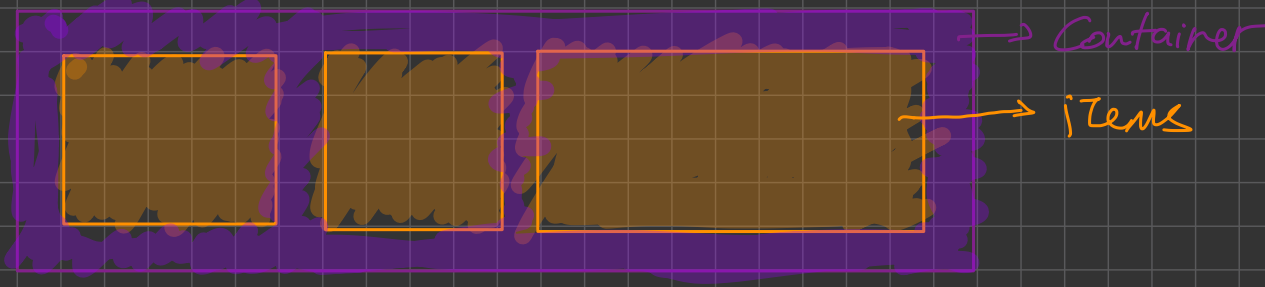
Flex - properties



Properties for the parent (container)

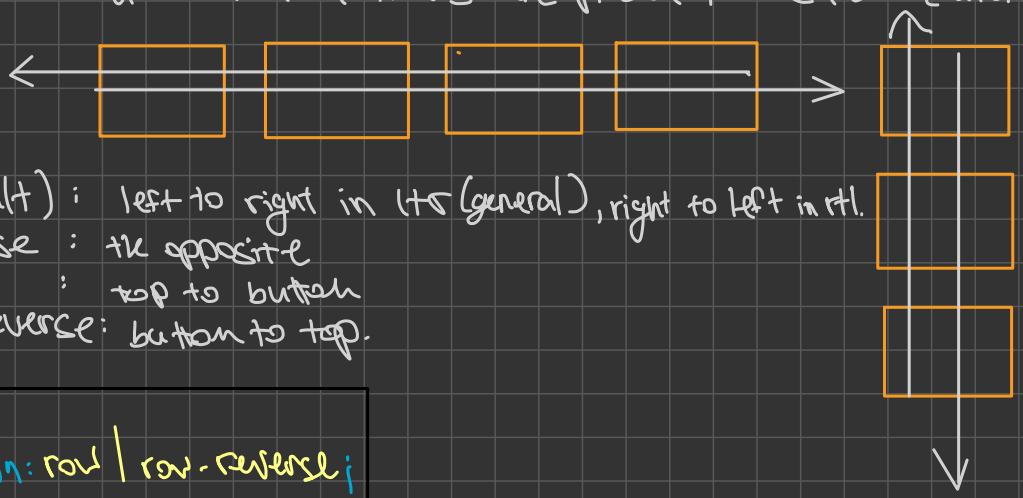
1. Display : we need to define the flex container first (usually by class). it enable a flex context for all its direct children.

```
.container {
    display: flex }
```

* note that the css column have no effect on a flex container

---

2. Flex-Direction: this established the main axis (defining the direction flex-items are placed in the flex container.



- row (default) : left to right in ltr (general), right to left in rtl.
- row-reverse : the opposite
- column : top to button
- column-reverse: button to top.

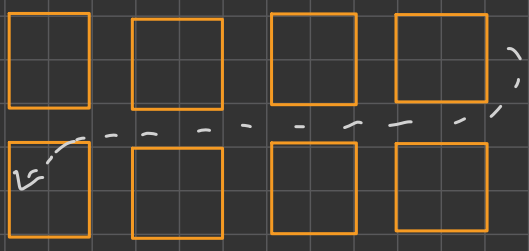
```
.container{
  flex-direction: row | row-reverse;
        }
```

---

3. Flex-wrap: by default, flex items will all try to fit in on line. we can change it by wrapping them.

- no-wrap : default, all items in one line
- wrap : all items will be by order, multiple lines, from top to bottom.
- wrap-reverse: items will wrap to multiple items in reverse-direction.

```
.container {
  flex-wrap: no-wrap ;
  }
```



4. Flex-Flow:

Shorthand for the flex-direction and flex-wrap properties. which together define the container main and cross axes (default value is "row-no wrap")

```
.container {
  flex-flow: column wrap;
}
```

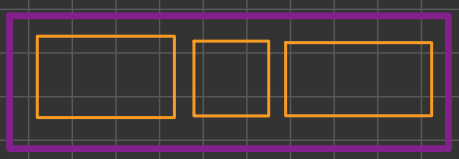
5. justif-content

this define the alignment along the main axis its help distribuite extra free space leftover.
it also exerts some control over the alignment of items when they overflow the line.

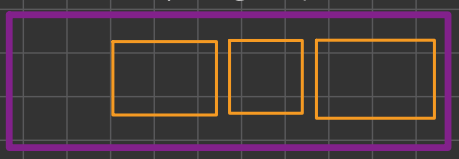
- Flex-start : (default) items are packed toward the start of the flex-direction.
- Flex-end : items are packed toward the end of the flex-direction.
- start : items are packed toward the start of the writing mode direction.
- end : items are packed toward the end of the writing mode direction.
- left : it like are packed toward left of the container. unless that dosent make sense with the flex-direction than its behave like "start".
- right : items are packed toward the right of the container. unless that dosent make sense (flex-direction) than its behave like "end"
- center : items are centered along the line
- space-between : items are evenly distributed in the line (first item-"start", second-"end").
- space-around : items are evenly distributed in the line, with equal space around them.
- space-evenly : items are distributed so that the space between two items and the space from the edges is equal!

NOTE - this is depens on browser support) the "safest" values are flex-start, flex-end, center.

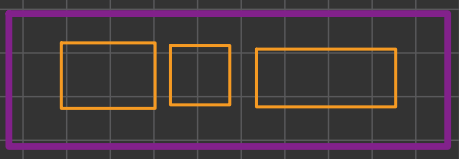
flex-start



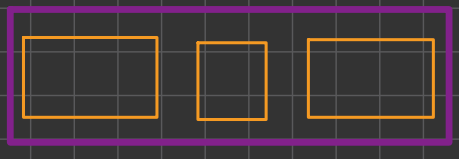
flex-end



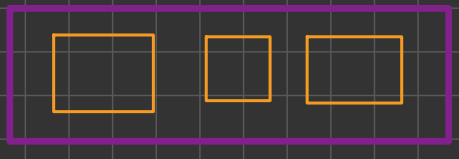
center



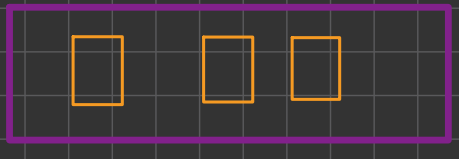
space-between



space-around



space-evenly

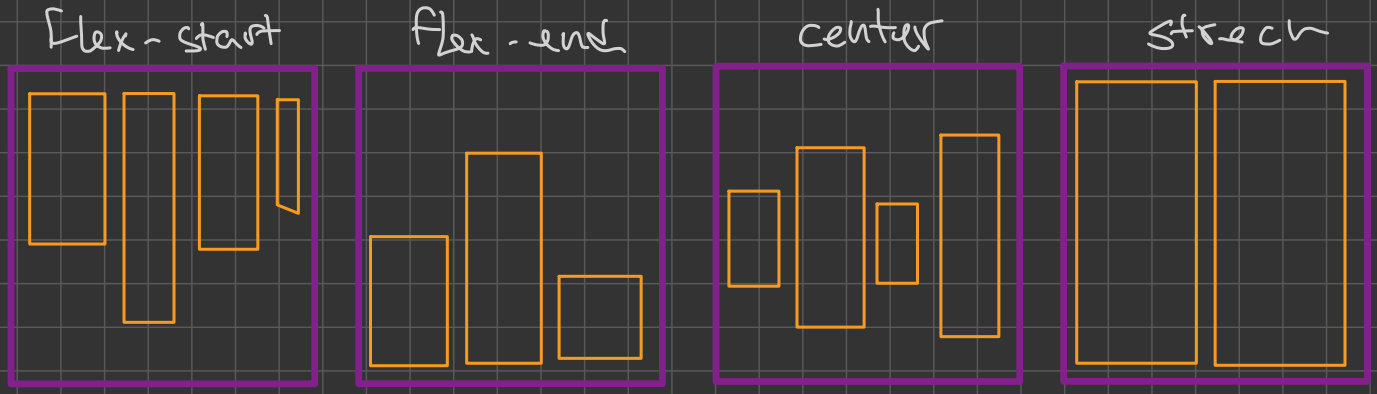


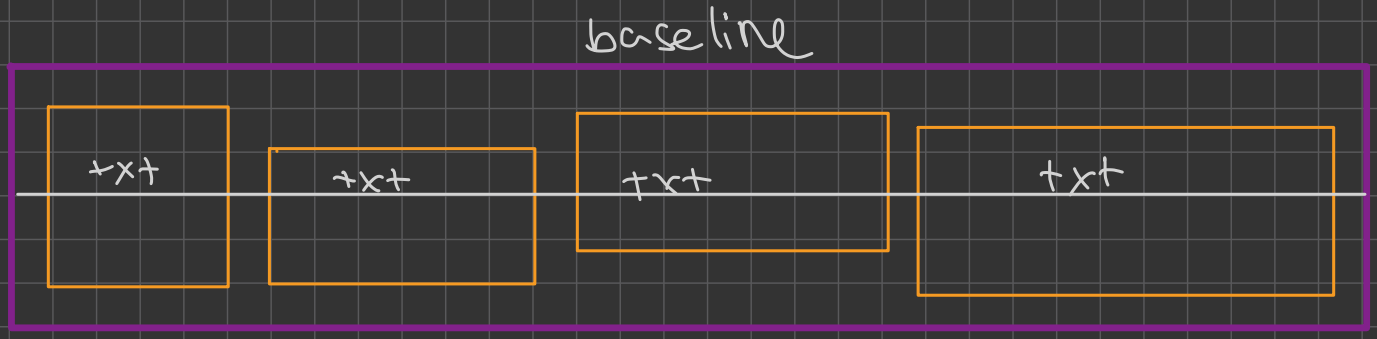
```
.container {
  justify-content: flex-start
  safe / unsafe }
```

* there are also two additional key words you can pair: using safe ensures that however you do this type of positioning you can't push an element off-screen, in such a way the content can't be scrolled too.

6. align-items: this define the default behaivor for how flex items are laid out along the cross axis

- stretch (default) : strech to fill the container. (still respect min-width / max-width)
- flex-start / start : (/self-start as well) items are placed at the start of the cross axis. the difference: subtle, is about respecting the flex-direction rules or the writing-mode rules.
- flex-end / end : (/self-end as well) items are placed at the end of the cross axis, the difference: subtle as well. by flex-direction / writing-mode rules.
- center : items are centerd in the cross-axis.
- baseline : items are aligne such as their baseline





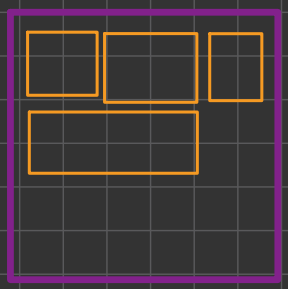

```
. container {
align-items: strech +
safe / unsafe
}
```

7. align-content : this aligns a flex containers line within where ther is an extra space in the cross-axis

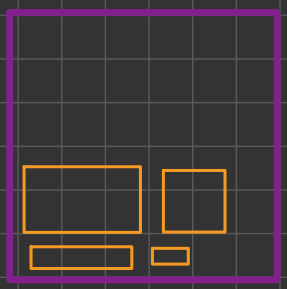
NOTE: this property only take affects on multi-line flexible containers!

- normal (defult) : items are packed in their defult pos. as if no value sets.
- flex-start / start : items are packed to the start of the container
- flex-end / end : items are packed to the end of the container.
- center : items are centered in the containers
- space-between : items evenly distributed (the first line- the start of the container).
- space-around : " " " with equal space around EACH LINE
- space-evenly : " " " " " AROUND THEM
- strech : lines strech to take up the remain space

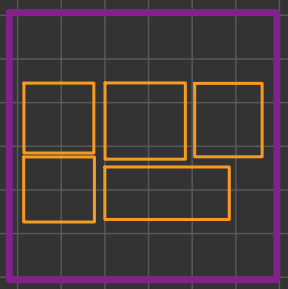
flex-start



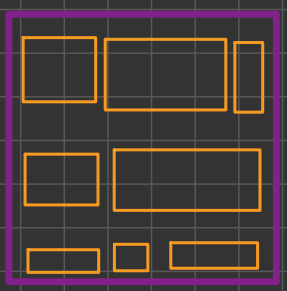
flex-end



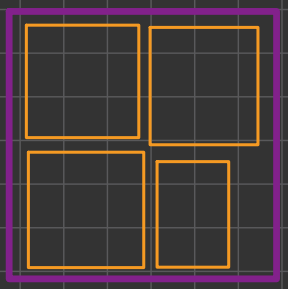
center



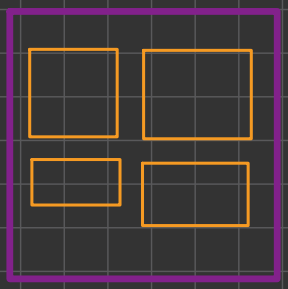
space between



strech



space around

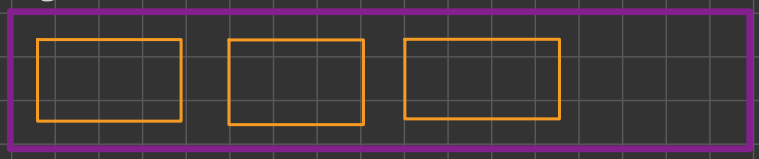


```
.container {
  align-content:
  flex-start + safe/ unsafe;
}
```

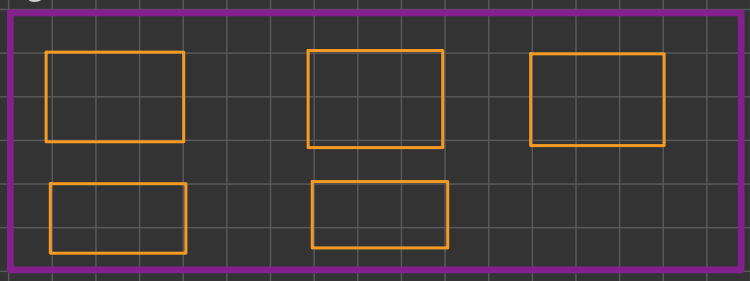
8. Gap, row-gap, column-gap:

the gap property explicity control the space between Flex items. it applies only between items!

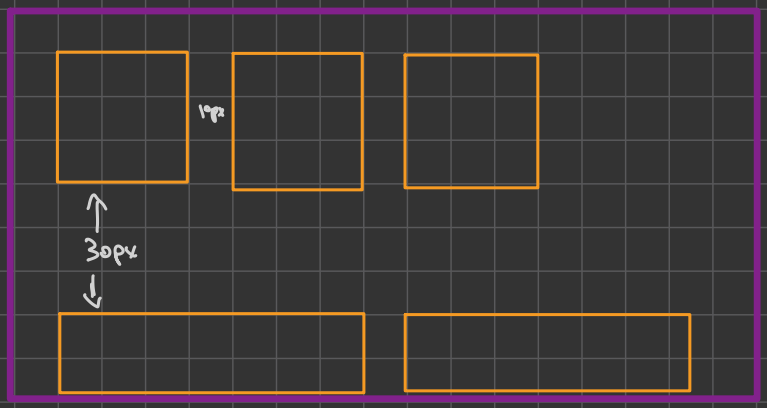
gap: 10px



gap: 30px



gap: 10px 30px



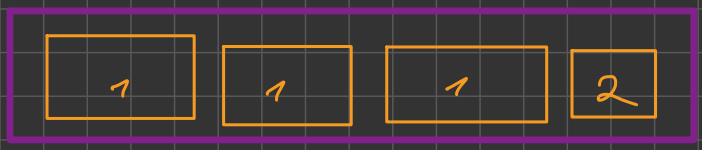

```
container {
  display: Flex;
  gap: 10px;
  gap: 10px 20px; /* row col */
  row-gap: 10px;
  column-gap: 20px;
```

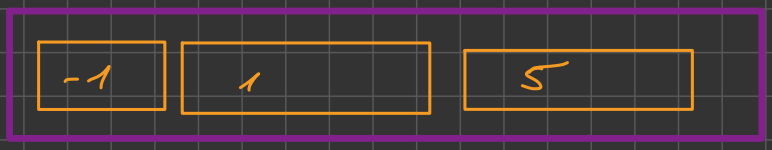
the behavior could be thought of as a minimum gutter, as if the gutter is bigger somehow then the gap will only take affect if the space would end up smaller.

it is not exclusively for flex-box!

Properties for the children - flex items

1. **order**: by default flex items are laid out in the source ordered. however, the order property controls the order.






```
.item {
order: 5; /* default is 0 */
}
```

---

2. **Flex-grow**: this define the ability for flex item to grow if necessary. it accepts unitless value that serves as a proportion. if dictates what amount of available space inside the flex container the item should take up.

```
.item{
flex-grow: 4;
/* default 0 */
}
```

if all items flex-grow set to 1, the remaining space in the container will be distributed equally to all children.
if one of the children have value of two. the child will take twice as much of the space either one of the others.

* negative numbers are invaild.

---

3. **Flex-Shrink**: this define the abillity of the items to shrink if necessary.

```
/* default is 1 */
.item {
flex shrink: 3; }
```

4. **Flex basis**: this defines the defult size of an element before the remaining space is distributed. it can be length (20%, 5rem, etc) or a keyword. the "auto" means- look for my width or height property. "content" means- sized it based on the item content.

---

5. **Flex**: this is the shorthand for flex-grow, shrink and basis. the second and the third are optionall. the defult is: 0, 1, auto) if sets with singel number valhe like: flex: 5; that changes the flex-basis to 0%, so its like grow: 5, shrink: 1, basis: 0%.

```
.item {
flex: none | [ <'flex-grow'> <'flex-shrink'>
? || <'flex-basis'> ]
```

---

6. **align-self**: this allows the defult alignment to be overriden for individuals flex items.

---

very simple example. solving an almost daily problem: perfect-centering!

```css
.parent {
  display: flex;
  height: 300px; /* Or whatever */
}

.child {
  width: 100px;  /* Or whatever */
  height: 100px; /* Or whatever */
  margin: auto;  /* Magic! */
}
```

Prefixing Flexbox

Flex box requires some vendor-prefixing to support the most browsers possible there are actully different properties and value names (changes over time) creating "old", "tweener" and "new" versions.

the best way to handle it is to writ in the new syntax and run your CSS through "Autoprefixer".

@mixin Sass - some of the prefixing:

```scss
@mixin flexbox() {
  display: -webkit-box;
  display: -moz-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
}

@mixin flex($values) {
  -webkit-box-flex: $values;
  -moz-box-flex:  $values;
  -webkit-flex:  $values;
  -ms-flex:  $values;
  flex:  $values;
}

@mixin order($val) {
  -webkit-box-ordinal-group: $val;
  -moz-box-ordinal-group: $val;
  -ms-flex-order: $val;
  -webkit-order: $val;
  order: $val;
}

.wrapper {
  @include flexbox();
}

.item {
  @include flex(1 200px);
  @include order(2);
}
```