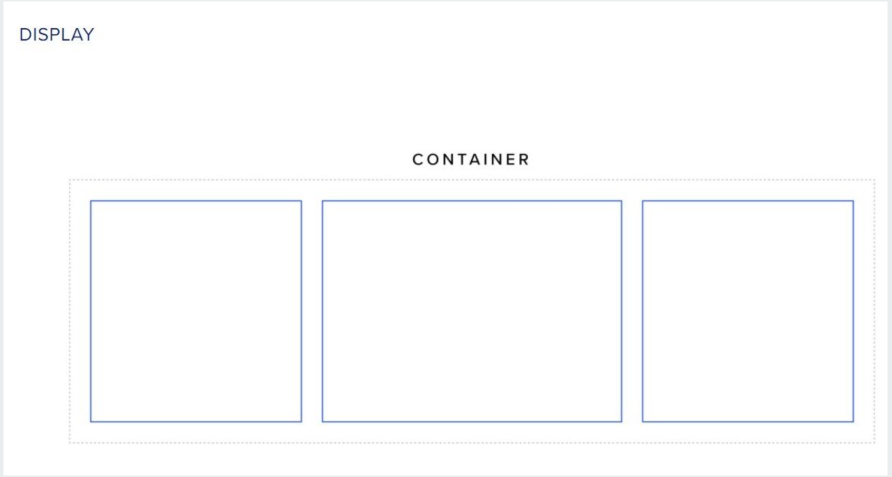
# Flexbox

# display: flex;

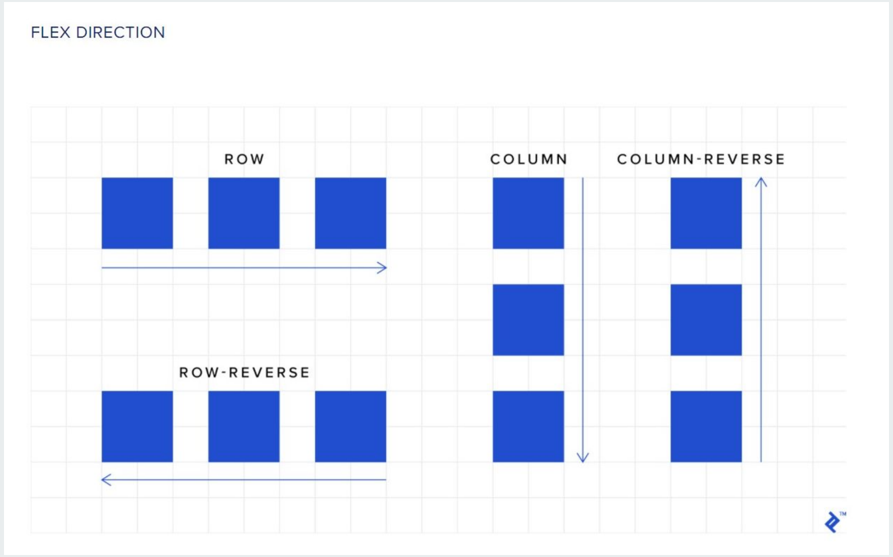
*Applies to parent element*
Initiates flexbox on the parent and child elements.

*container {display: flex;}*



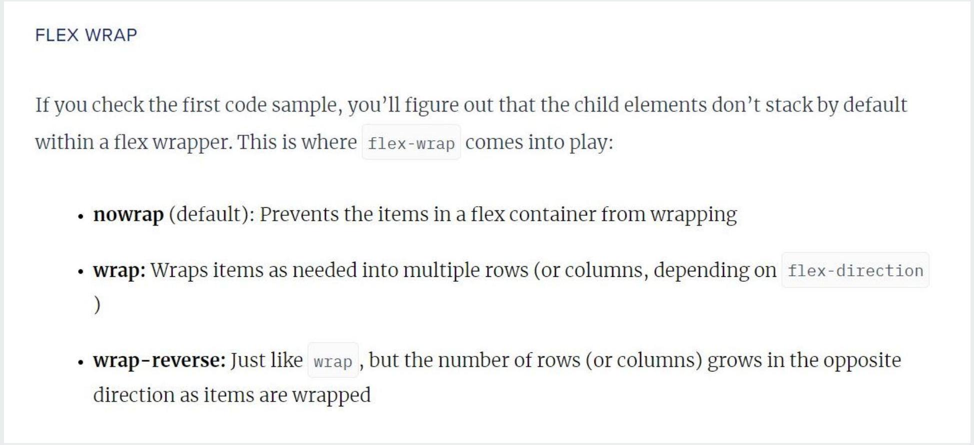
# flex-direction

*Applies to parent element*
Defines vertical or horizontal behaviour and direction. Also applied on the parent/container element.



# flex-wrap

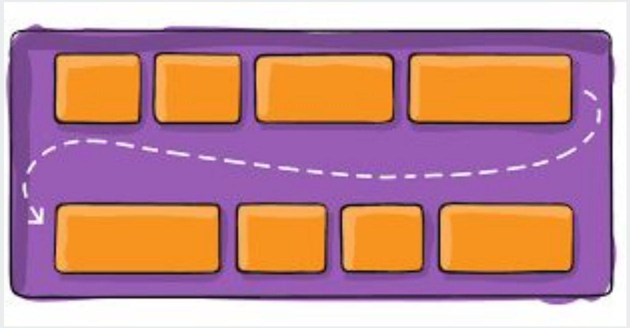
*Applies to parent element*
Wraps items into multiple rows if needed.



FLEX WRAP

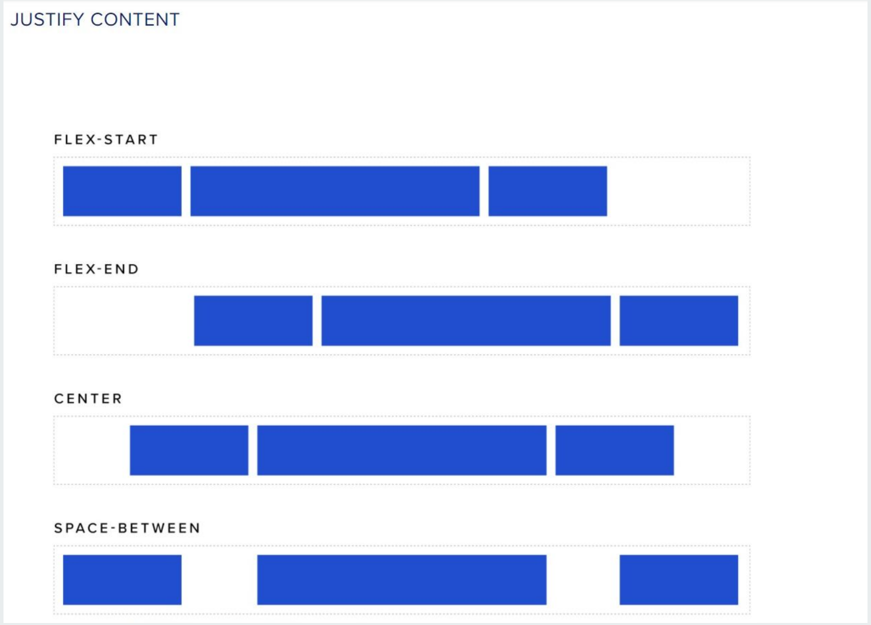
If you check the first code sample, you'll figure out that the child elements don't stack by default within a flex wrapper. This is where `flex-wrap` comes into play:

- **nowrap** (default): Prevents the items in a flex container from wrapping
- **wrap:** Wraps items as needed into multiple rows (or columns, depending on `flex-direction` )
- **wrap-reverse:** Just like `wrap`, but the number of rows (or columns) grows in the opposite direction as items are wrapped



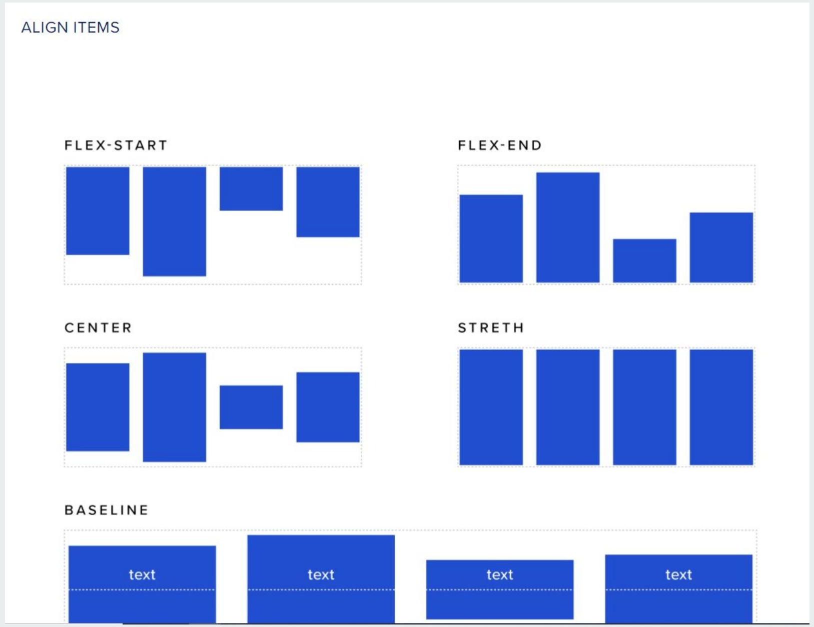
## justify-content

*Applies to parent element*
This property is used to control the horizontal alignment of the child elements



# align-items

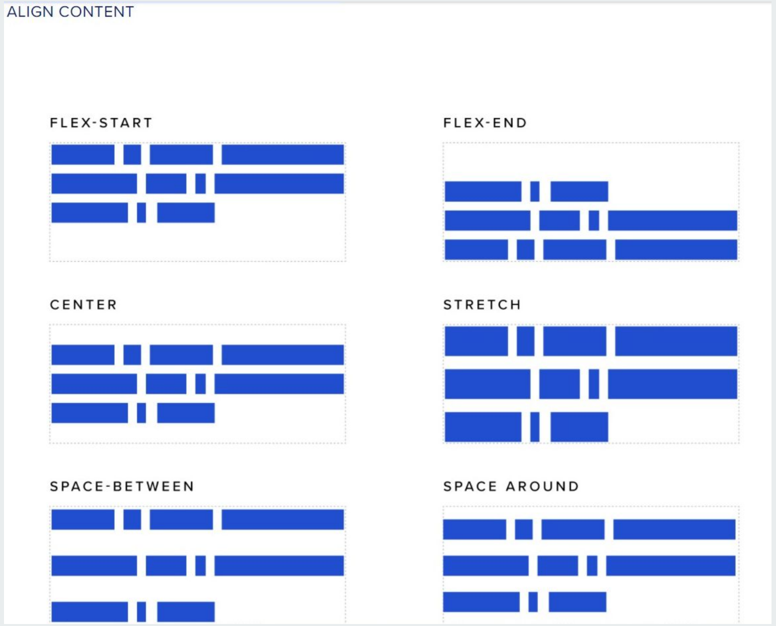
*Applies to parent element*
This property is similar to justify-content but the context of its effects is the rows instead of the wrapper itself.



## align-content

*Applies to parent element*

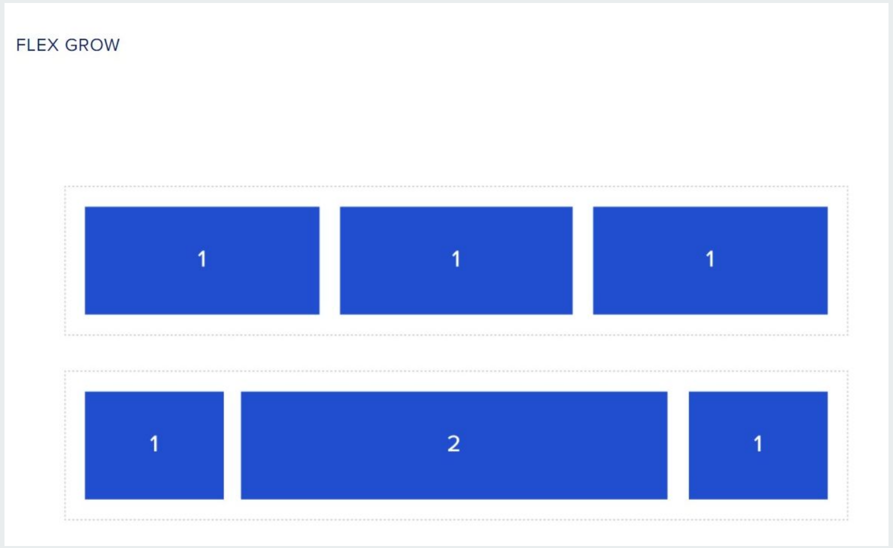
This property is similar to justify-content and align-items but it works in the vertical axis and the context is the entire wrapper (not the row like the previous example). To see its effects, you will need more than one row



# flex-grow
# flex-shrink

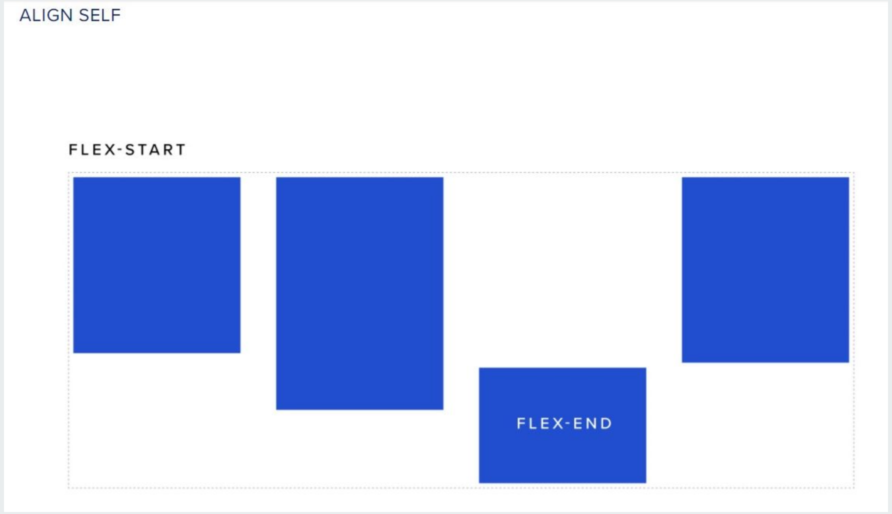
This property sets the relative proportion of the available space that the element should be using. The value should be an integer, where 0 is the default.

Shrink is similar to flex-grow, this property sets whether the element is "shrinkable" or not with an integer value (opposite to flex-grow).



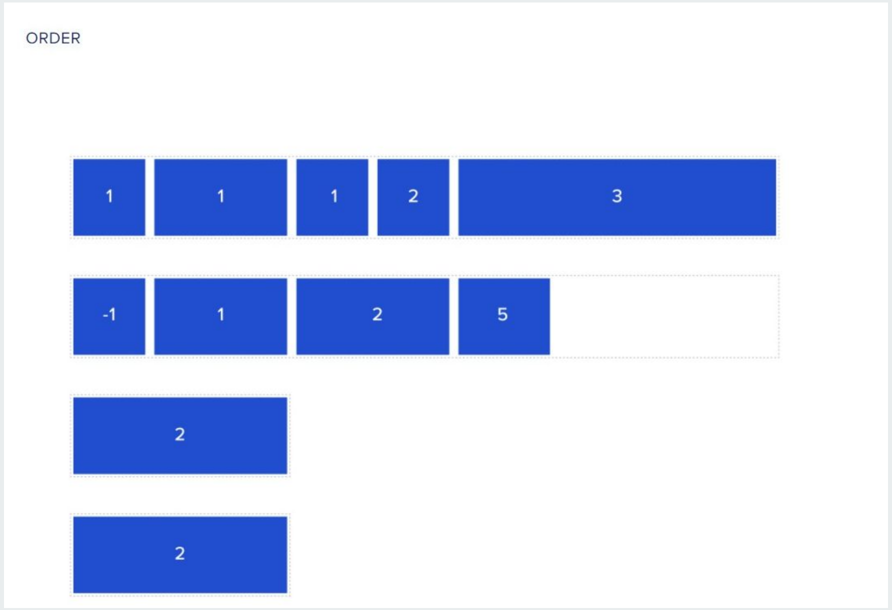
# align-self

This property is similar to align-items but the effect is applied individually to each element. The possible values are



## order

In the same way that z-index controls the order in which items are rendered, order controls the order in which elements are positioned within the wrapper; that is, elements with a lower order value (which can even be negative, by the way) are positioned before those with a higher order value.



## References:

https://www.toptal.com/front-end/how-to-build-css-only-smart-layouts-wit
h-flexbox

https://tobiasahlin.com/blog/common-flexbox-patterns/