CSS LAYOUT WITH FLEXBOX

About flexbox

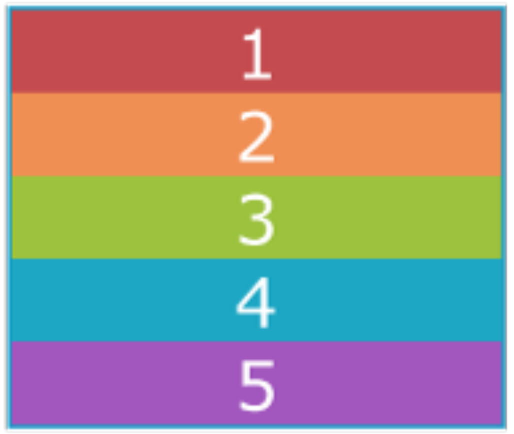
- **Flexbox** is a display mode that lays out elements along one axis (horizontal or vertical) – it works in 1 dimension only.
- Features flex-containers (parent) and flex-items (children). Both are required to work.
- Items in a flexbox can expand, shrink, and/or wrap onto multiple lines, making it a great tool for responsive layouts.
- Excels at vertical centering and equal heights.
- Items can easily be reordered, so they aren't tied to the source order.
- Useful for individual components on a page like menu options, galleries, product listings, etc.
- Although it isn’t made for whole page layouts, in the real world whole web pages are being laid out with flexbox.

Flexbox container

`display: flex`

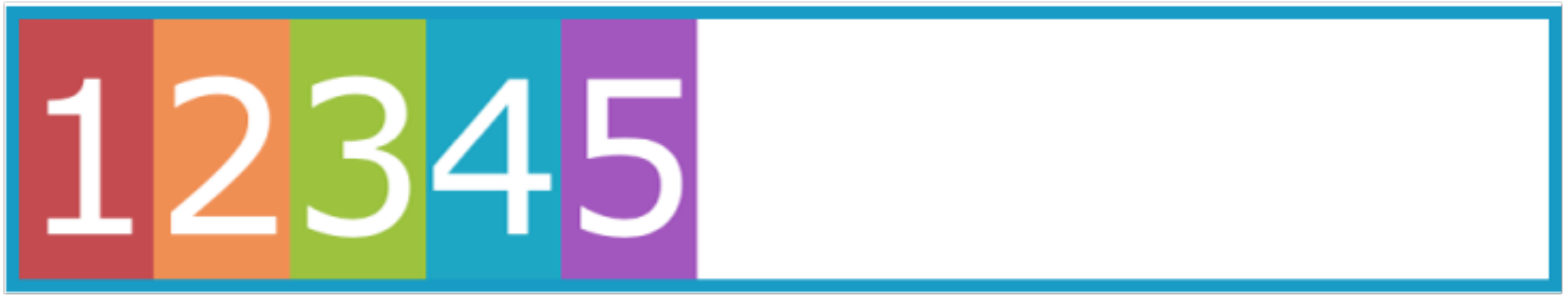
- To turn on flexbox mode, set the parent element’s **`display`** to **`flex`**.
- This makes the element a **flexbox container**.
- All of its direct children become **flex items** in that container.
- By default, items line up in the writing direction of the document (left to right rows in "ltr” reading languages).

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

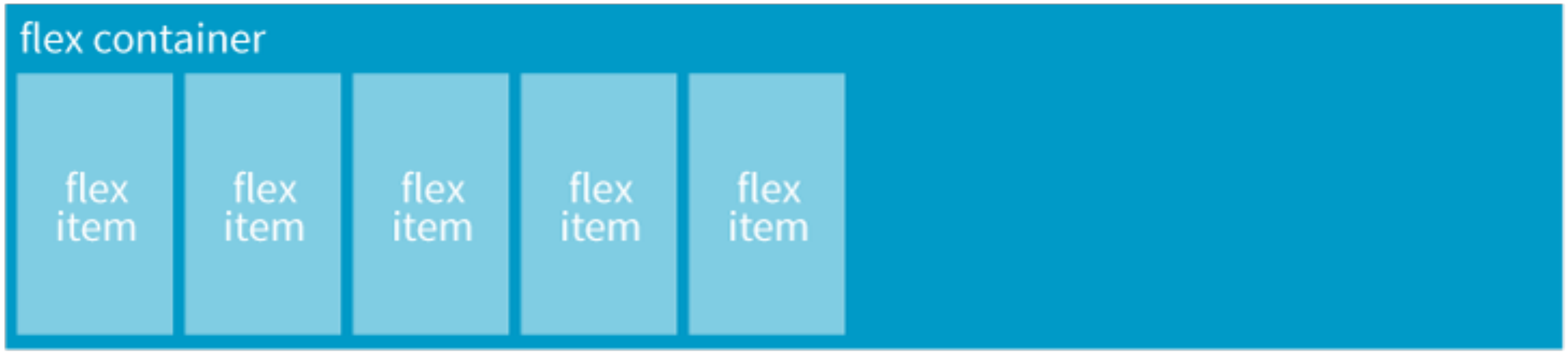


block layout mode

```
display: flex;
```



flexbox layout mode

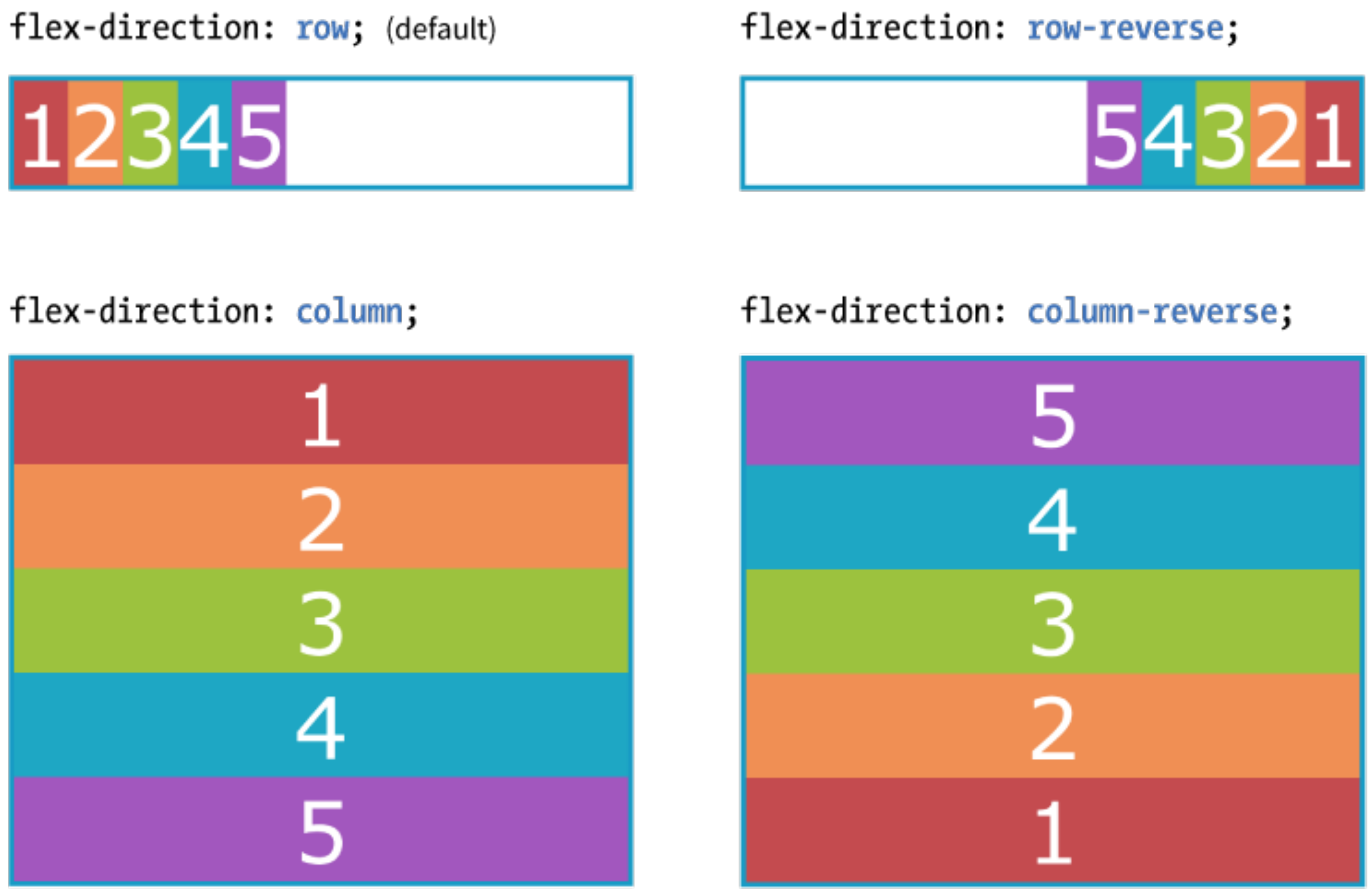


Rows and columns (direction)

`flex-direction`

Values: `row`, `column`, `row-reverse`, `column-reverse`

The default value is **`row`** (for ltr languages), but you can change the direction so items flow in columns or in reverse order:


flex-direction: row; (default)
12345
flex-direction: row-reverse;
54321
flex-direction: column;
1 2 3 4 5
flex-direction: column-reverse;
5 4 3 2 1

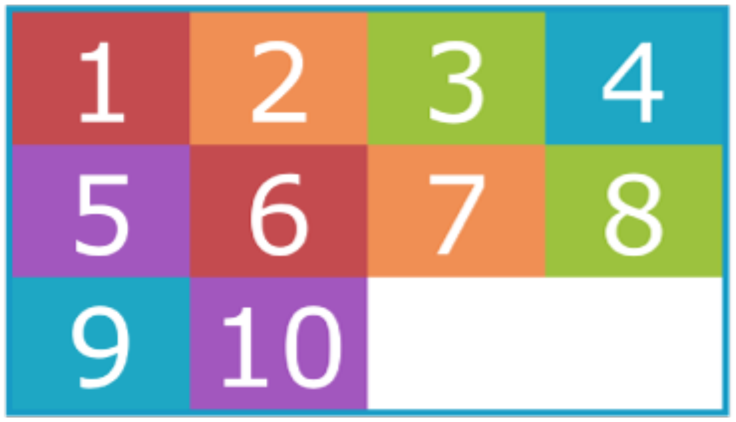
Wrapping flex lines

`flex-wrap`

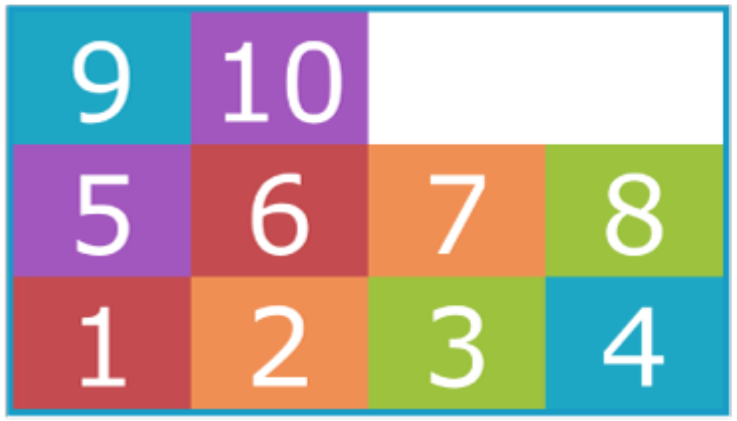
Values: `wrap`, `nowrap`, `wrap-reverse`

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

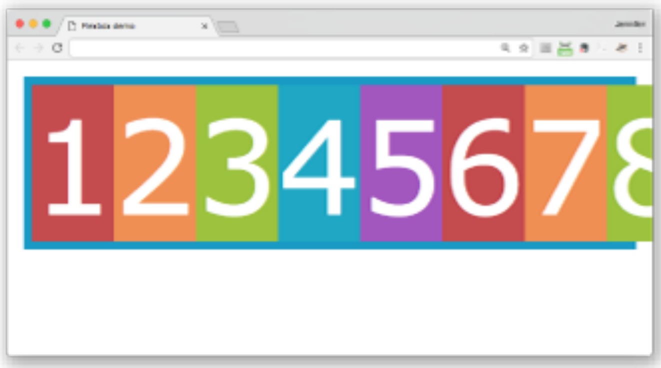
`flex-wrap: wrap;`



`flex-wrap: wrap-reverse;`



`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 flow (direction + wrap)

`flex-flow`

Values: *flex-direction flex-flow*

The shorthand **`flex-flow`** property specifies both direction and wrap in one declaration.

Example

```
#container {
  display: flex;
  height: 350px;
  flex-flow: column wrap;
}
```

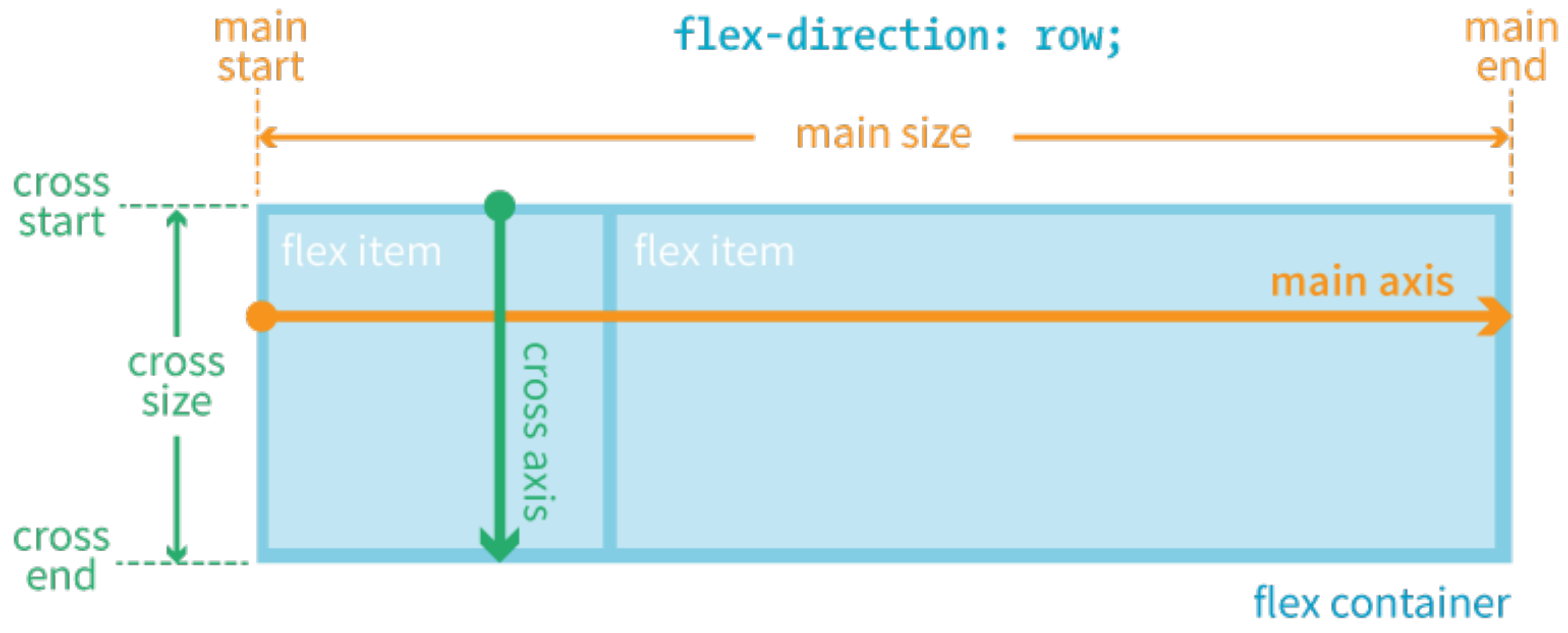
Flexbox alignment terminology

- Flexbox is “direction-agnostic,” so we talk in terms of *main axis* and *cross axis* instead of rows and columns.
- The **main axis** runs in whatever direction the flow has been set.
- The **cross axis** runs perpendicular to the main axis.
- Both axes have a **start**, **end**, and **size**.

ROW: Main and cross axis

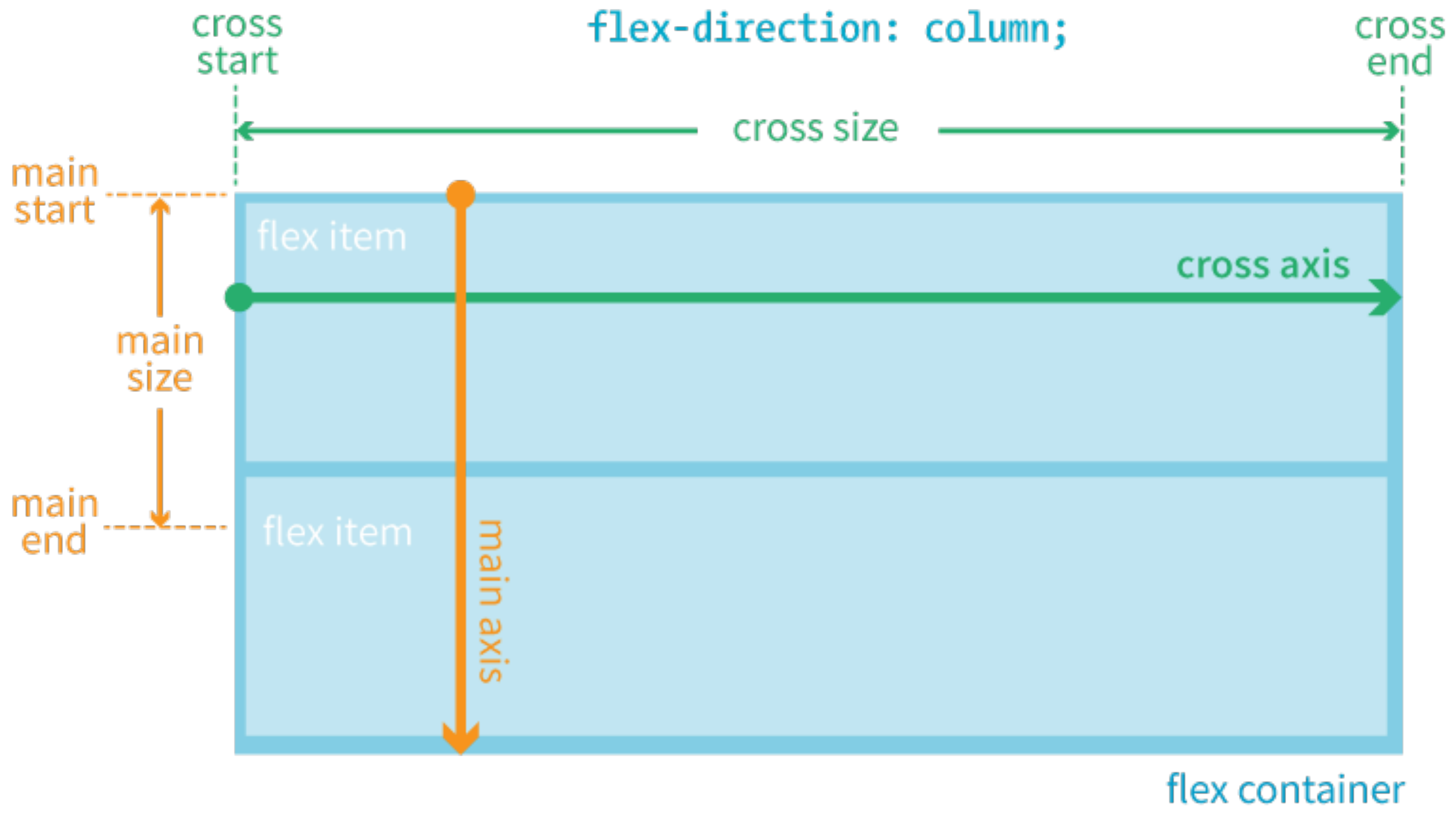
FOR LANGUAGES THAT READ HORIZONTALLY FROM LEFT TO RIGHT:

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



COLUMN: Main and cross axis

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



Aligning on the main axis

`justify-content`

Values: `flex-start`, `flex-end`, `center`, `space-between`, `space-around, space-evenly`

When there is space left over on the **main axis**, you can specify how the items align with the **`justify-content`** property (notice we say *start* and *end* instead of left/right or top/bottom).

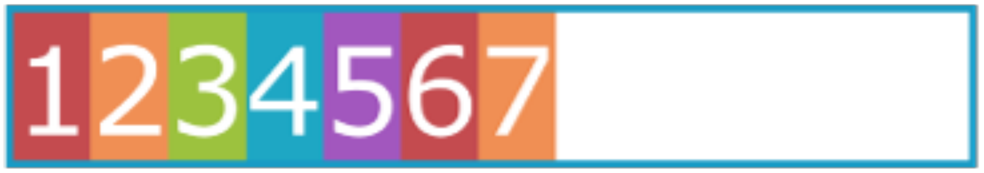
The **`justify-content`** property applies to the **flex container**.

Example:

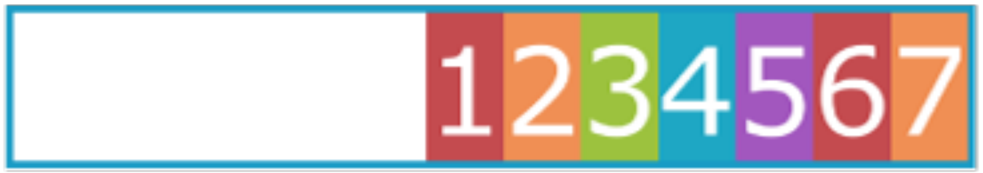
```
#container {
  display: flex;
  justify-content: flex-start;
}
```

When the direction is **row**, and the **main axis is horizontal**

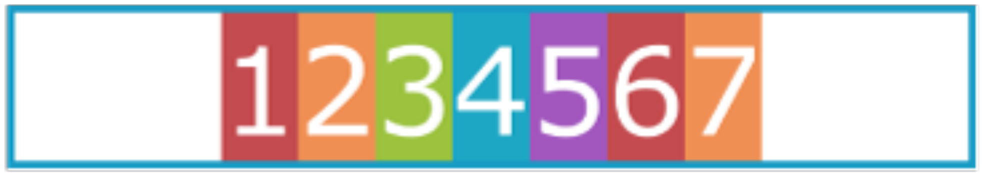
`justify-content: flex-start;` (default)



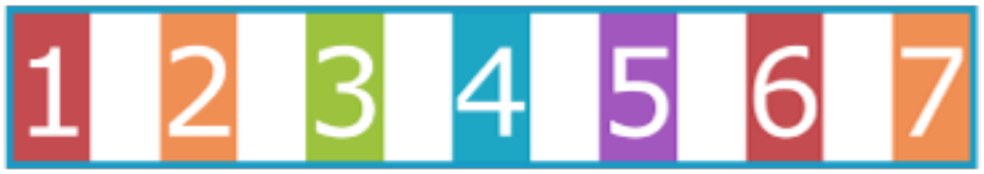
`justify-content: flex-end;`



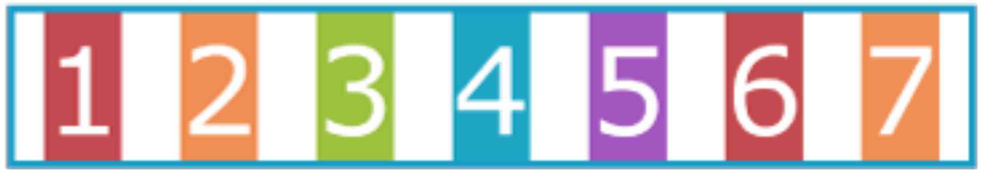
`justify-content: center;`



`justify-content: space-between;`

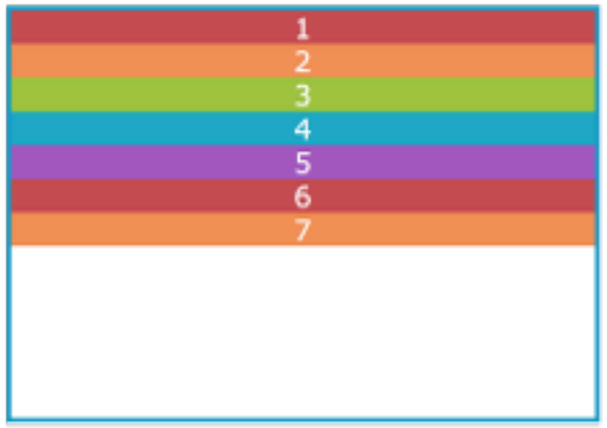


`justify-content: space-around;`

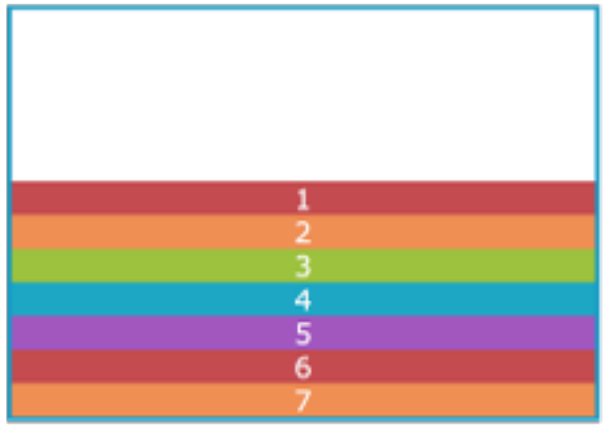


When the direction is **column**, and the **main axis is vertical**

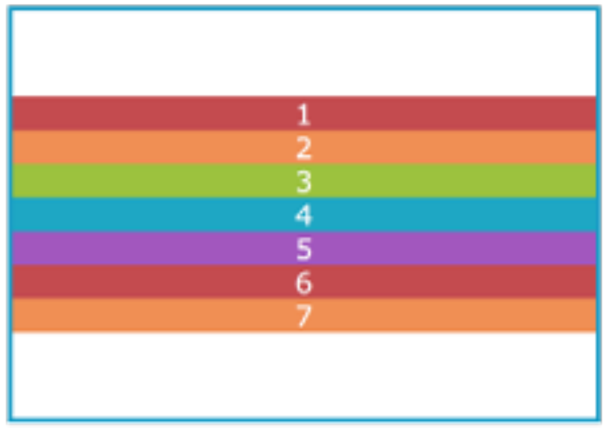
`justify-content: flex-start;` (default)



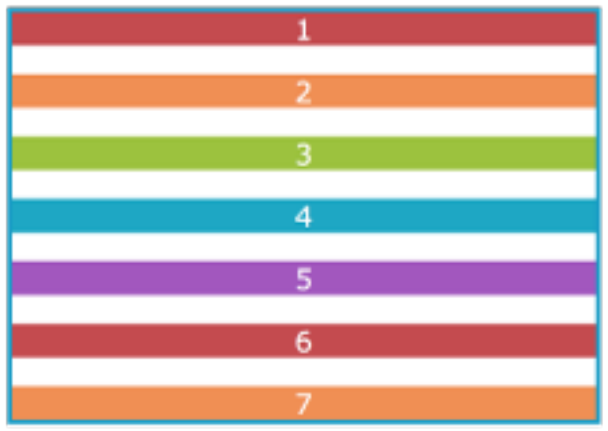
`justify-content: flex-end;`



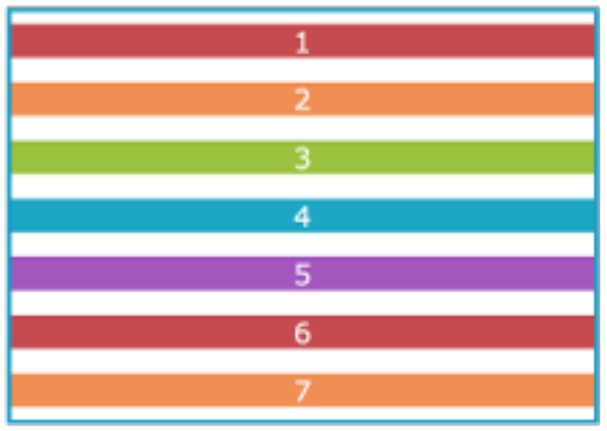
`justify-content: center;`



`justify-content: space-between;`



`justify-content: space-around;`



NOTE: I needed to specify a height on the container to create extra space on the main axis. By default, it's just high enough to contain the content.

“Keeping the main and cross axes straight in your mind when changing between rows and columns is one of the trickiest parts of using flexbox.

Once you master that, you’ve got it!”

Aligning on the cross axis

`align-items`

Values: `flex-start`, `flex-end`, `center`, `baseline`, `stretch`

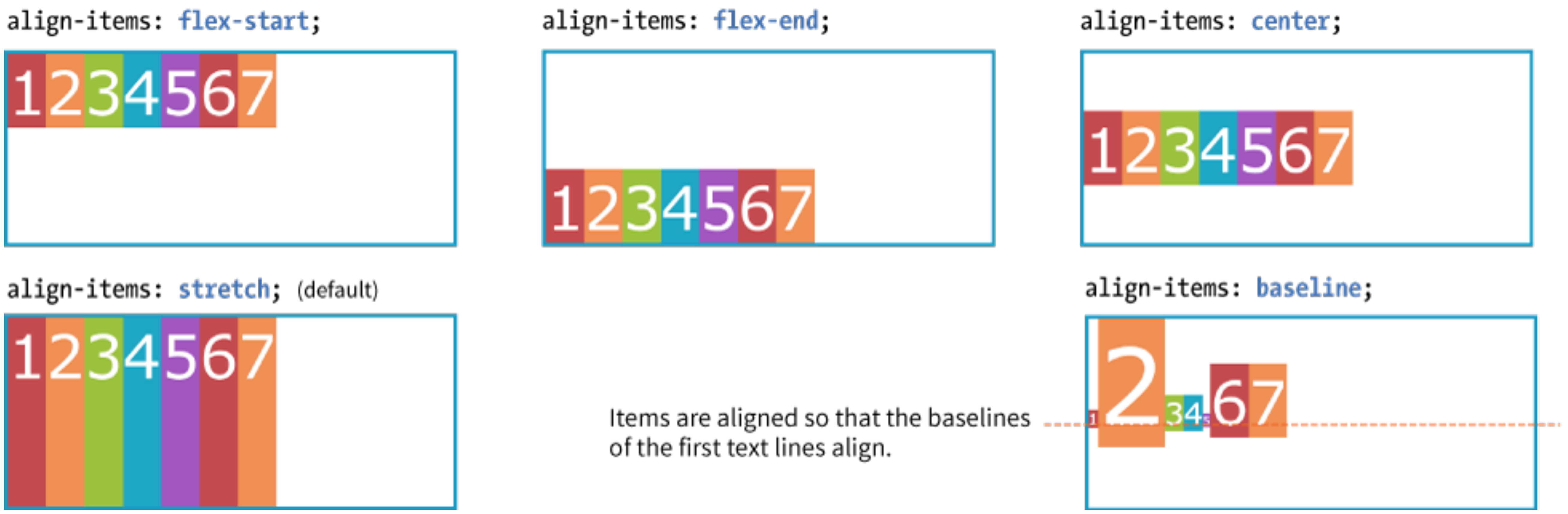
When there is space left over on the **cross axis**, you can specify how the items align with the **`align-items`** property.

The **`align-items`** property applies to the **flex container**.

Example:

```
#container {
  display: flex;
  flex-direction: row;
  height: 200px;
  align-items: flex-start;
}
```

When the direction is **row**, the main axis is horizontal, and the **cross axis is vertical**.



```
align-items: flex-start;
align-items: flex-end;
align-items: center;
align-items: stretch; (default)
align-items: baseline;
```

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

NOTE: Specify a height on the container to create extra space on the cross axis. By default, it's just high enough to contain the content.

`align-self`

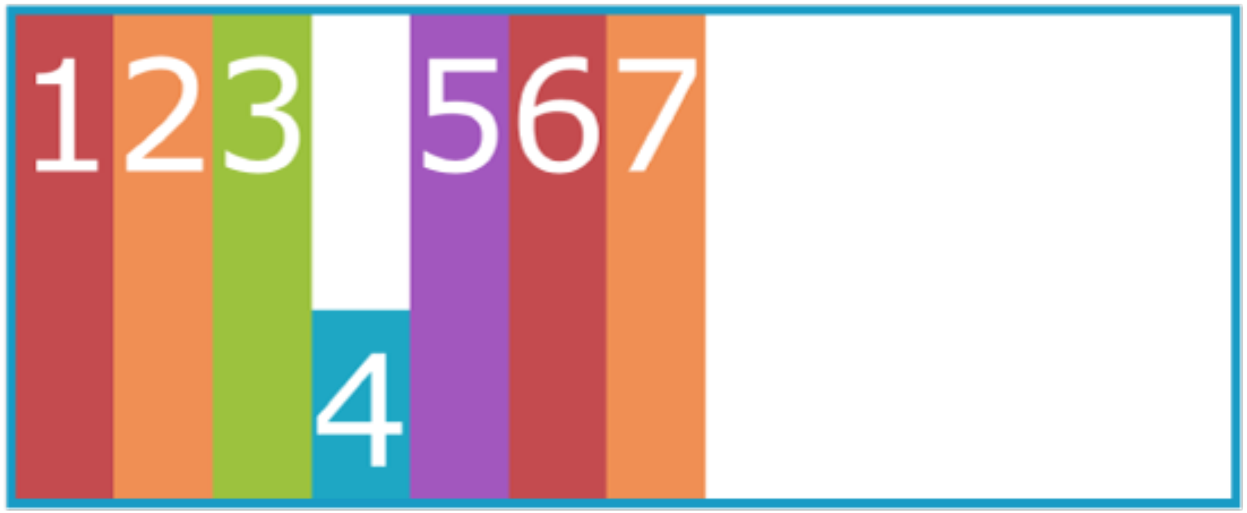
Values: `flex-start`, `flex-end`, `center`, `baseline`, `stretch`

Aligns an **individual item** on the cross axis. This is useful if one or more items should override the **`align-items`** setting for the container.

The **`align-self`** property applies to the **flex item**.

Example:

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

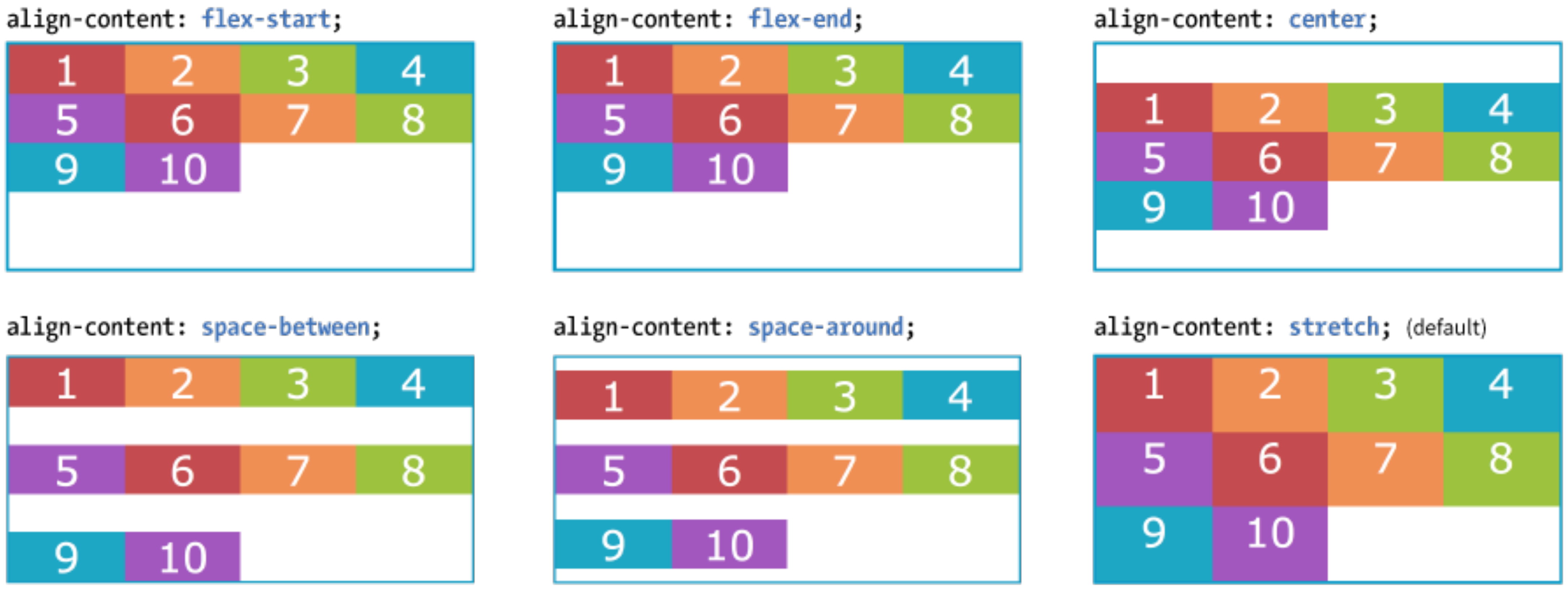


`align-content`

Values: `flex-start`, `flex-end`, `center`, `space-around`, `space-between`, `stretch`, `space-evenly`

When lines are set to **wrap** and there is extra space on the cross axis, use `align-content` to **align the lines of content**.

The **`align-content`** property applies to the **flex container**.

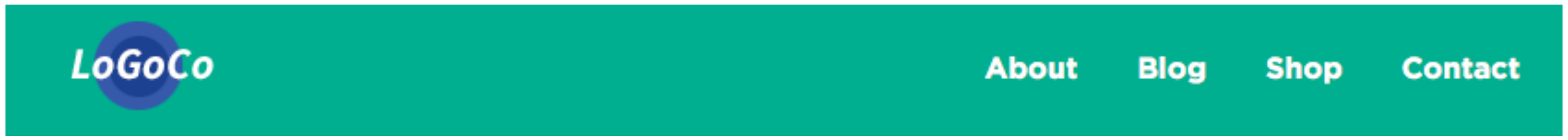


```
align-content: flex-start;
align-content: flex-end;
align-content: center;
align-content: space-between;
align-content: space-around;
align-content: stretch; (default)
```

Aligning with margins

Use a margin (set to auto) to put extra space on the side of particular flex items.

Example: Adding an auto margin to the right of the first flex item (the **`li`** with the logo) pushes the remaining **`li`** to the right:



```
ul {
  display: flex;
  align-items: center;
  ...
}
li.logo {
  margin-right: auto;
}
```

Specifying how items “flex”

`flex`

Values: `none`, '*flex-grow flex-shrink flex-basis*'

- Items can resize (flex) 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 identifies a starting size
- It distributes extra space in the container *within* items (compared to `justify-content` that distributes space *between and around* items).

Flex property example

`Flex` is a shorthand for separate **`flex-grow`**, **`flex-shrink`**, and **`flex-basis`** properties.

The values 1 and 0 work like on/off switches.

```
li {
  flex: 1 0 200px;
}
```

In this example, list items in the flex container start at 200 pixels wide, are permitted to expand wider (`flex-grow: 1`), and are not permitted to shrink (`flex-shrink: 0`).

NOTE: The spec recommends always using the **`flex`** property and using individual properties only for overrides.

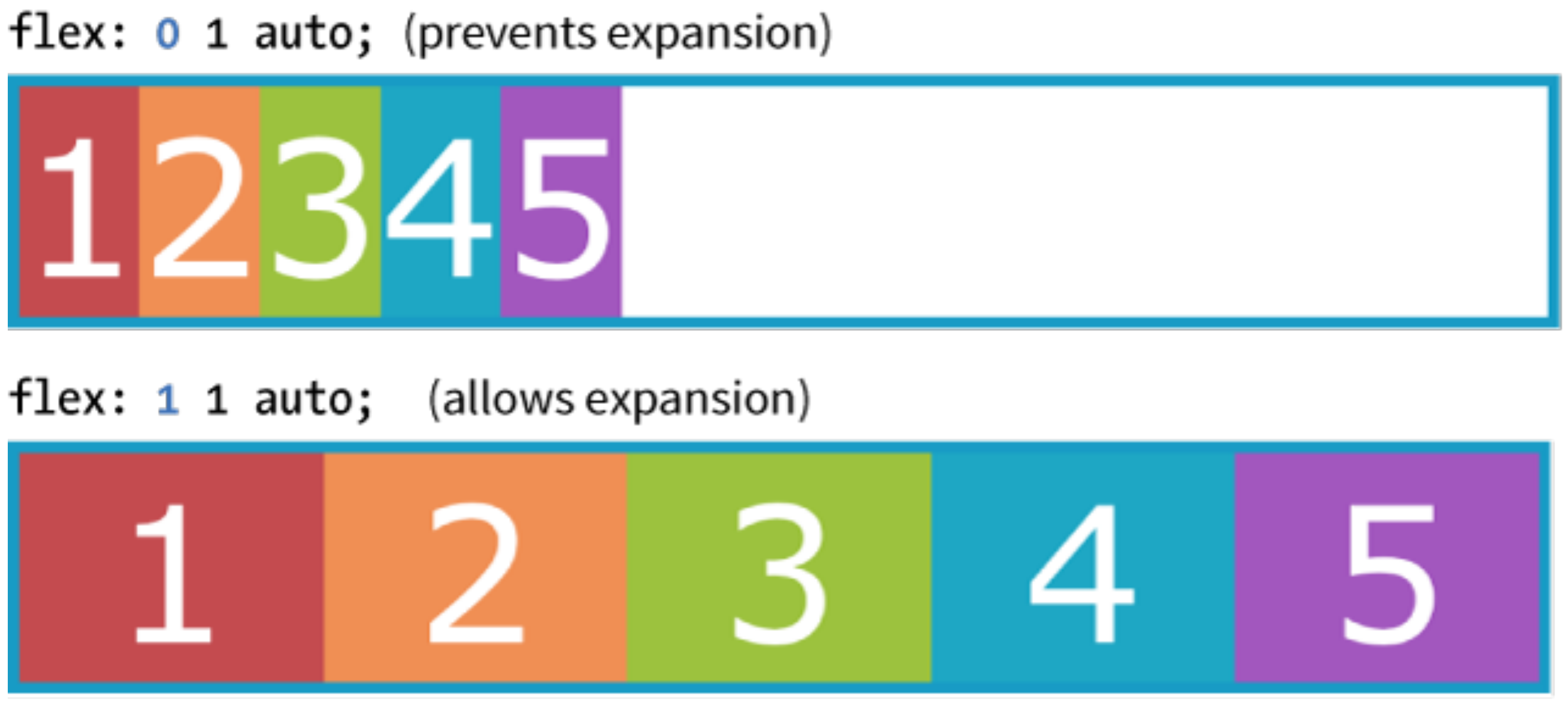
Expanding items (flex-grow)

`flex-grow`

Values: *Number*

Specifies whether and in what proportion an item may stretch larger. 1 allows expansion; 0 prevents it.

`flex-grow` is applied to the **flex item element**.

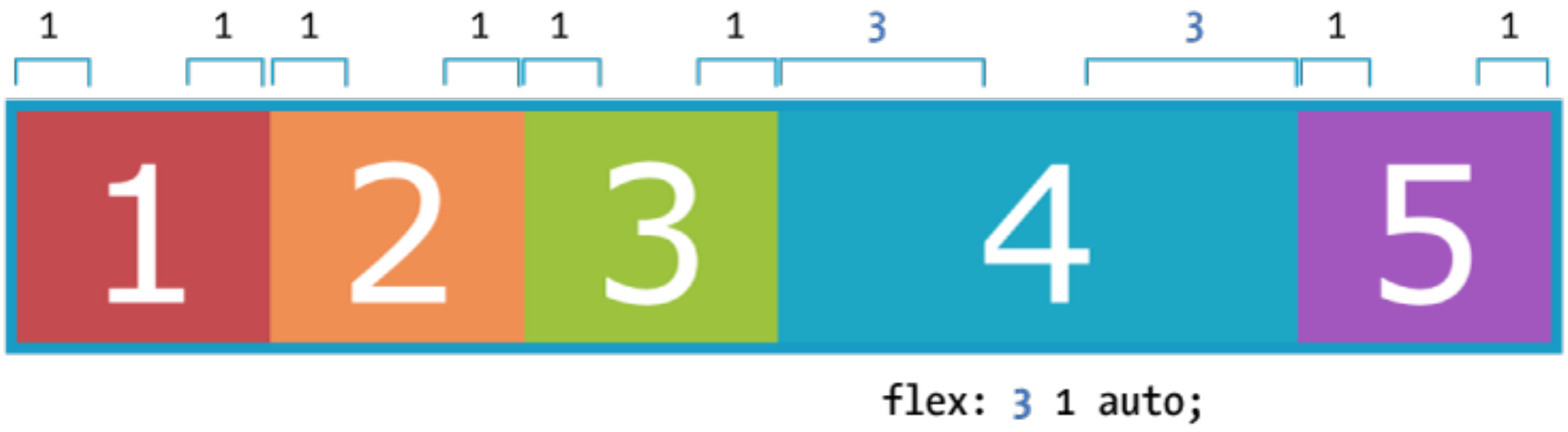

flex: 0 1 auto; (prevents expansion)
12345
flex: 1 1 auto; (allows expansion)
1 2 3 4 5

Relative flex

*When the **`flex-basis`** has a value **other than 0***, higher integer values act as a ratio that applies more space within that item.

Example: A value of **`3`** assigns **three times more space** to box4 than items with a `flex-grow` value of `1`. (Note that it isn't necessarily 3x as wide as the other items.)

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

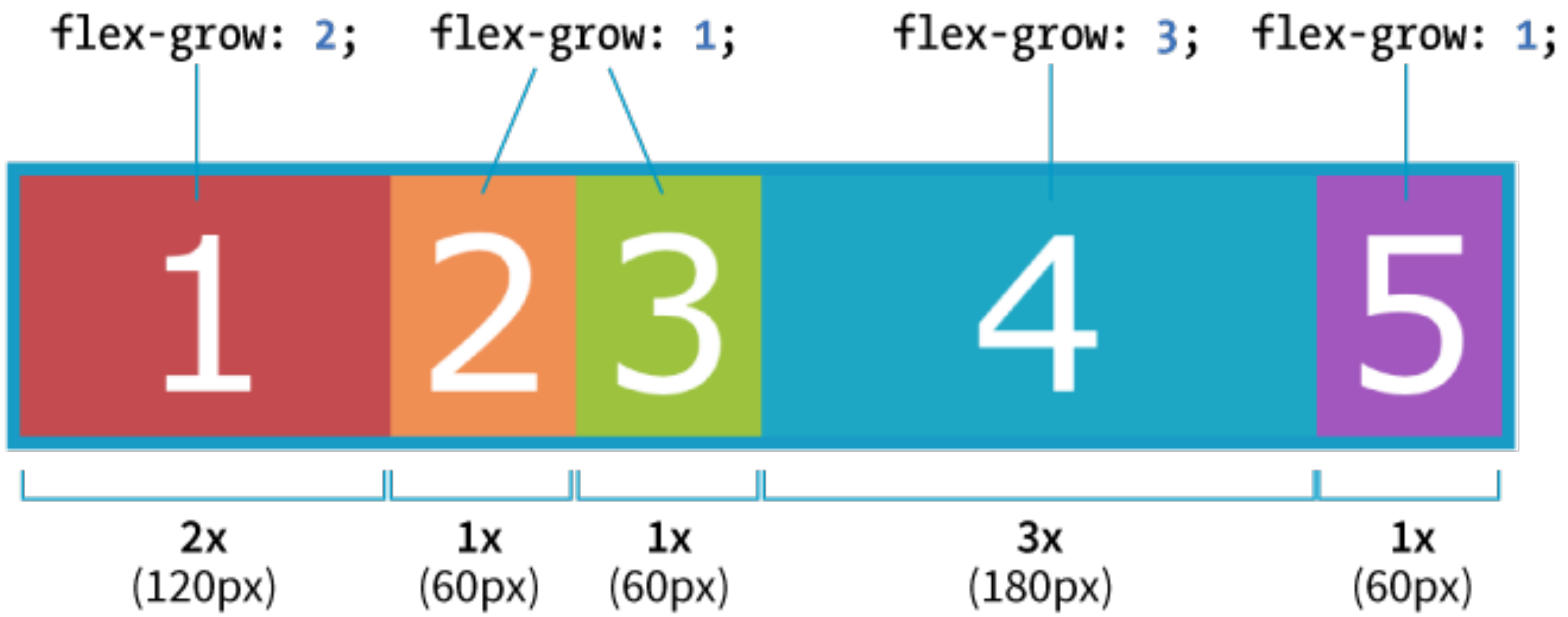


Absolute flex

*When the **`flex-basis`** is **0***, items get sized proportionally according to the flex ratio.

Example: A value of **3** makes "box4" **3x as wide** as the others when `flex-basis: 0`.

```
.box4 { flex: 3 1 0%; }
```



Shortcut flex values

- **`flex: initial`** (same as `flex: 0 1 auto;`)
  Prevents the item from growing, but allows it to shrink to fit the container.

- **`flex: auto`** (same as `flex: 1 1 auto;`)
  Allows items to be 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;`)
  Creates a flexible item with **absolute flex** (so `flex-grow` integer values are applied proportionally)

Changing item order

`order`

Values: *Number*

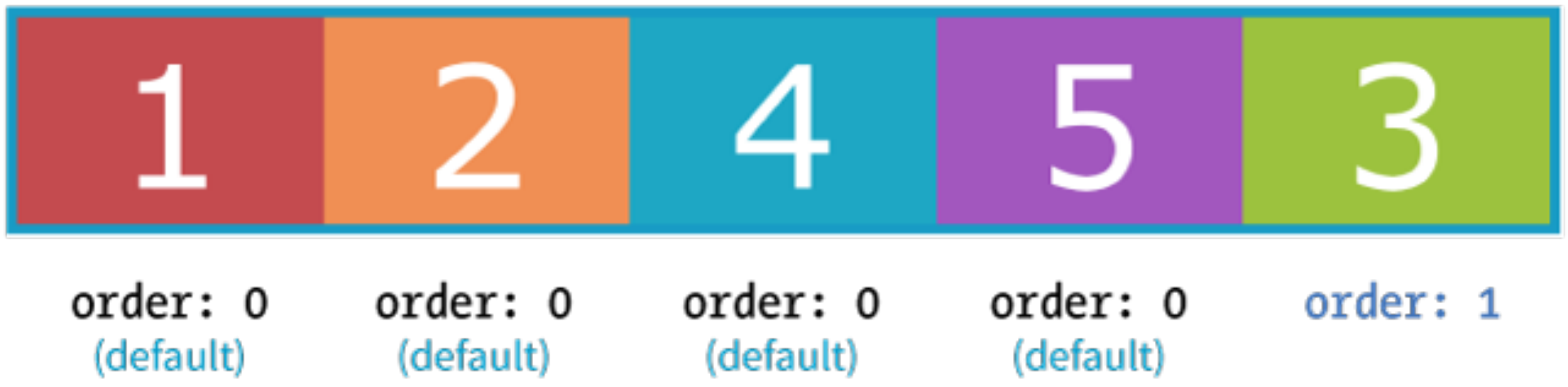
Specifies the order in which a particular item should appear in the flow (independent of the HTML source order):

- **`order`** is applied to the **flex item element**.
- The default is 0. Items with the same order value are placed according to their order in the source.
- Items with different order values are arranged from lowest to highest.
- The specific number value doesn’t matter; only how it relates to other values (like z-index) matters.

Example:

"box3" has a higher order value (1) than the others with default order of 0. It appears last in the line even though it's third in the markup:

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

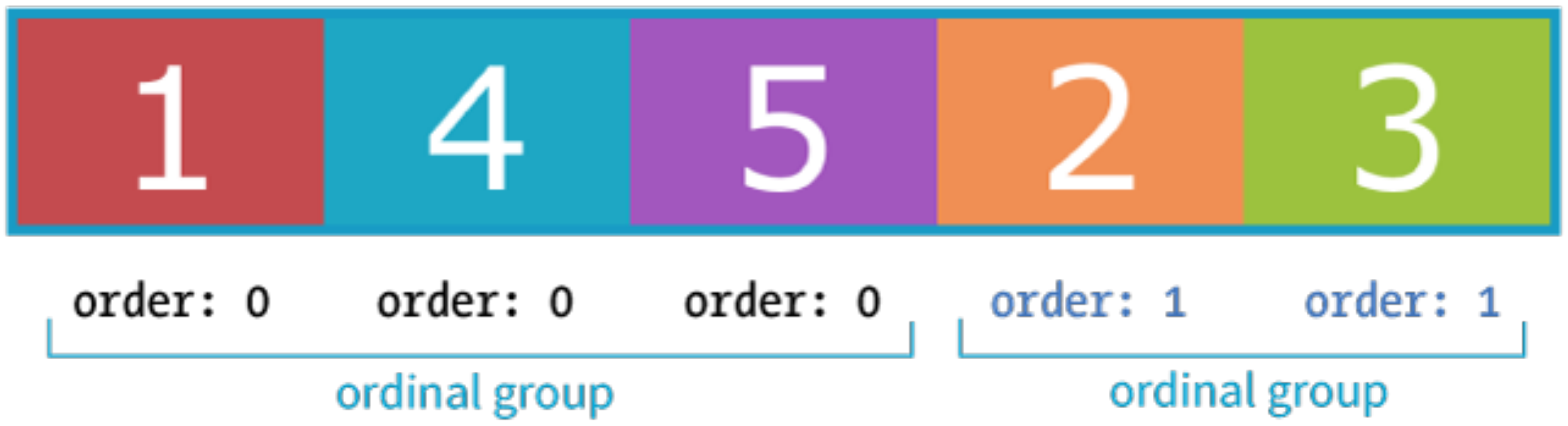


Ordinal groups

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;
}
```



Flex container properties

display: flex | inline-flex;

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

flex-wrap: wrap | nowrap | wrap-reverse;

flex-flow (shorthand for flex-direction and flex-wrap)

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

align-items: flex-start | flex-end | center | baseline | stretch;

align-content (cross axis - adjust to largest item): flex-start | flex-end | center | stretch | space-between | space-evenly | space-around;

Flex item properties

order: <integer>;

flex-grow: <number>;

flex-shrink: <number>;

flex-basis: <length> | auto;

flex: shorthand for grow, shrink, and basis (default: 0 1 auto)

align-self: overrides alignment set on parent