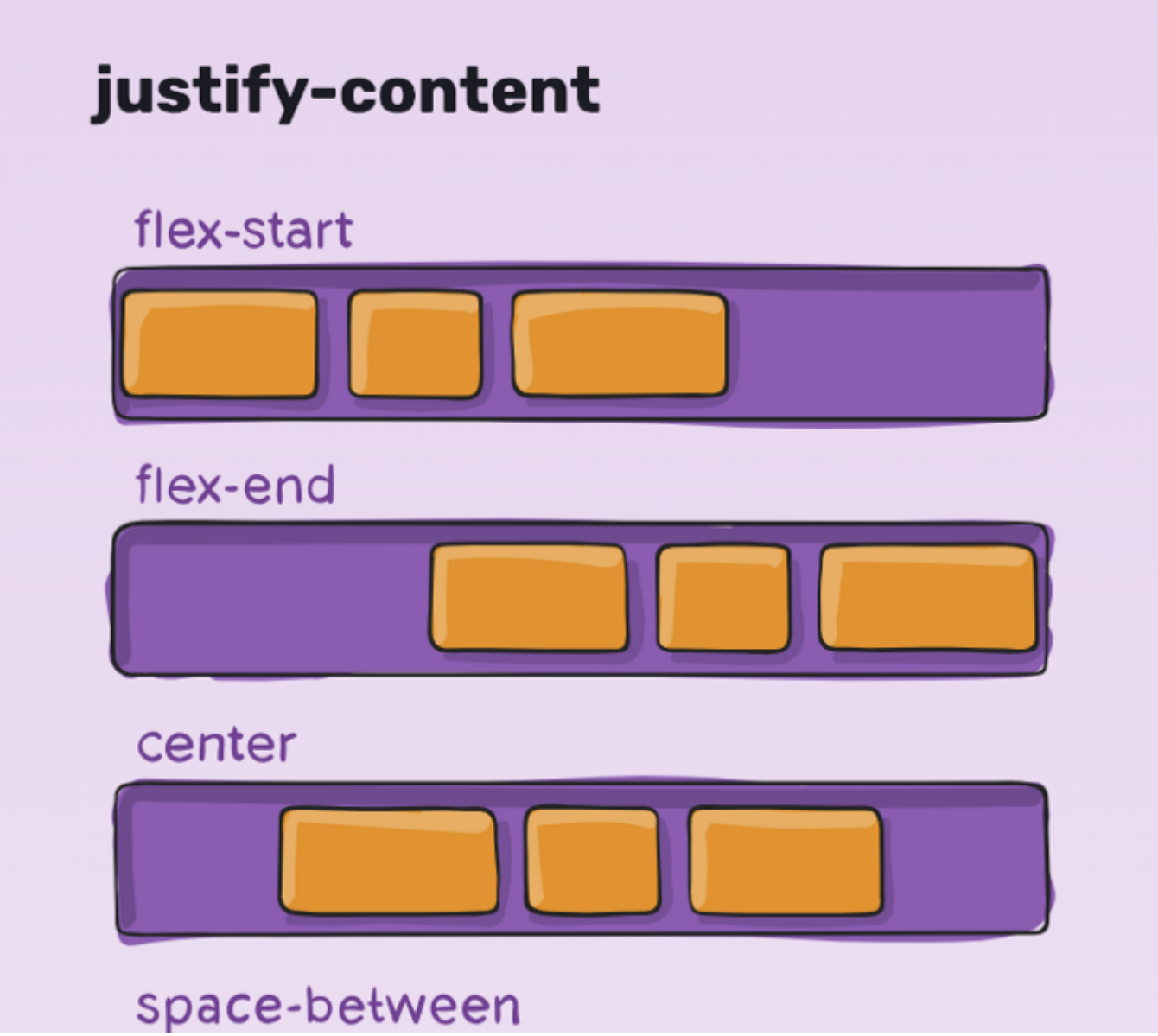
flex-wrap: no-wrap/wrap/wrap-reverse;

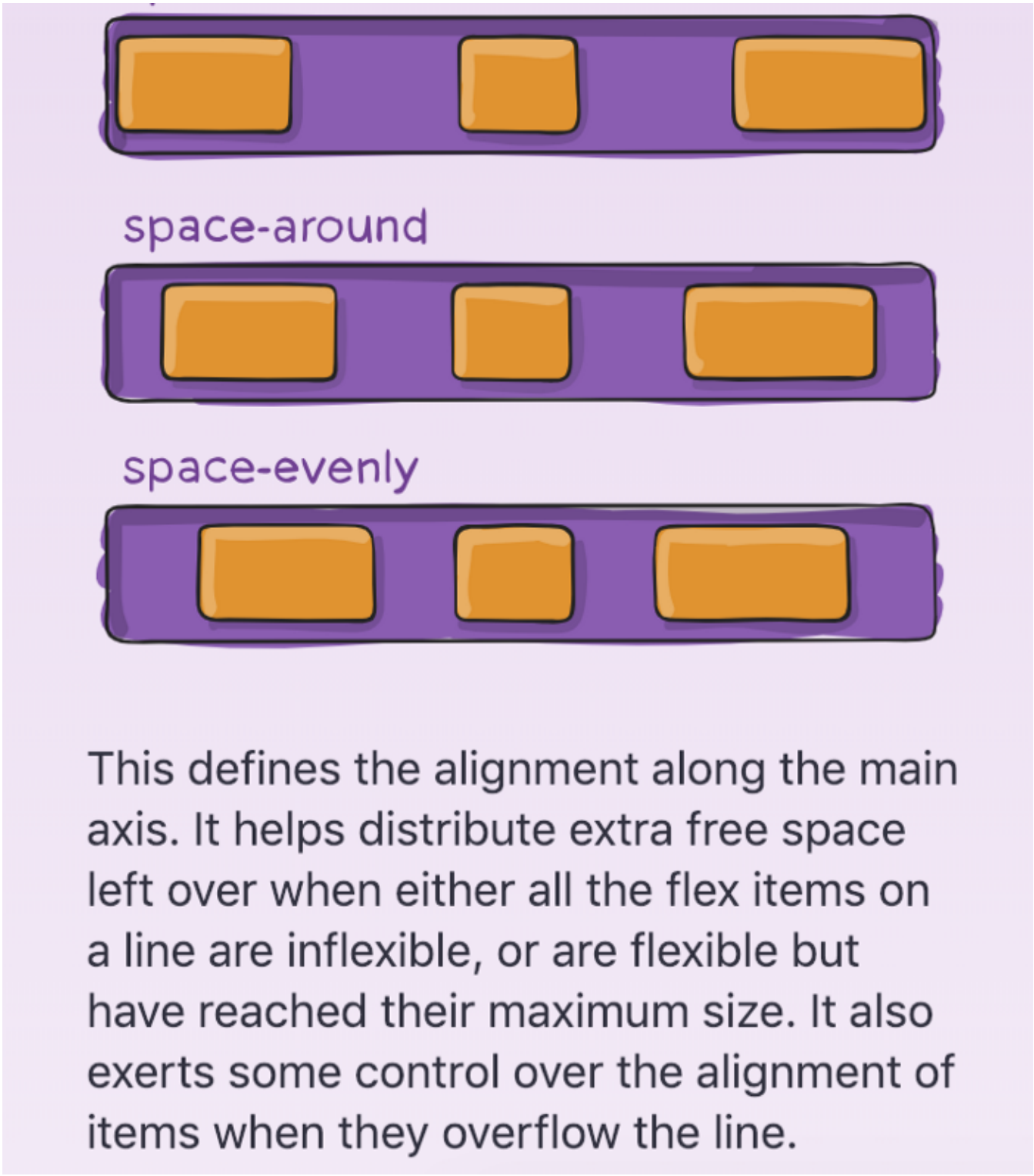


- `nowrap` (default): all flex items will be on one line
- `wrap` : flex items will wrap onto multiple lines, from top to bottom.
- `wrap-reverse` : flex items will wrap onto multiple lines from bottom to top.

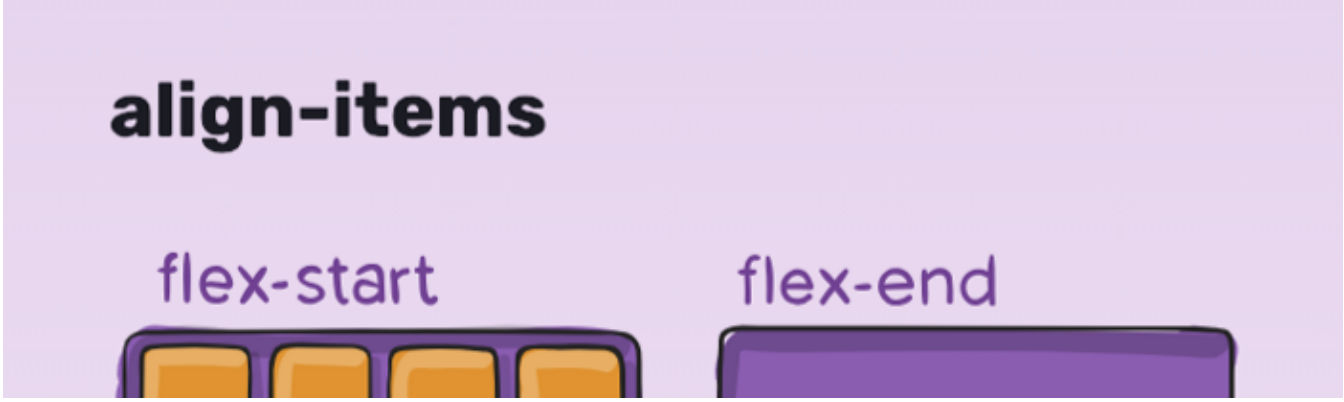
/** horizontal / main axis **/

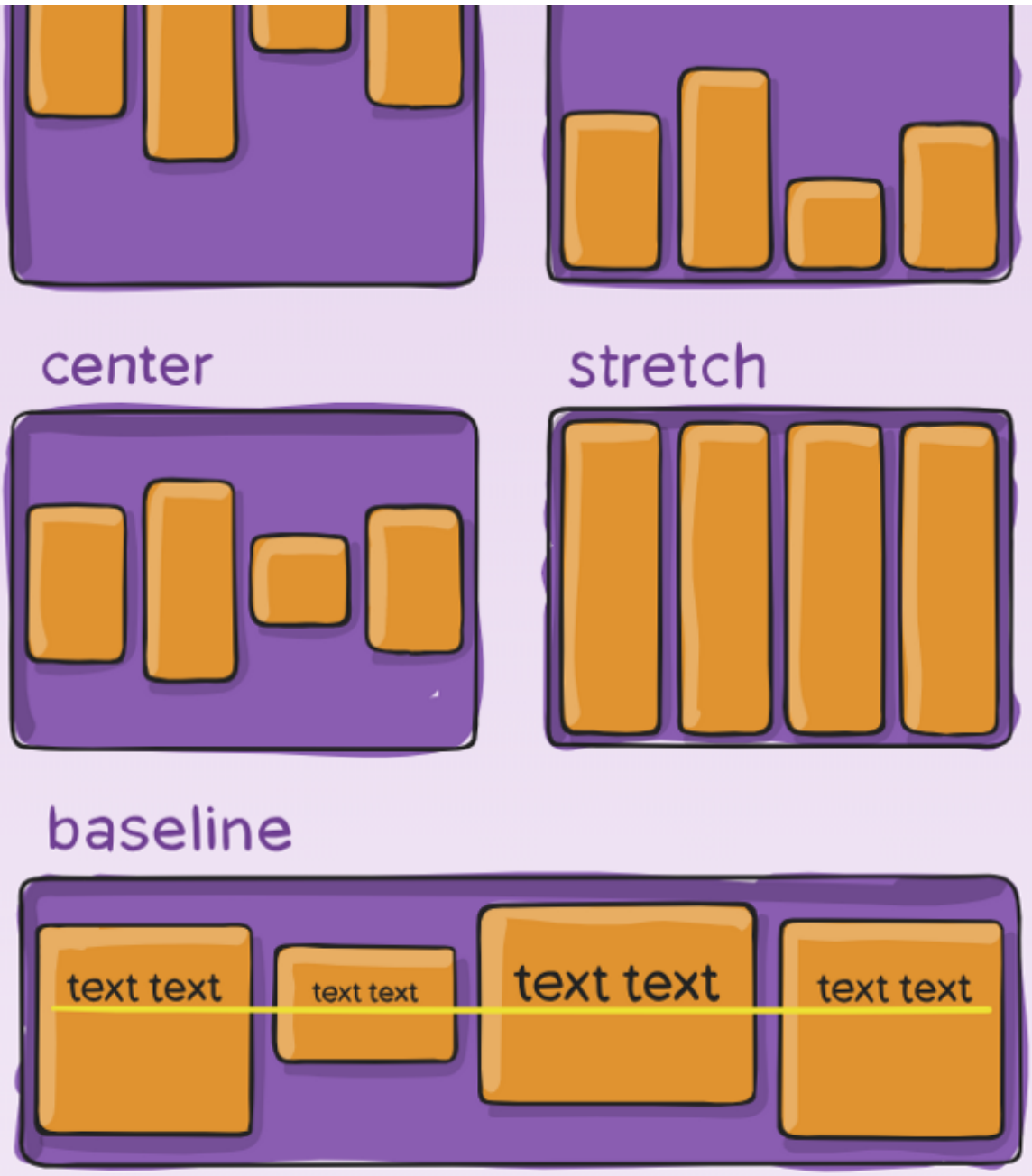
justify-content: flex-start/flex-end/center/space-between/space-around/space-evenly





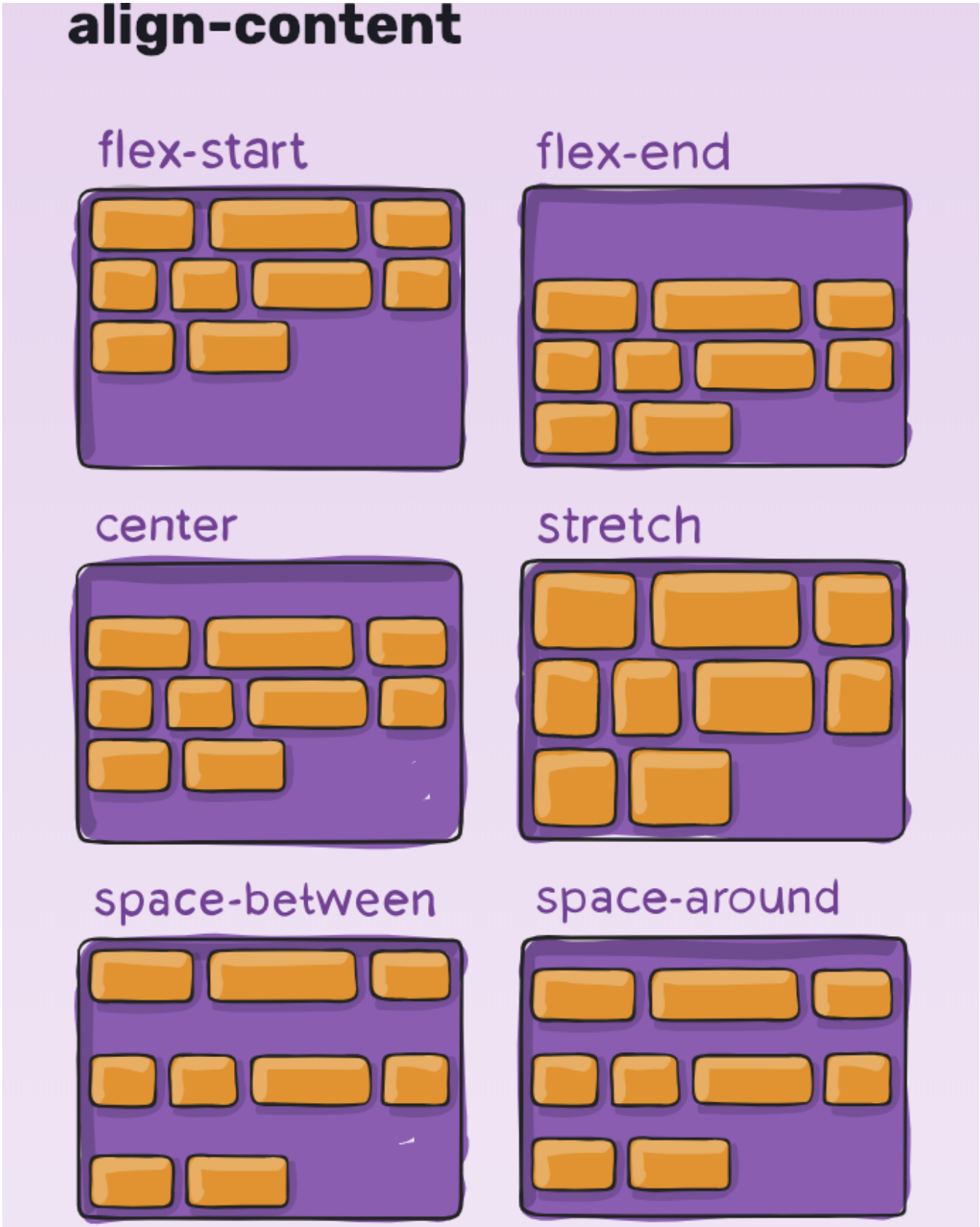
align-items: flex-start/flex-end/center/stretch/baseline





This defines the default behavior for how flex items are laid out along the cross axis on the current line. Think of it as the `justify-content` version for the cross-axis (perpendicular to the main-axis).

align-content: flex-start/flex-end/center/stretch/space-between/space-around



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.

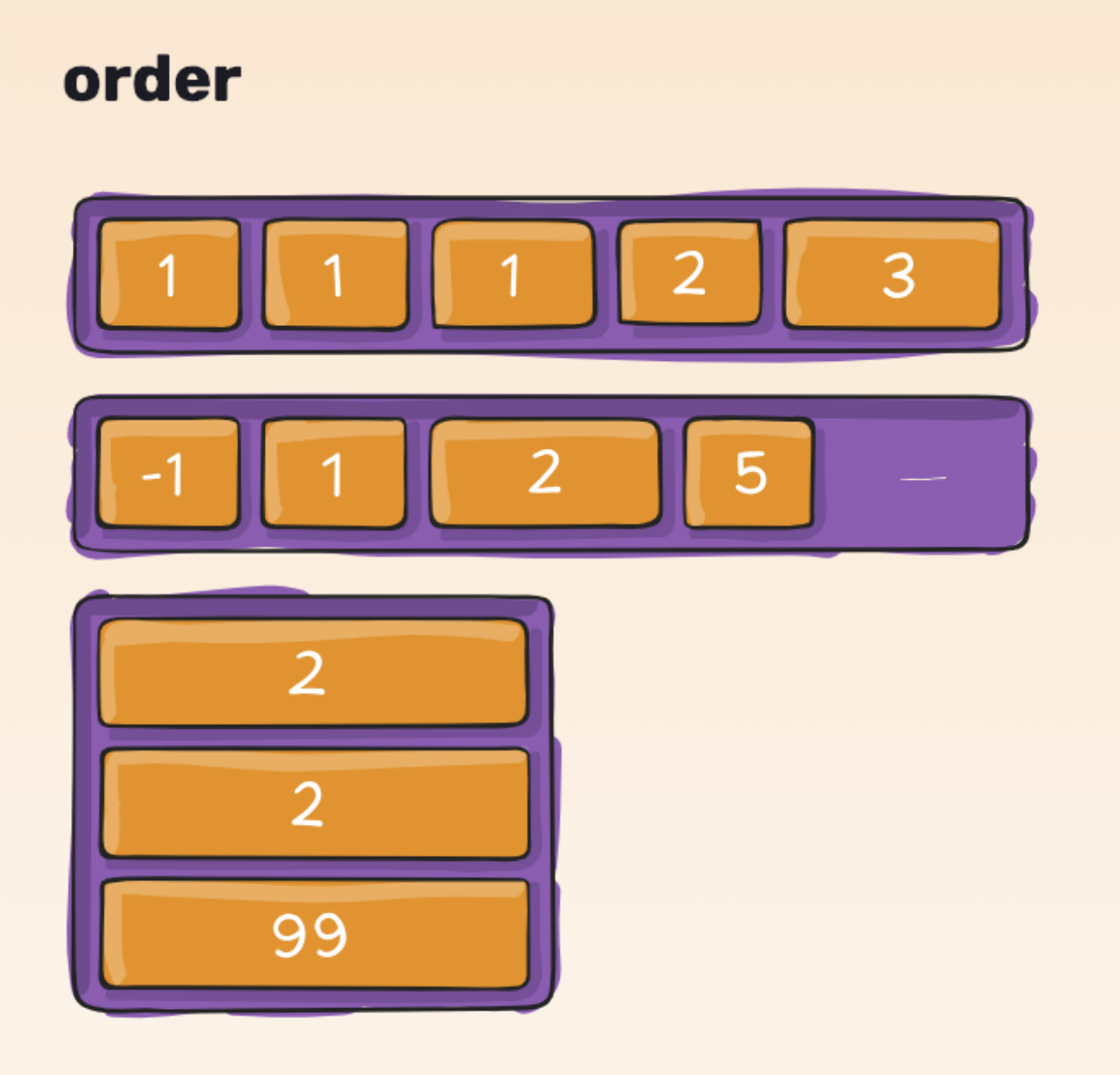
Note: this property has no effect when there is only one line of flex items.

}

2. Secondly, properties for the flex items (child component):

.item {

order: <integer>



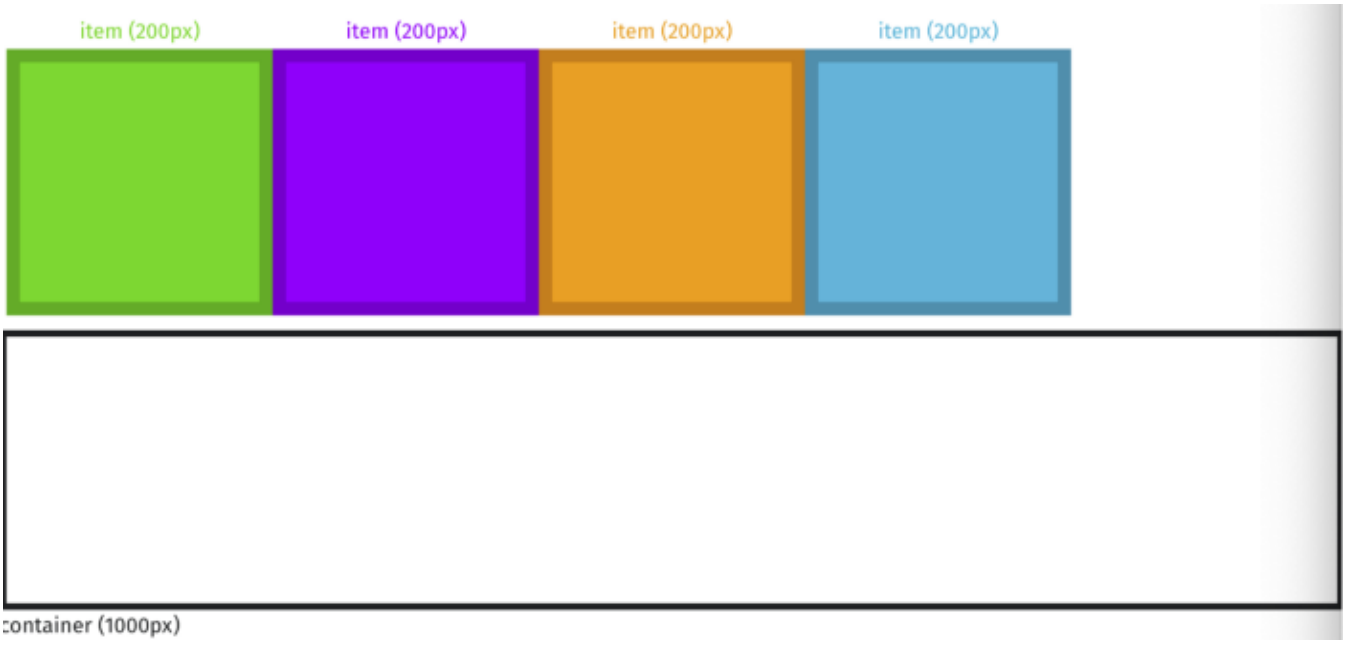
> By default, flex items are laid out in the source order. However, the `order` property controls the order in which they appear in the flex container.

/** the defaults size of an item before the space is distributed among other flex items to suit the flex container size **/

flex-basis: <length> → formula → content — width — flex-basis(limited by max/min-width)

illustration:

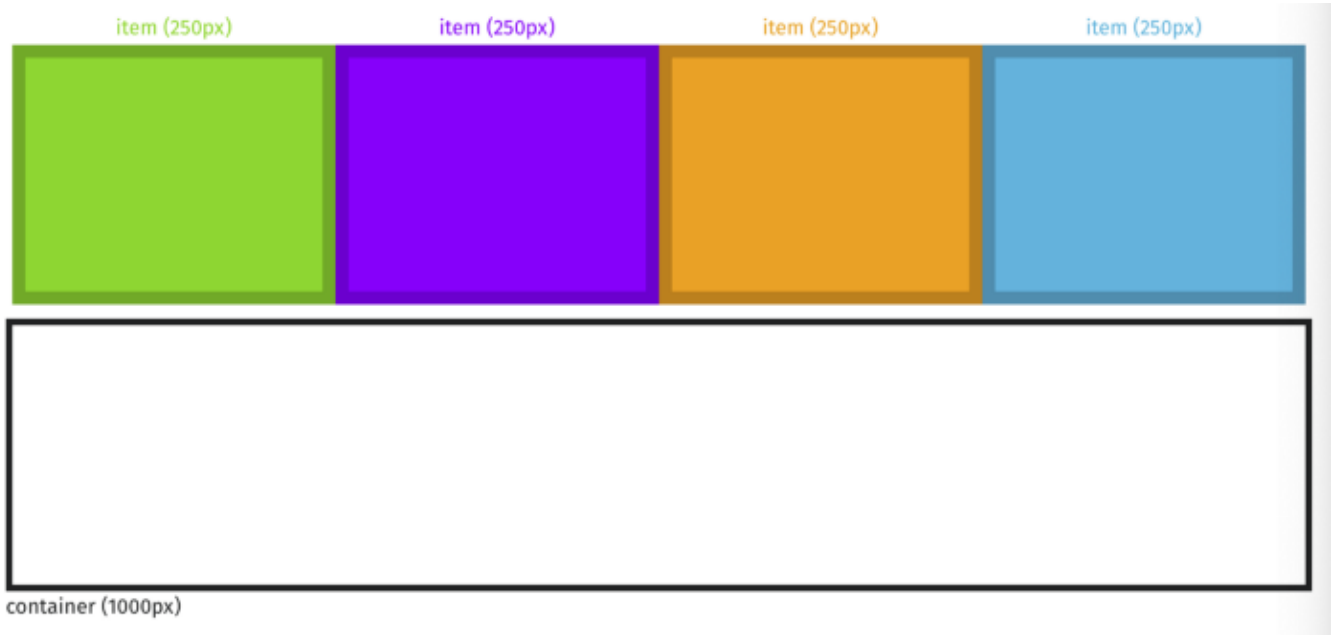
1. container{
   display: flex;
   width: 1000px;
   }
2. item{
   width: 200px;
   height: 200px;
   }



with **flex-basis:**

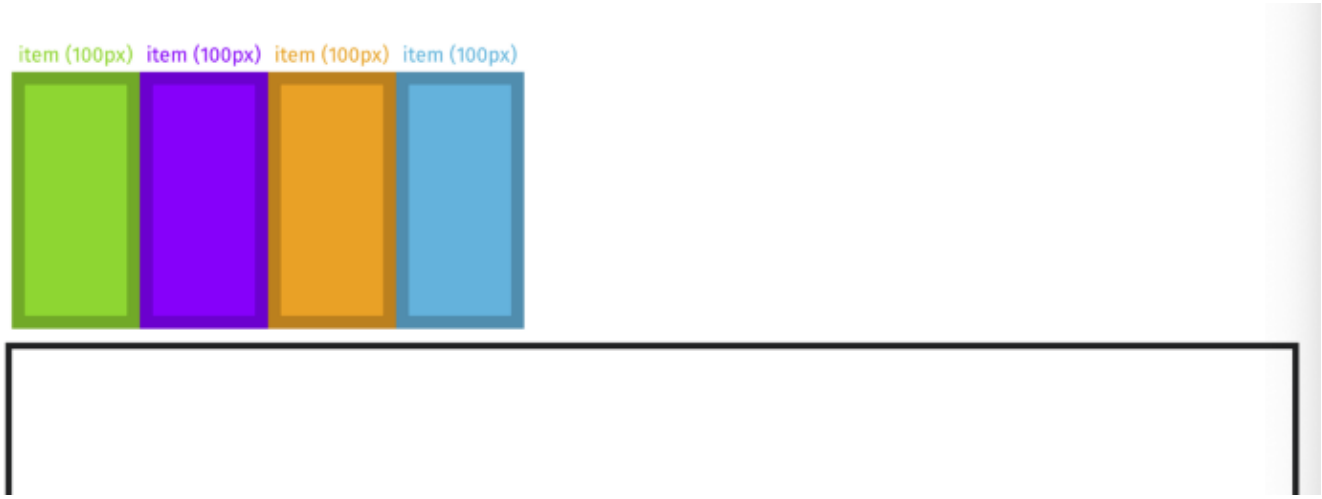
```
item{
width: 30px;
flex-basis: 250px;
}
```

when we specify flex-basis, width of the item won't be counted, so we can just remove the width as flex-basis will adjust to its flex container.

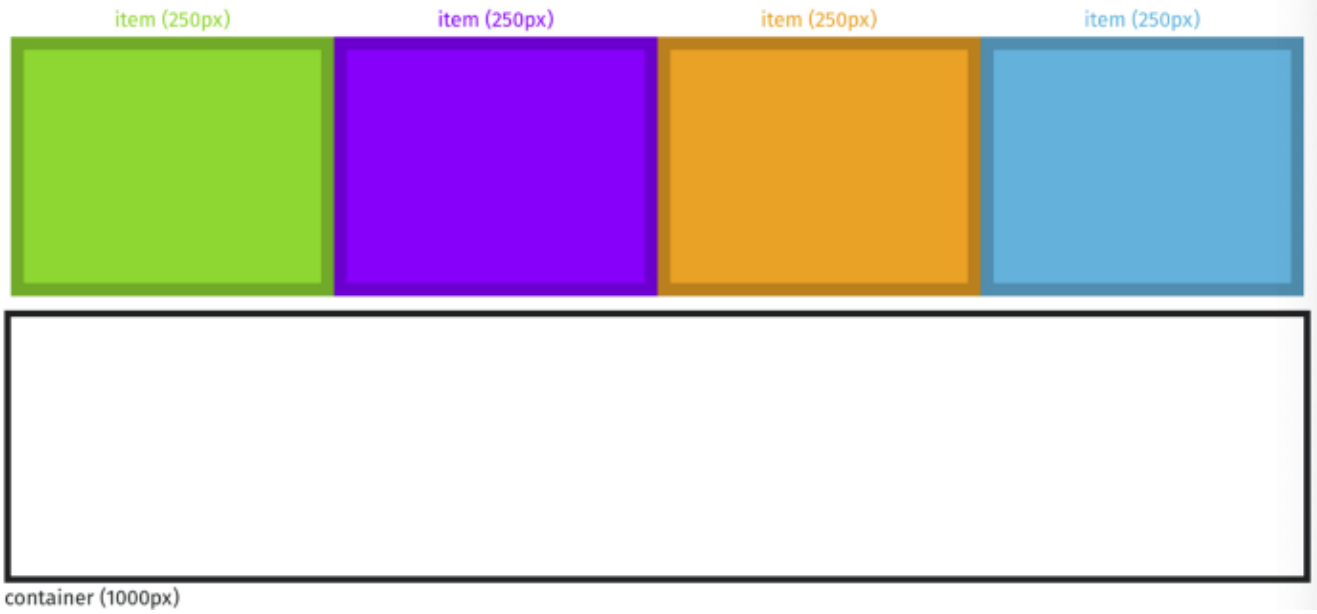


However, flex-basis is limited by max-width.

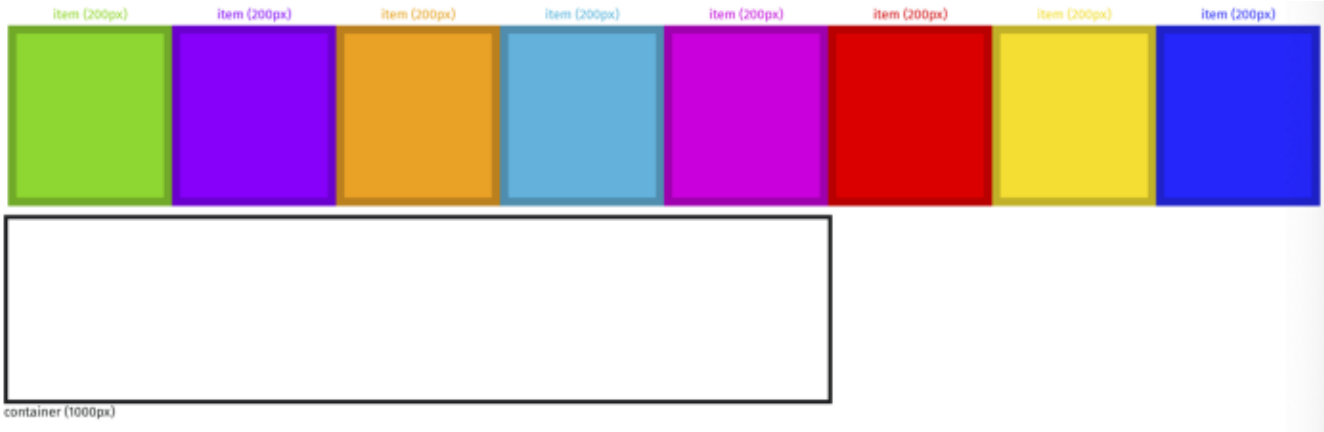
```
item{
flex-basis: 250px;
max-width: 100px;
}
```




```
item{
flex-basis: 100px;
min-width: 250px;
}
```



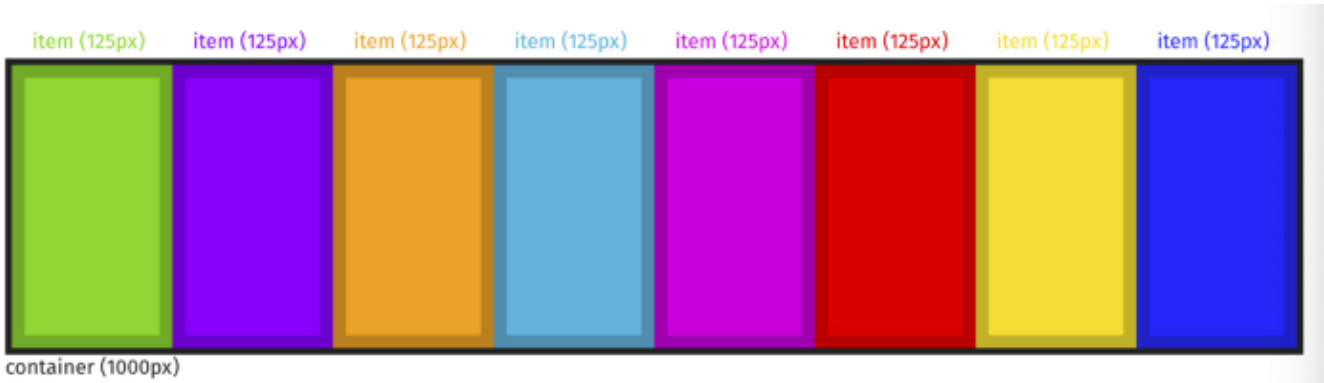
Hence, what exactly is flex-basis? *It's the size of flex-items before the are placed into a flex container. What if not enough space?* For example we want to fit 8 items with flex-basis 200px each into 1000px flex container.



When there's not enough room for all the items' full `flex-basis` (200px each) then flex items by default will *shrink* to fit:

/** the ability of flex item to shrink whenever it doesn't suit the container size **/

flex-shrink: <number>



All of the items started out at 200px wide, but since we were short on space they shrunk at an even rate until they all fit (125px each). That **shrink rate** is what the `flex-shrink` ratio is all about. You can override the default to make specific items shrink faster (higher ratio) or slower (lower ratio) or not shrink at all (0).

What if there is extra space in the container?

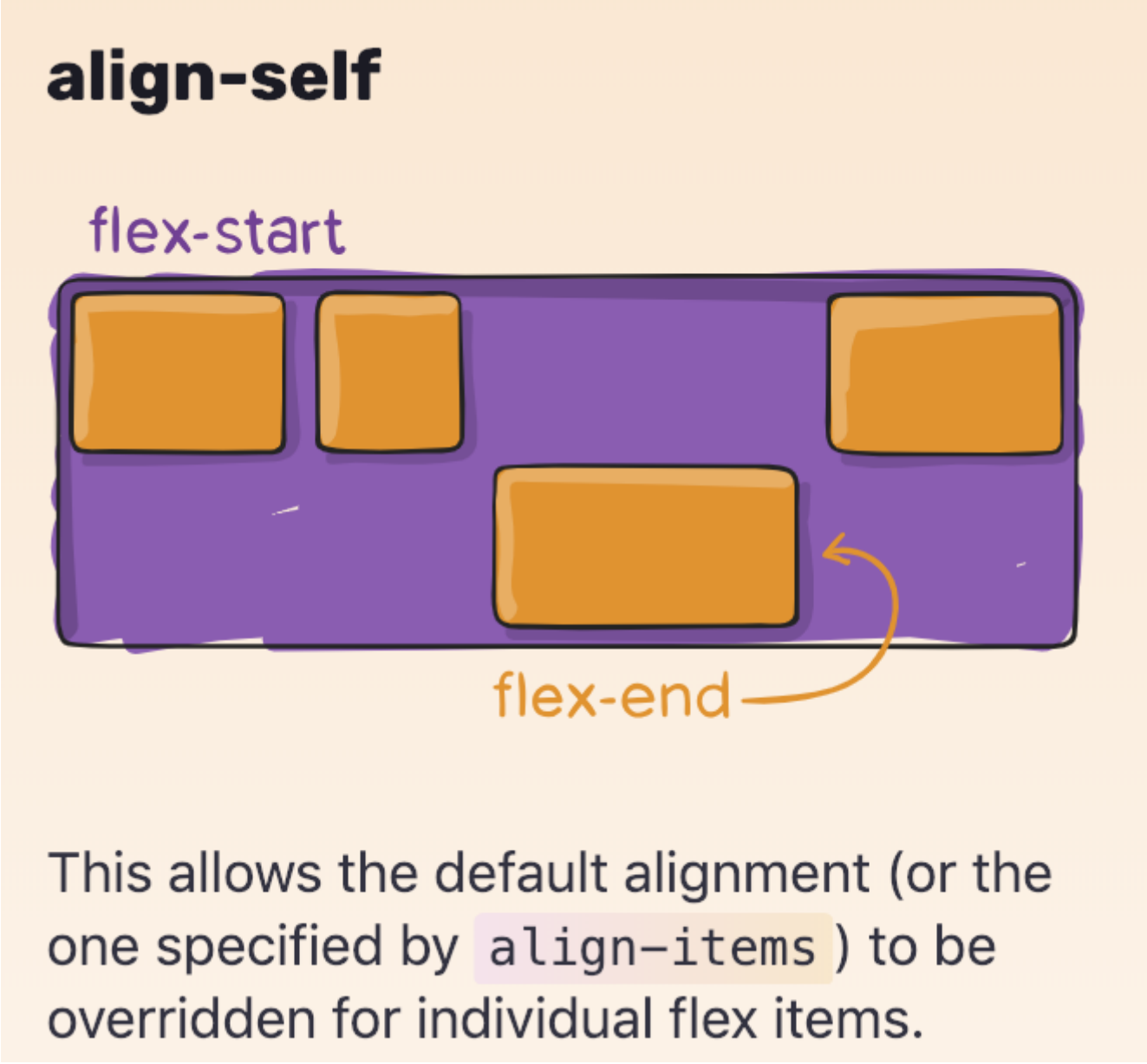
flex-grow: <number>

```
item{
flex-basis: 100px;
flex-grow: 1;
}
```

Growing and shrinking is a huge part of the coolness of flexbox, and is what makes flexbox so great for responsive UI design. Flexbox Zombies mastery game covers `flex-shrink` and `flex-grow` in more detail.

flex: <flex-grow> <flex-shrink> <flex-basis>

align-self: flex-start/flex-end/center/baseline/stretch;



}

Hope you enjoy this post about Flexbox. Personally, I still have so many things to learn about Flexbox, but hopefully this is enough to provide brief on how Flexbox works.

Please find my references here:

1. https://css-tricks.com/snippets/css/a-guide-to-flexbox/
2. https://gedd.ski/post/the-difference-between-width-and-flex-basis/

CSS Front End Development Frontend Flexbox Reactjs