Chapter 4
CSS Layout with Flexbox and Grid

INT150 Web Technologies (2/2024)

Sanit Sirisawatvatana

CSS page layout

- Two important CSS page Layout tools
    - Flex for greater control over arranging items along one axis
    - Grid for honest-to-goodness grid based layout
- Each tool has it special purpose, but you can use them together to achieve layouts.

Flexbox

- Flexbox
    - is a display mode that lays out element along one axis (horizontal or vertical).
    - is useful for menu options, galleries, product listings, etc.
- Flexbox can be used for individual components on a page or the whole page layout.

Flexbox can do

- Allow items to stretch, shrink, and/or wrap onto multiple lines, making it a great tool for responsive layouts

- Make all neighboring items the same height

- Easy horizontal and vertical centering

- Change the order in which items display

According to the spec,

The defining aspect of flex layout is the ability to make the flex items "flex," altering their width/height to fill the available space in the main dimension.

Flexbox container

- To turn on Flexbox layout

  display : flex (or inline-flex)

- This makes the element a flexbox container.

- All of its direct children become flex items in that container.

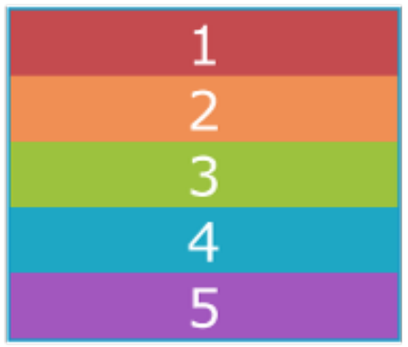
- The flex items are laid out and aligned along flex lines (the string).

- By default, items line up in the writing direction of the document (left to right rows in left-to-right reading languages).

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Flexbox container

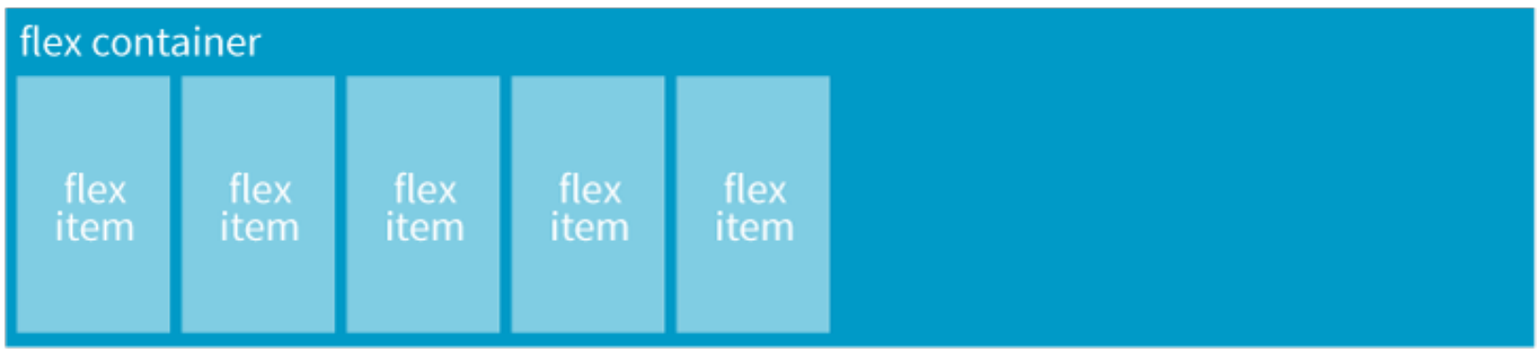
By default, the **divs** display as block elements, stacking up vertically. Turning on flexbox mode makes them line up in a row.



block layout mode

```
display: flex;
```

flexbox layout mode



Controlling the "Flow" within the container

- Once an element becomes a flex container, you can set a few properties on that container to control how items flow within it.

- The flow refers to the direction in which flex items are laid out as well as whether they are permitted to wrap onto additional lines.

Flow direction

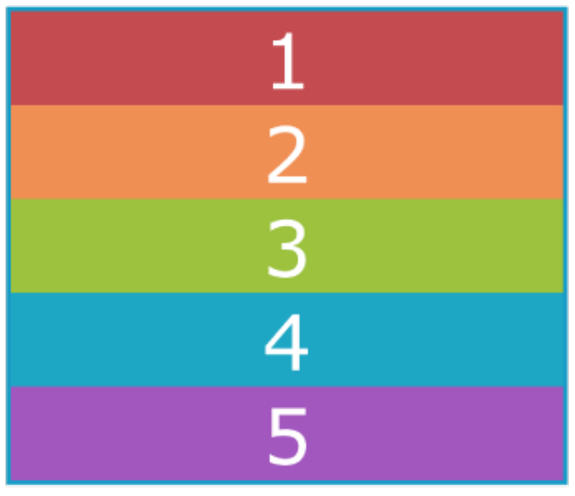
flex-direction

Values:	row | column | row-reverse | column-reverse
Default:	row
Applies to:	flex containers
Inherits:	no

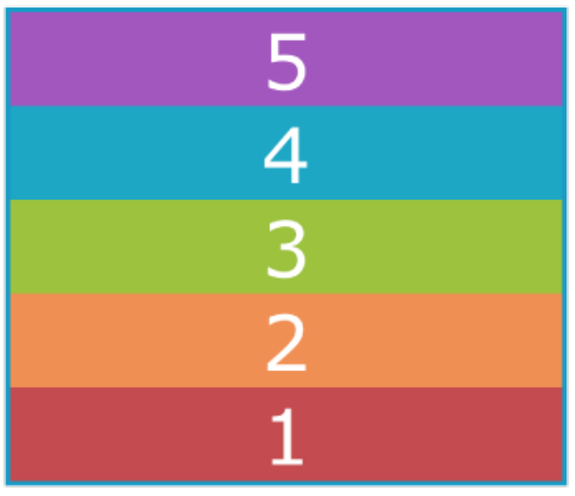
flex-direction: row; (default)

flex-direction: row-reverse;

flex-direction: column;



flex-direction: column-reverse;



Flexbox axes

- Flexbox is "direction-agnostic", there are no references to "left", "right", "top, or "bottom".

- In Flexbox system, there are two axes:

    - Main axis runs in whatever direction the flow has been set.
    - Cross axis runs perpendicular to the main axis.

- Both main and cross axes have a start and an end, based on the direction in which the items flow.

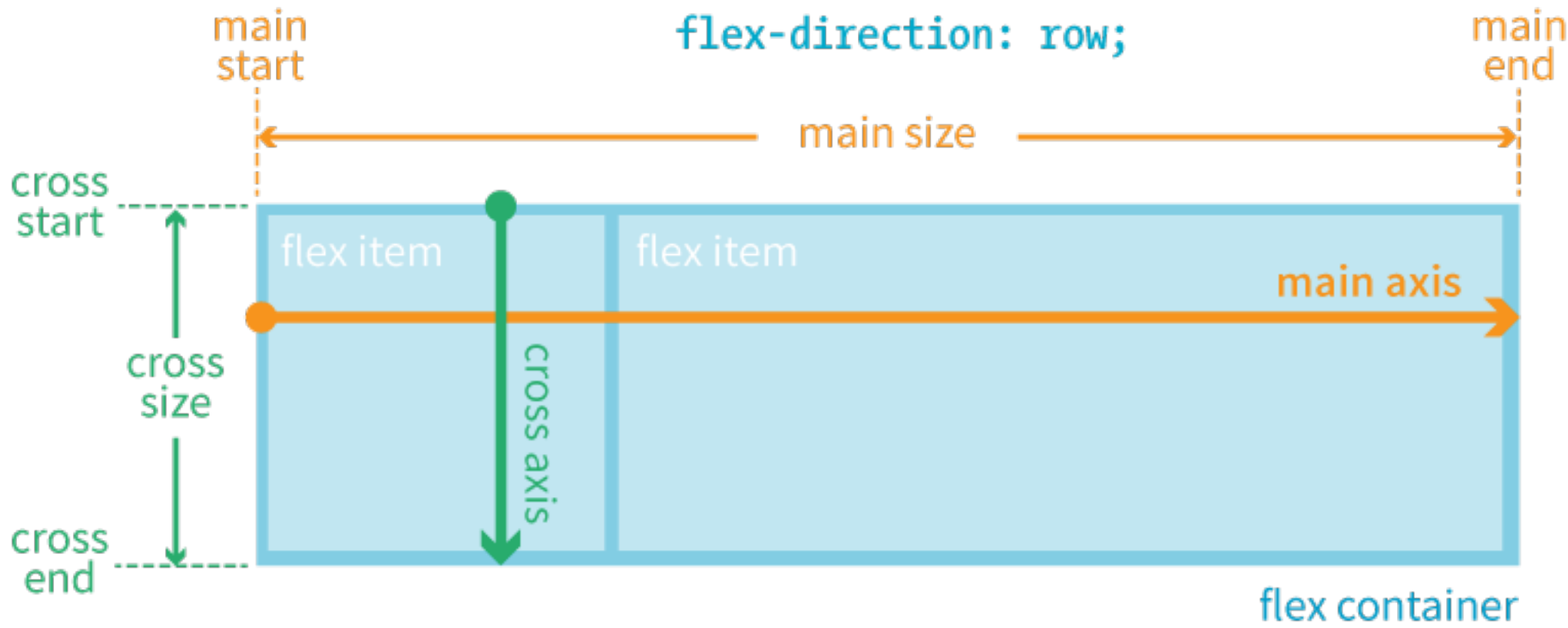
- The main size is the width (or height it it's column) of container along the main axis, and the cross size is height (or width if it's a column) along the cross axis.

direction-agnostic

flex-direction : row

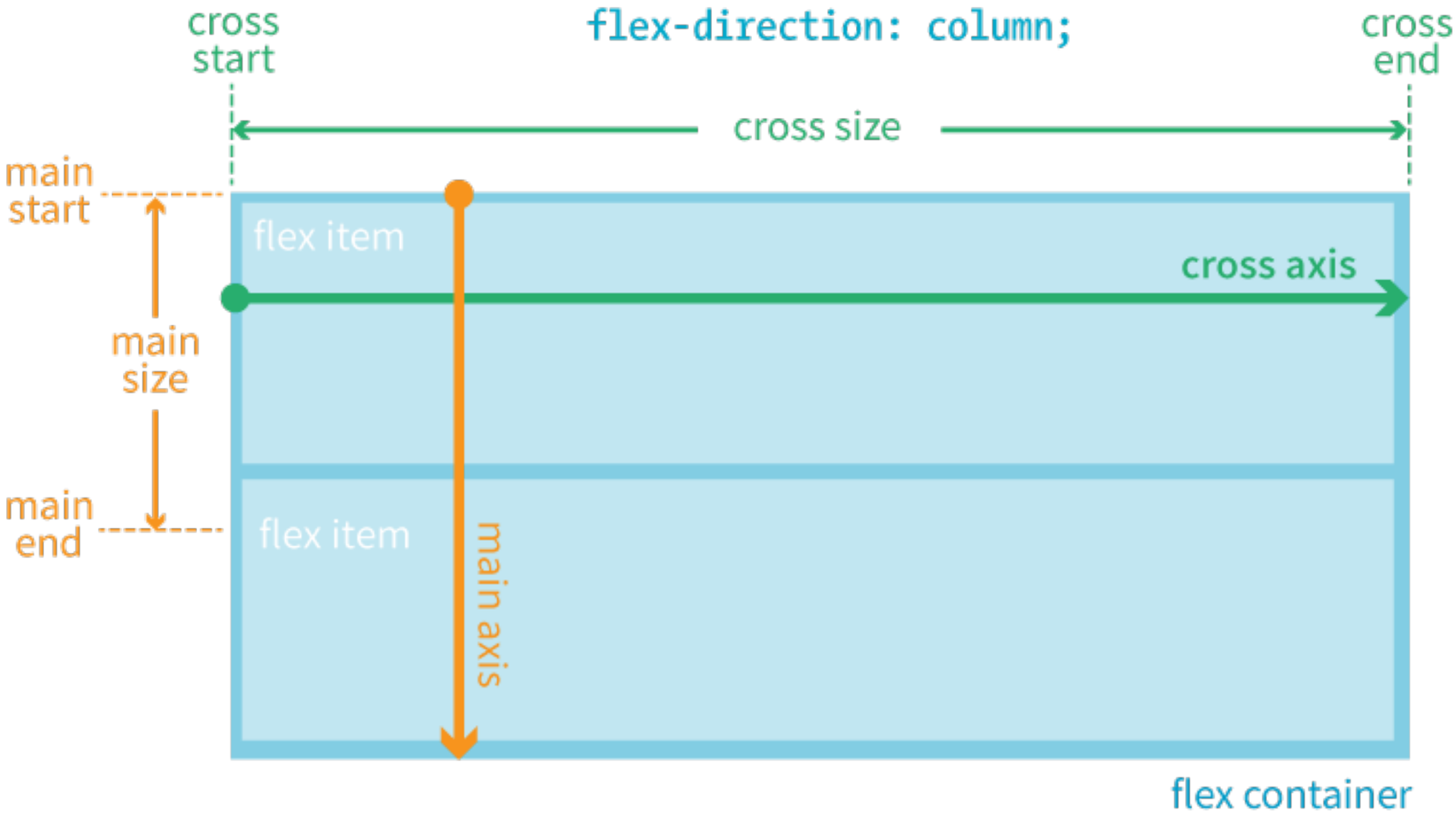
FOR LANGUAGES THAT READ HORIZONTALLY FROM LEFT TO RIGHT:

When `flex-direction` is set to `row`, the main axis is horizontal and the cross axis is vertical.



flex-direction : column

When **flex-direction** is set to **column**, the main axis is vertical and the cross axis is horizontal.



Wrapping items onto multiple lines

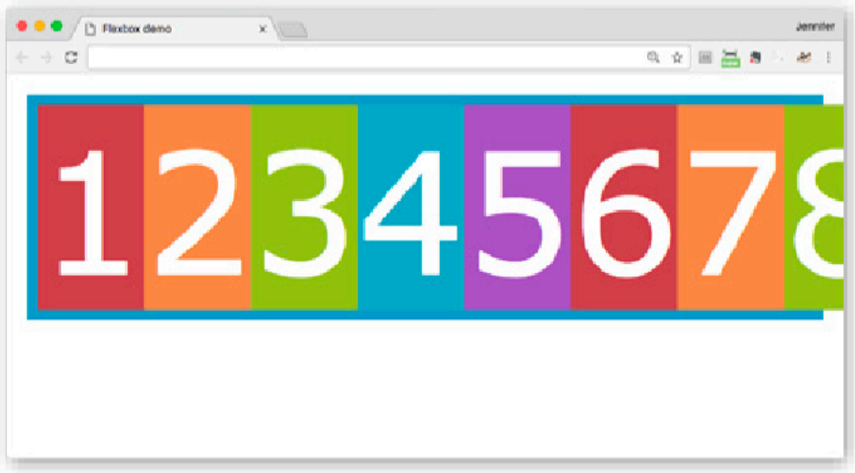
- Flex items line up one axis, but you can allow that axis to wrap onto multiple lines with the flex-wrap property.

flex-wrap

Values:	nowrap | wrap | wrap-reverse
Default:	nowrap
Applies to:	flex containers
Inherits:	no

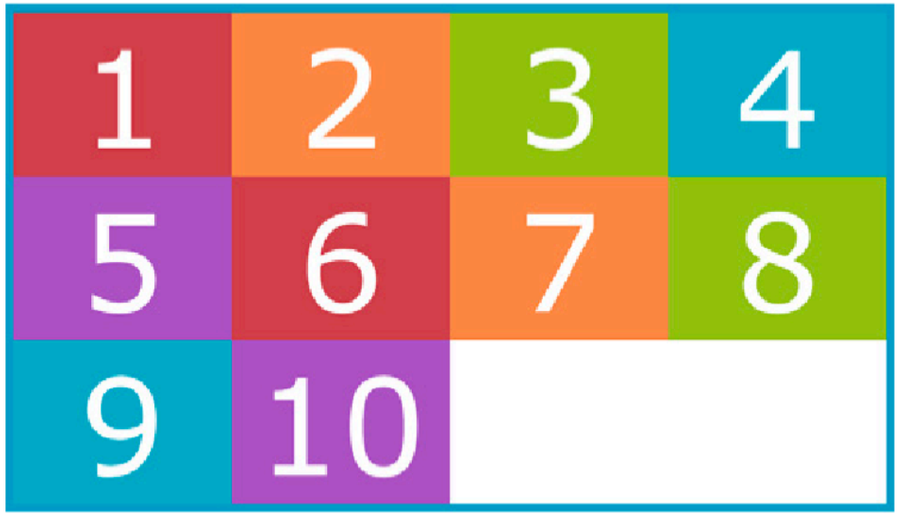
flex-wrap (row direction)

`flex-wrap: nowrap;` (default)

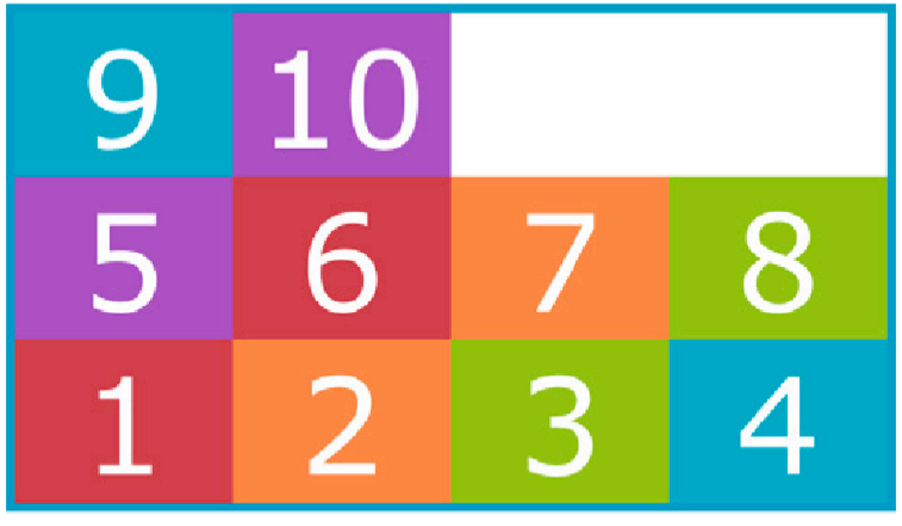


When wrapping is disabled, flex items squish if there is not enough room, and if they can't squish any further, may get cut off if there is not enough room in the viewport.

`flex-wrap: wrap;`

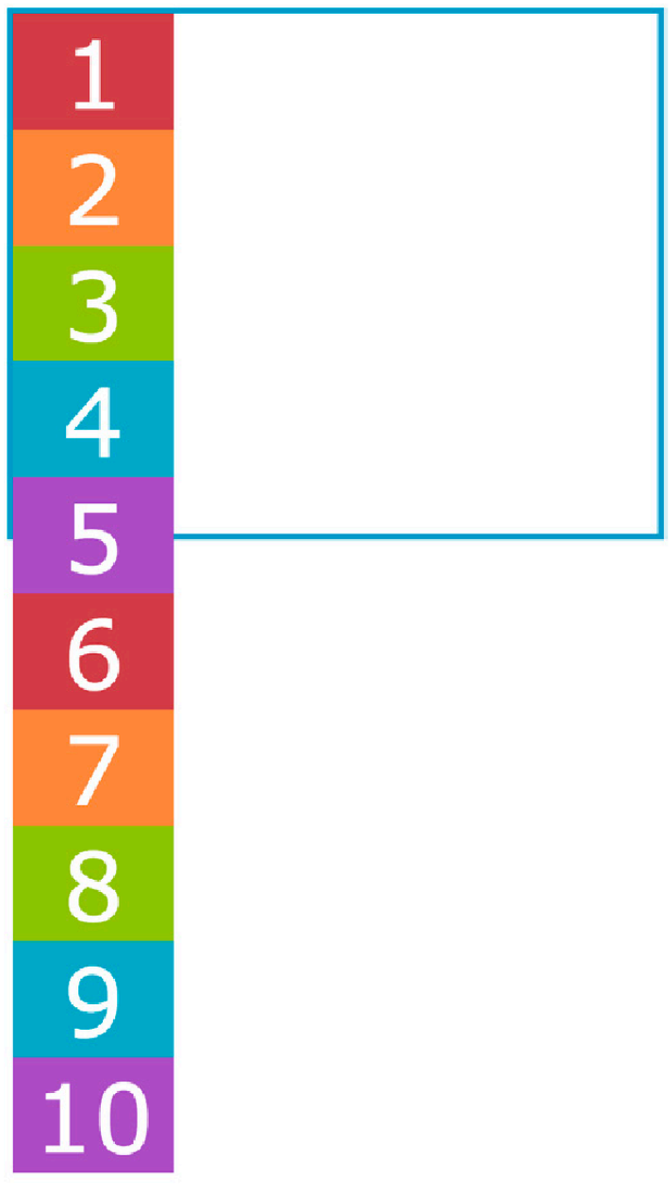


`flex-wrap: wrap-reverse;`

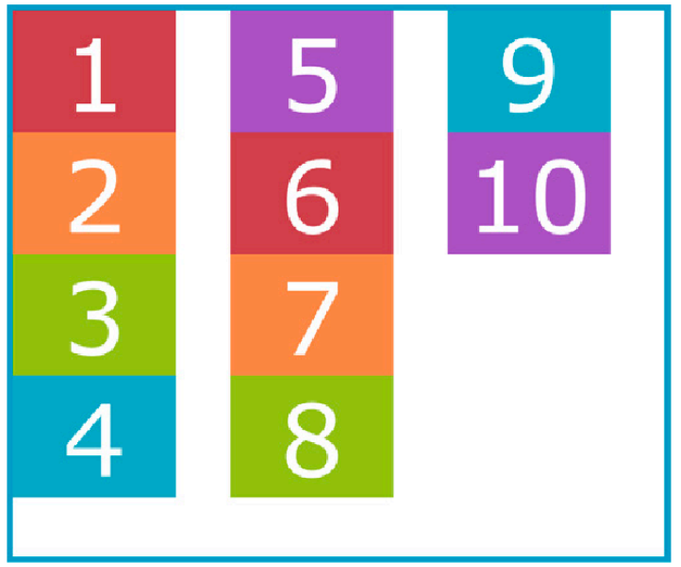


flex-wrap (column direction)

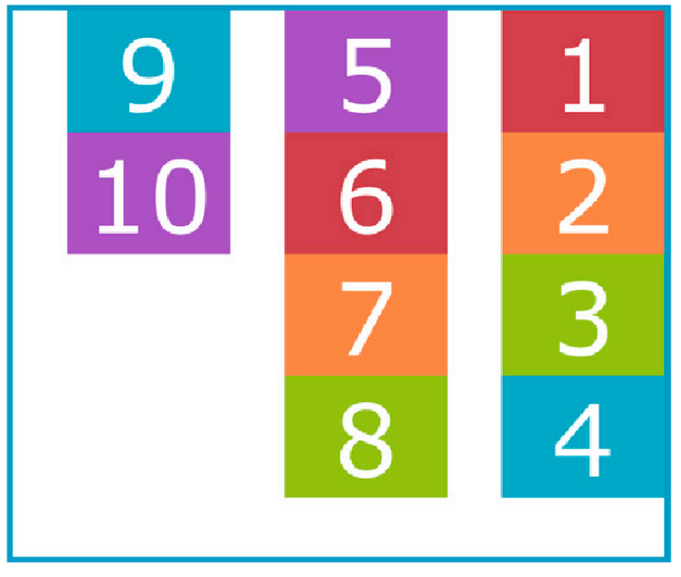
`flex-wrap: nowrap;` (default)



`flex-wrap: wrap;`



`flex-wrap: wrap-reverse;`



flex-flow (shortcut)

flex-flow

Values:	flex-direction flex-wrap
Default:	row nowrap
Applies to:	flex containers
Inherits:	no

Controlling the alignment of flex items

- Alignment of items along
  - the main axis (justify-content)
  - the cross axis (align-items and align-content)

Aligning on the main axis:

justify-content

justify-content

Values:	flex-start | flex-end | center | space-between | space-around
Default:	flex-start
Applies to:	flex containers
Inherits:	no

- flex-start and flex-end position the line of items towards the start and end of the main axis, respectively, and center centers them.

- space-between -- the first item is positioned at the start point, the last goes at the end point, and the remaining space is distributed evenly between the remaining items.

- space-around -- adds an equal amount of space on the left and right side of each item, resulting in a doubling up of space between neighboring items.

justify-content (row direction)

`justify-content: flex-start;` (default)

`justify-content: flex-end;`

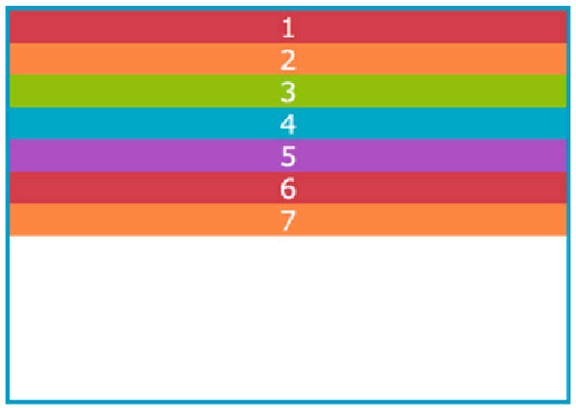
`justify-content: center;`

`justify-content: space-between;`

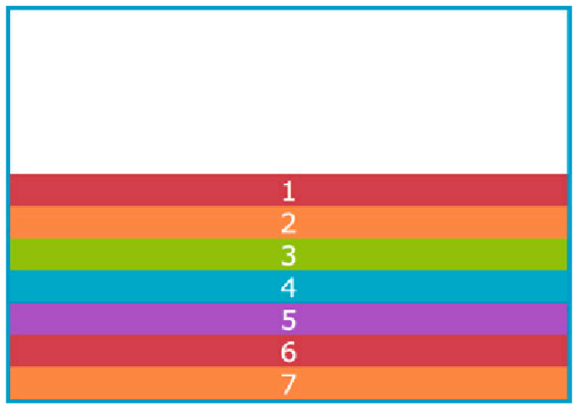
`justify-content: space-around;`

justify-content (column direction)

`justify-content: flex-start;` (default)

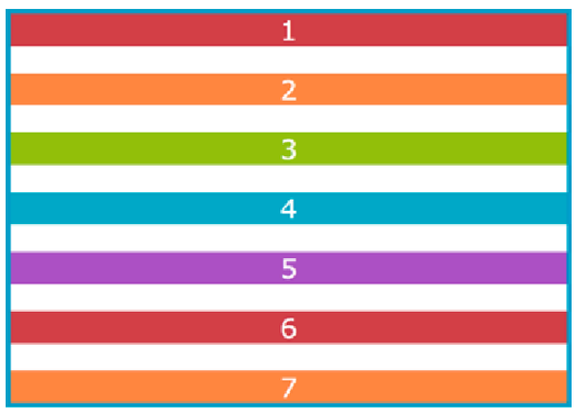


`justify-content: flex-end;`



`justify-content: center;`

`justify-content: space-between;`



`justify-content: space-around;`

Aligning on the cross axis:

align-items

align-items

Values:	flex-start | flex-end | center | baseline | stretch
Default:	stretch
Applies to:	flex containers
Inherits:	no

- baseline -- aligns the baselines of the first lines of text, regardless of their size. (a good option for lining up elements with different text sizes such as headlines and paragraphs across multiple items)

- stretch -- causes items to stretch util they fill the cross axis.

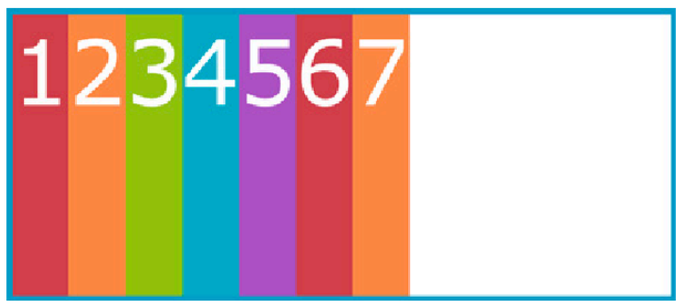
align-items (row direction)

`align-items: flex-start;`

`align-items: flex-end;`

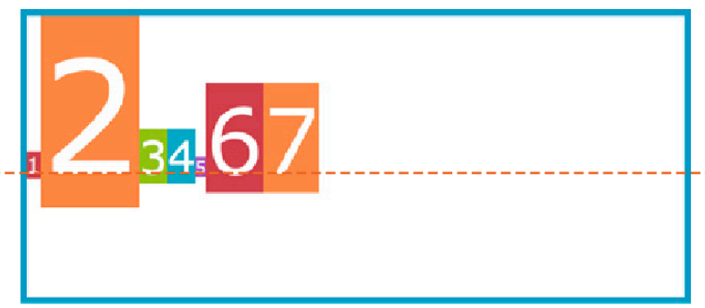
`align-items: center;`

`align-items: stretch;` (default)



Items are aligned so that the baselines of the first text lines align.

`align-items: baseline;`



align-self

align-self

Values:	flex-start | flex-end | center | baseline | stretch
Default:	stretch
Applies to:	flex items
Inherits:	no

```
.box4 {
  align-self: flex-end;
}
```

align-self: flex-end;

Aligning multiple lines :
align-content

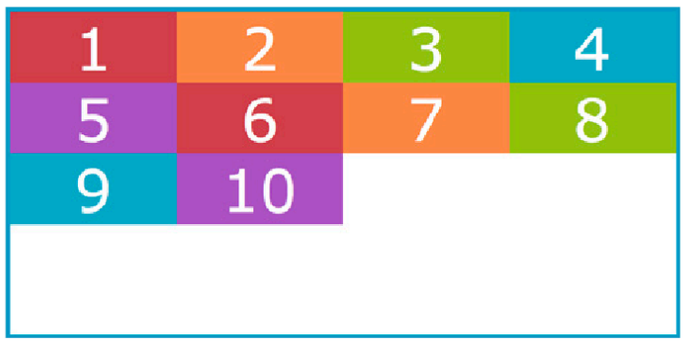
align-content

Values:	flex-start | flex-end | center | space-around | space-betwen | stretch
Default:	stretch
Applies to:	multi-line flex containers
Inherits:	no

- align-content -- affects how multiple flex lines are spread out across the cross axis.
    - This property applies only when flex-wrap is set to wrap or wrap-reverse and there are multiple lines to align.
    - If the items are on a single line, it does nothing.

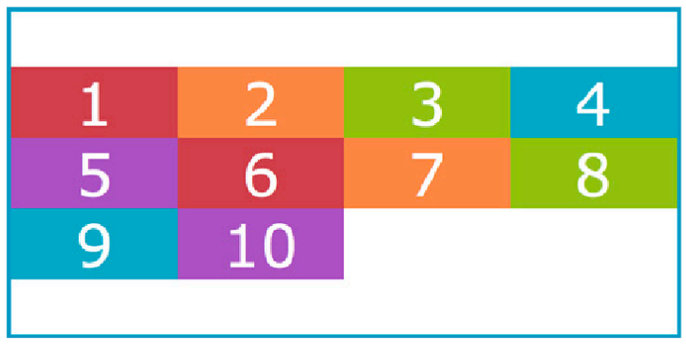
align-content

`align-content: flex-start;`

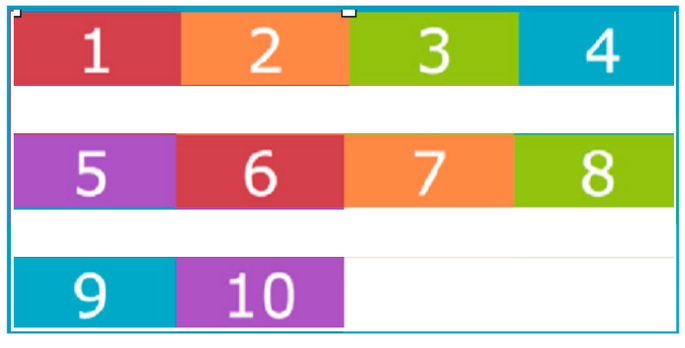


`align-content: flex-end;`

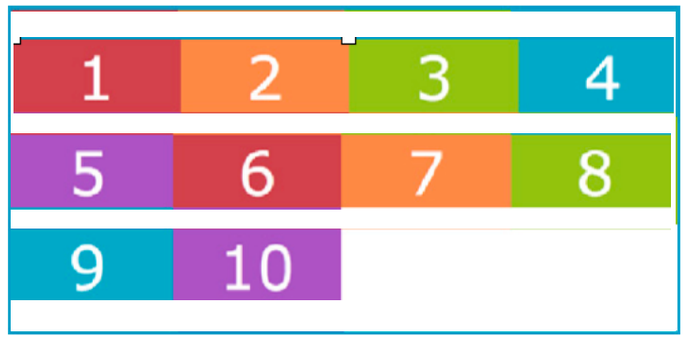
`align-content: center;`



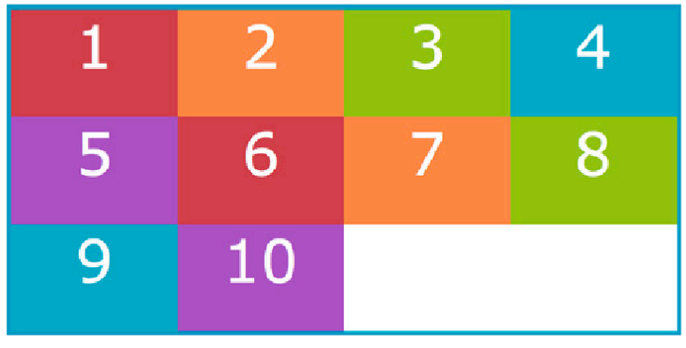
`align-content: space-between;`



`align-content: space-around;`



`align-content: stretch;` (default)



Align items with margins

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>HTMLiveCode</title>
        <style>
            ul {
                display: flex ;
                align-items: center ;
                background-color: #00af8f ;
                list-style: none ;
                padding: 0.5em ;
                margin: 0 ;
            }
            li {
                margin: 0 1em ;
            }
            li.logo {
                margin-right:  auto ;
            }
        </style>
    </head>
    <body>
        <ul>
            <li class="logo">Logo</li>
            <li>About</li>
            <li>Blog</li>
            <li>Shop</li>
            <li>Contact</li>
        </ul>
    </body>
</html>
```

Use margins to add space on the sides of particular flex items

When you use margin: auto on a flex item, the justify-content property no longer has a visual effect because you've manually assigned a location for the extra space on the main axis.

Determining how items "flex(ible)" in the container

flex

Values:	none | 'flex-grow flex-shrink flex-basic'
Default:	0 1 auto
Applies to:	flex items (not flex container)
Inherits:	no

- Items can resize (flexible) to fill the available space on the main axis in the container.
- The flex property identifies how much an item can grow and shrink and identified a starting size.
- For the flex-grow and flex-shrink properties, the values 1 and 0 (like on/off switch), where:
  - 1 "turn on" or allows an item to grow or shrink
  - 0 "turn off" or prevents an item from growing or shrinking
- The flex-basic property sets the starting dimensions, either to a specific size or a size base on the contents

Expanding items (flex-grow)

flex-grow

Values:	number
Default:	0
Applies to:	flex items (not flex container)
Inherits:	no

- The flex-grow property specifies whether (an in what proportion) an item may stretch larger.

- By default is 0, which means an item is not permitted to grow wider than the size of its content or its specified width.

flex-grow

- Because items do not expand by default, the alignment properties have the opportunity to go in effect.

- If the extra space was taken up inside items, alignment would not work.

- If flex-grow is set to 1 for all the items in a container, the browser takes whatever extra space is available along the main axis and applies it equally to each item, allowing them all to stretch the same amount.

flex-grow

THE MARKUP

```
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
```

THE STYLES

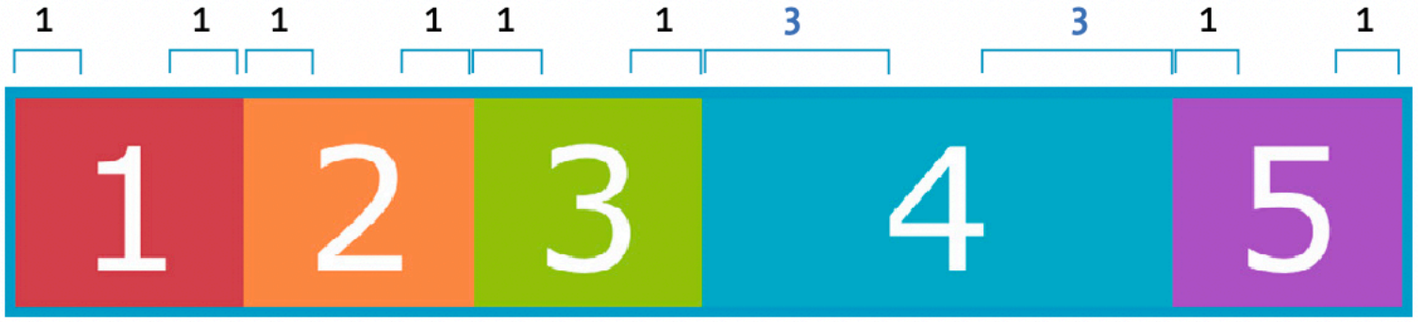
```
.box {
  …
  flex: 1 1 auto;
}
```

flex: 0 1 auto; (prevents expansion)

flex: 1 1 auto; (allows expansion)

flex-grow greater than 1 (as ratio)

```
.box4 {
   flex: 3 1 auto;
}
```



flex: 3 1 auto;

- If set a higher flex-grow integer to an item, it act as a ratio that applies more space within that item.
- The flex-grow of box4 is 3, the box4 receives three times the amount of space than the remaining items (flex-grow is 1).
- The resulting item is not three times as wide as the others (only got three times the amount of space added to it).

Squishing items (flex-shrink)

flex-shrink

Values:	number
Default:	1
Applies to:	flex items (not flex container)
Inherits:	no

- By default, the flex-shrink value is set to 1, which means if do nothing, items shrink to fit at the same rate (when the container is not wide enough).
- When flex-shrink is 0, items are not permitted to shrink, and they may hang out of their container and out of view of the viewport.
- Finally, as in flex-grow, a higher integer works as ratio.
- An item with a flex-shrink of 2 will shrink twice as fast as if it were set to 1.

flex-shrink

- Flex item stop shrinking when they reach their minimum size (defined by min-width/min-height).

- By default (when min-width/min-height is auto), this minimum is based on its min-content size.

- But it can be set to zero, or 12em, or any other length that seems useful.

Providing an initial size (flex-basic)

- The flex-basic defines the starting size of item before any wrapping, growing, or shrinking occurs.

- It may be used instead of the width property (or height property for columns) for flex items.

(Flex setting override specified widths/heights for flex items)

Providing an initial size (flex-basic)

flex-basic

Values:	length | percentage | content | auto
Default:	auto
Applies to:	flex items (not flex container)
Inherits:	no

- auto -- use the specified width/height property value for the item size.

- If the item's main size property (width/height) is not set or is auto (its default), flex-basic use the content width.

- content -- explicitly set to the width of the content.

flex-basic

```
box {
  flex: 0 1 100px;
}
```

flex: 0 1 100px;

When the container is wide, the items will not grow wider than their **flex-basis** of 100 pixels because **flex-grow** is set to 0.

When the container is narrow, the items are allowed to shrink to fit (**flex-shrink: 1**).

Example : flex

```
box {
    width: 100px ;
    flex: 1 0 auto ;
}
```

- The flex-size for the box is set to 100 pixels because the auto value use the value set by the width.

- Items are allowed to grow (flex-grow : 1), taking up any extra space in the container, but they are not allowed to shrink (flex-shrink : 0).

Handy shortcut flex values

- flex : initial ; (same as flex : 0 1 auto ;)
    - Prevent the item from growing, but allows it to shrink to fit the container
- flex : auto ; (same as flex : 1 1 auto ;)
    - Allow items to fully flexible as needed. Size is based on the width/height properties.
- flex : none ; (same as flex : 0 0 auto ;)
    - Creates a completely inflexible item while sizing it to the width/height properties.
- flex : integer ; (same as flex : integer 1 0px ;)
    - The flex-basic of 0, which means it has absolute flex and free space is allocated according to the flex number applied to items.

Absolut vs. Relative Flex

- The flex-grow higher than 1 (extra space) is assigned to items based on their flex ratio.

- This is called relative flex, and it is how extra space is handled whenever the flex-basis is set to any size other than zero (0), such as particular width/height value or auto.

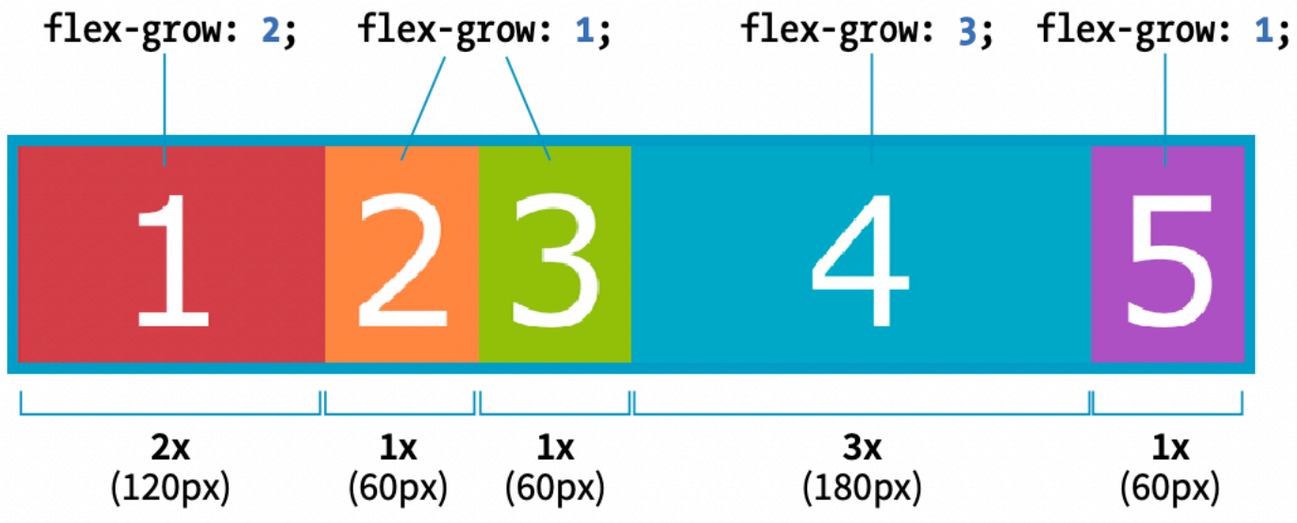
- When flex-basic is 0, the items get sized proportionally according to the flex ratio, which is known as absolute flex (flex-size: 0).

- An item with a flex-grow value of 2 would be twice as wide as items set to 1.

Absolute flex

- In absolute flex, boxes are sized according to the flex value ratios.

```
.box {
  /* applied to all boxes */
  flex: 1 0 0%;
}
.box1 {
  flex: 2; /* shortcut value for flex: 2 1 0px */
}
.box4 {
  flex: 3; /* shortcut value for flex: 3 1 0px */
}
```



Changing the Order of Flex item

order

Values:	integer
Default:	0
Applies to:	flex items and absolutely positioned children of flex containers
Inherits:	no

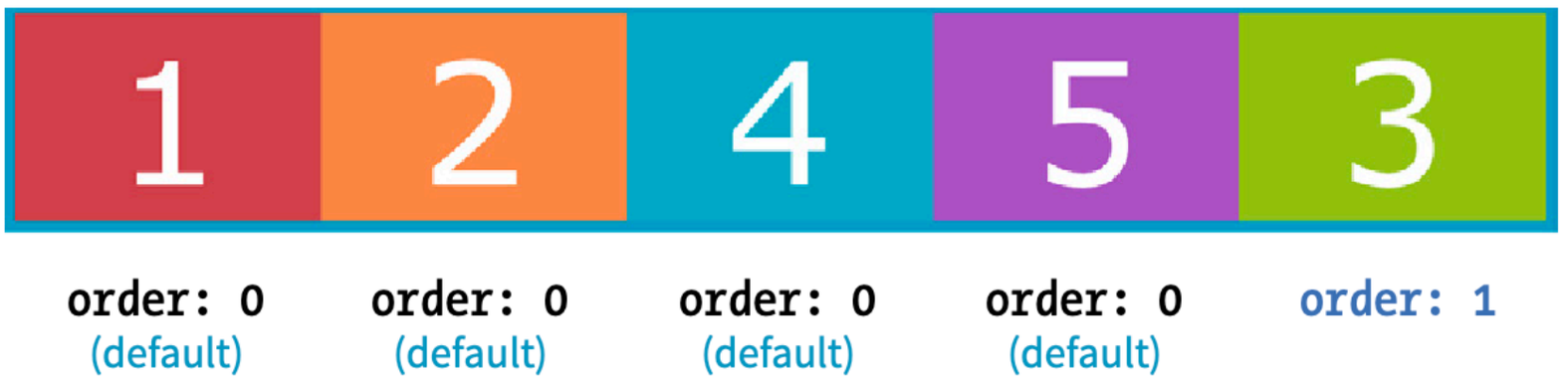
- Specifies the order in which a particular item should appear in the flow (independent of the HTML source order)

- The default is 0. Items with the same order value are place according to their order in the source.

- Item with different order values are arranged from lowest to highest.

order

```
.box3 {
  order: 1;
}
```



Ordinal groups

- Ordinal groups
  - Items that share the same order value are called an ordinal group.
  - Ordinal groups stick together and are arranged from lowest value to highest.

```
.box2, .box3 {
  order: 1
}
```

order with negative values

```
.box5 {
  order: -1
}
```