Justification and alignment (OPTIONAL)

Justification and alignment (OPTIONAL)

Note: This material is included for completeness. However, many are able to use flexbox satisfactorily without it. None of the material here will appear in any graded question.

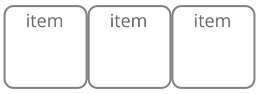
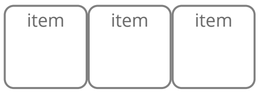
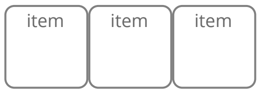
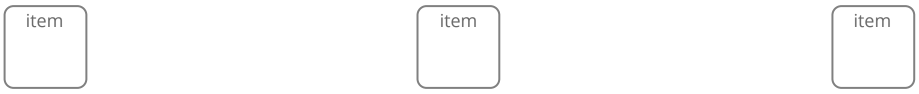
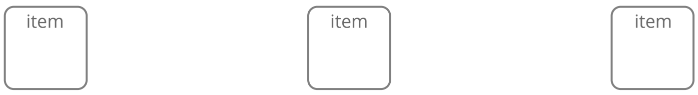
justify-content

```
.fc { justify-content: space-around; }
```

When all the flex items in a flexbox container are fully resizable, then there will not be any extra space to put between the items. However, when the flex items are fixed size, or cannot grow anymore, then the flexbox container will put the extra space between or outside the items. The closest analogue from typography is called "justifying". Thus, the `justify-content` property serves a similar function for flexbox containers.

The `justify-content` property is applied to the flex container. It governs how any extra space along the main layout axis is distributed between the flexbox items. The possible values are: `flex-start`, `flex-end`, `center`, `space-between`, and `space-around`. The `flex-start`, `flex-end`, and `center` values do not distribute any space between the flex items. Instead, these values determine where the flex items should be positioned within the flex container, and any extra space is outside them. The `space-between` and `space-around` values both put space evenly between the flex items, but `space-between` places the flex items flush against the main start and main ends of the flexbox container. Remember that this is only spacing in the direction of the main axis. `justify-content` does not affect any spacing or placement in the direction of the cross axis.

The table below should help illustrate this. It shows the justification options for a flexbox container with `flex-flow:row`;

flex-start	
center	
flex-end	
space-between	
space-around	

If the `flex-direction` were `row-reverse`, then the only thing to change in the table above would be that the appearances of `flex-start` and `flex-end` would be reversed.

If the `flex-direction` were `column`, then remember that, if the flexbox container is a block level element, its default size would be that of its content. Which means there would be no extra vertical space to distribute. So all the five options above would be identical (a tight stack of items) *unless* the height of the flexbox container were explicitly made larger.

align-content and align-items

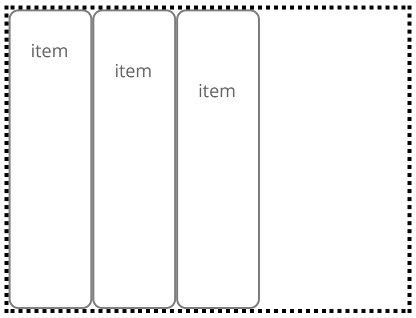
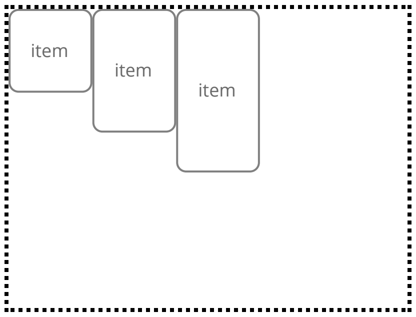
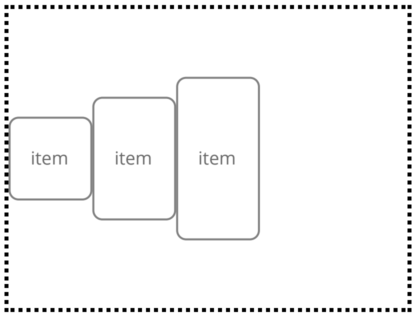
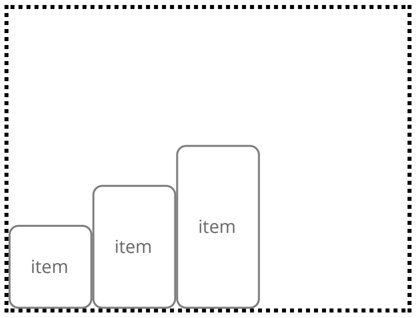
The `align-content` and `align-items` are often confused for one another. But they are very different. Both properties only apply if there is extra space in the cross axis direction. This is important to remember, because in many situations there isn't any cross axis space. In the example above (used for `justify-content`), none of the flexbox containers has any extra cross axis space (vertical space).

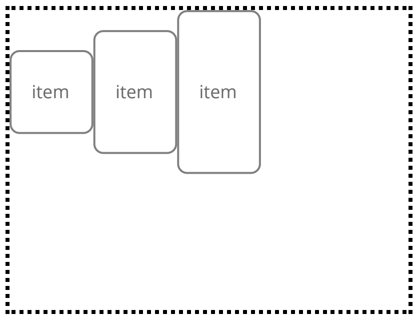
align-items

```
.fc { align-items: stretch; }
```

The `align-items` determines how items are aligned in the cross axis direction. This is applied to the flexbox container. The possible values are `stretch`, `flex-start`, `flex-end`, `center`, and `baseline`. In the context of alignment, `flex-start` and `flex-end` refer to the cross start and cross end sides, which may be swapped if the `flex-wrap:`wrap-reverse option is elected. `align-items` defaults to `stretch`, if it is not set. The table below should help illustrate this. It shows a flex container with `flex-flow:row;` and a `min-height` value that is greater than the height of any of the items.

In the example below, each item has a different `line-height` value, so you can see how they align to each other.

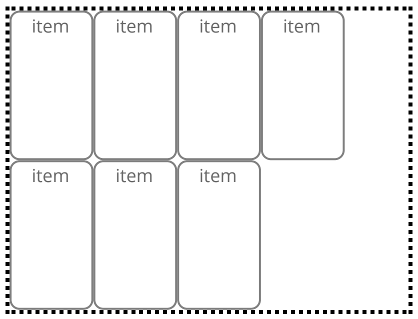
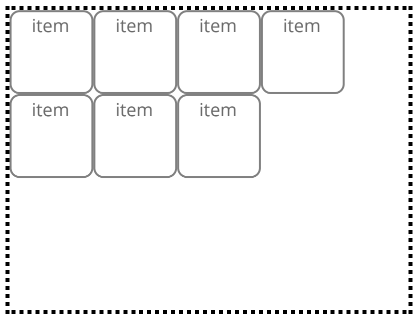
		Notes
stretch		stretch will not work on flex items that have a fixed dimension. Use min-width or min-height instead of width or height, if you want the item to actually stretch.
flex-start		
center		
flex-end		

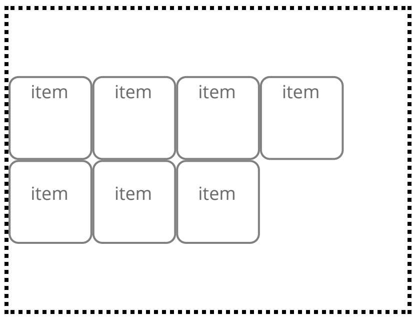
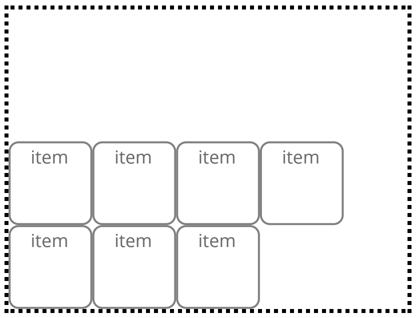
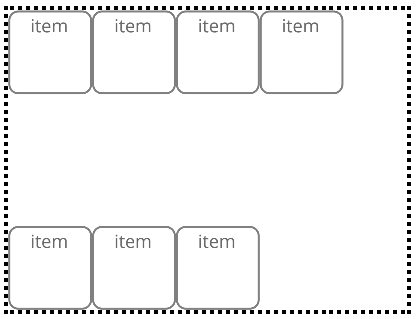
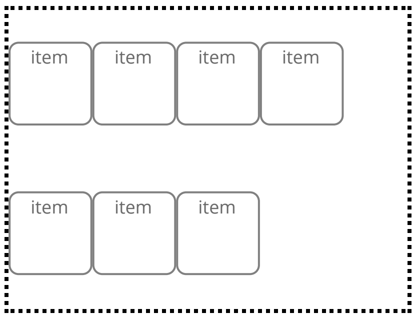
baseline		

align-content

```
.fc { align-content: space-between; }
```

`align-content` is only relevant when the flexbox container supports wrapping and the flex items are, in fact, wrapping. The `align-content` determines how the *wrapped lines* are positioned or spaced. `Align-content` is not applied to individual items, but rather to the wrapped lines. The `align-content` property supports the `stretch` value, as well as the same values as `justify-content` (`flex-start, center, flex-end, space-between, space-around`). This is easier to understand from an example. Below we have a flex container with `flex-flow:row wrap;` and a height value that is greater than the height of any of the items.

		Notes
stretch		stretch will not work on flex items that have a fixed dimension. Use min-width or min-height instead of width or height, if you want the item to actually stretch.
flex-start		

center		
flex-end		
space-between		
space-around		

align-self

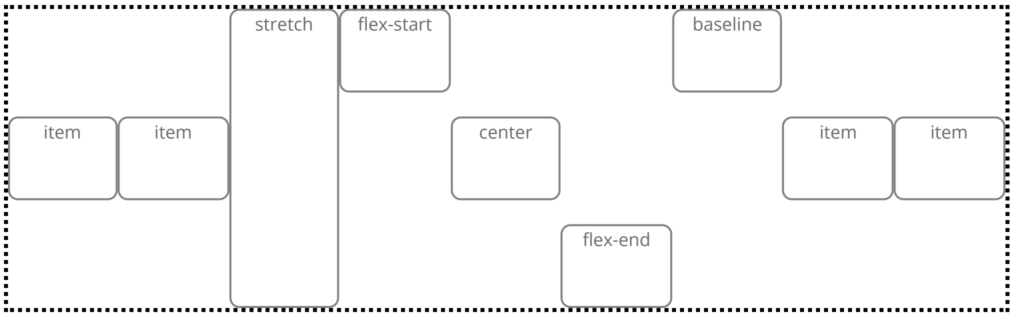
```
.item { align-self: center; }
```

Unlike the other flexbox align properties, `align-self` is applied to an individual flex item, not to a flexbox container. This allows an individual flex item to be aligned differently than its siblings. Again, this is only true for cross axis alignment, and will only come into play, if there is extra space in the cross axis direction to be exploited.

The values for `align-self` are `stretch`, `flex-start, center, flex-end`, and `baseline`.

`align-self` is ignored, if any of the four margins on the item is set to `auto`.

In the example below, we have a flex container with `flex-flow:row;` and `align-items:center;`. The individual items have their `align-self` property set.

align-self	**Notes**
 item item stretch flex-start center flex-end baseline item item	stretch will not work on flex items that have a fixed dimension. Use min-width or min-height instead of width or height, if you want the item to actually stretch.