CSS – Flex Box

Topics

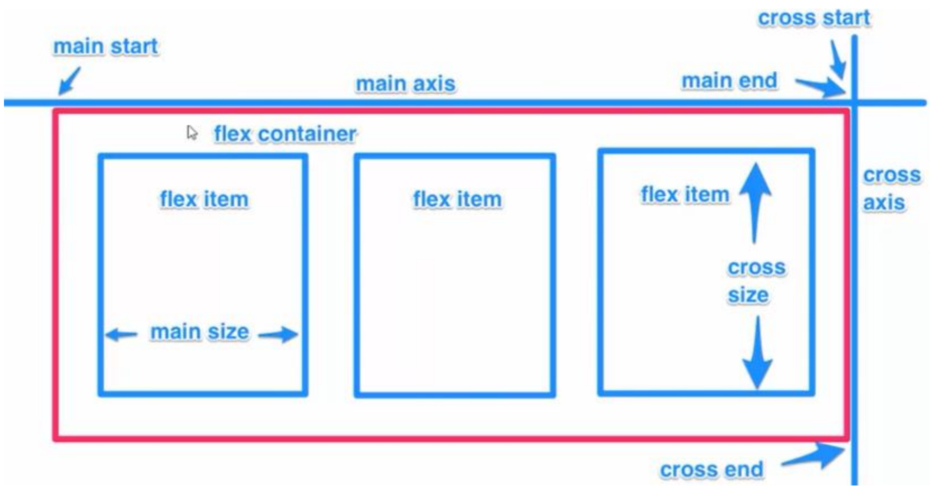
- Flex Box

Flex Box:

- ➢ Flex box is a on-dimensional layout method for laying out items in rows or columns. Items flex to fill additional space and shrink to fit into smaller spaces.
- ➢ For a long time, the only reliable cross browser-compatible tools available for creating CSS layouts were things like floats and positioning. These are fine and they work, but in some ways, they are also rather limiting and frustrating.

In short a CSS layout mode that provides an easy and clean way to arrange items within a container.

Benefits:

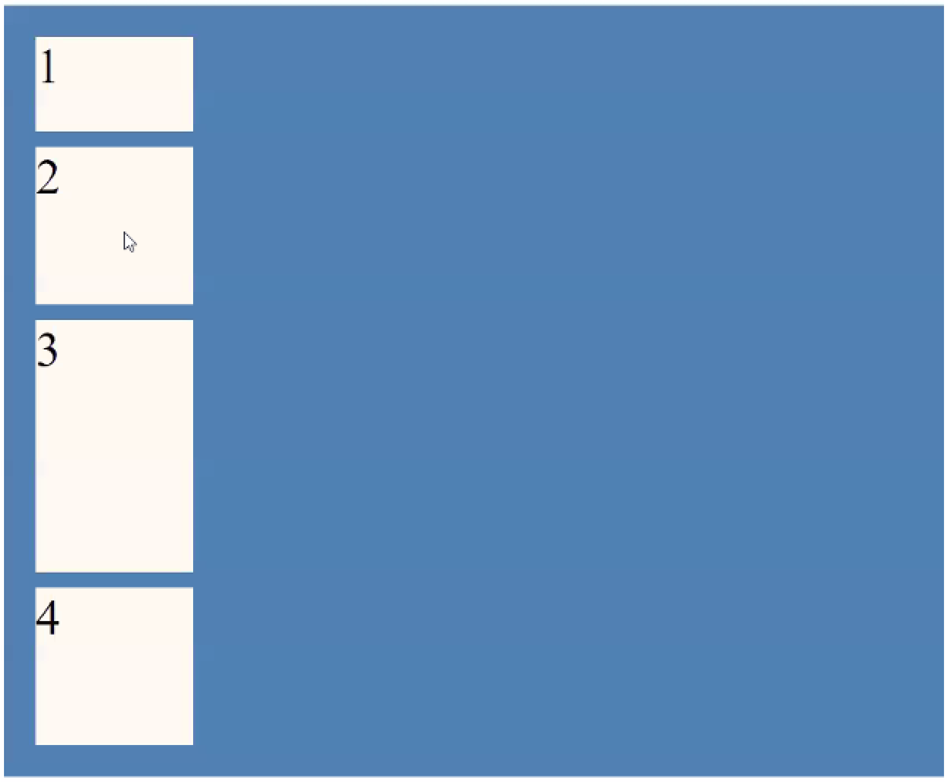


- ✓ NO FLOATS!
- ✓ Responsive and Mobile Friendly
- ✓ Positioning child elements is MUCH easier
- ✓ Flex container's margins do not collapse with the margins of its contents.
- ✓ Order of Elements can easily be changed without editing the source HTML.
- ✓ The ability to alter item width and height to best fit in its containers available free space
- ✓ Flexbox is direction-agnostic
- ✓ Built for small-scale layouts while the upcoming 'Grid' specification is for more large scale.

Example:

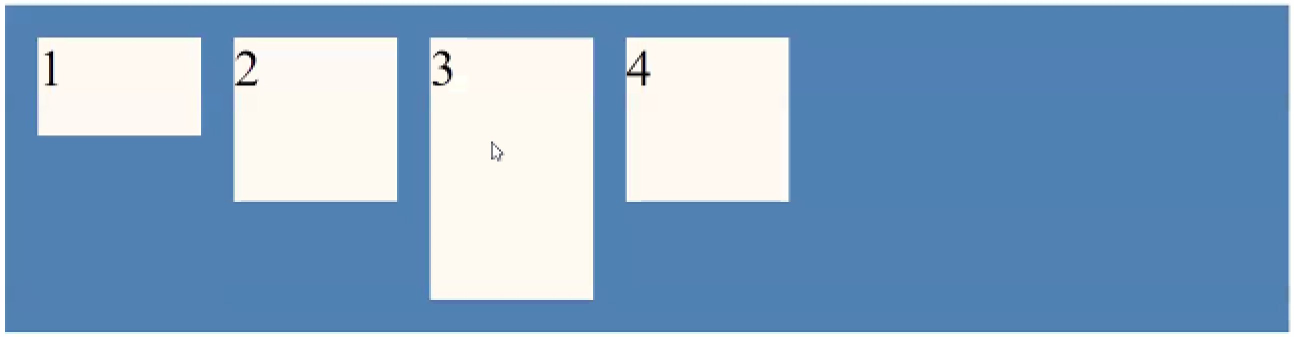
IN our css:

```
.box-container {
background-color: steelblue;
padding:5px;
}

.box {
height: 50px;
width:50px;
margin: 5px;
background-color:
florawhite;
}

.box-3 {
height:80px;
}

.box-1{
height:30px;
}
```



Once we put in display:flex into the css properties,

`.box-container {` `background-color: steelblue;` `padding:5px;` `}`	to	`.box-container {` `background-color: steelblue;` `padding:5px;` **`display: flex;`** `}`

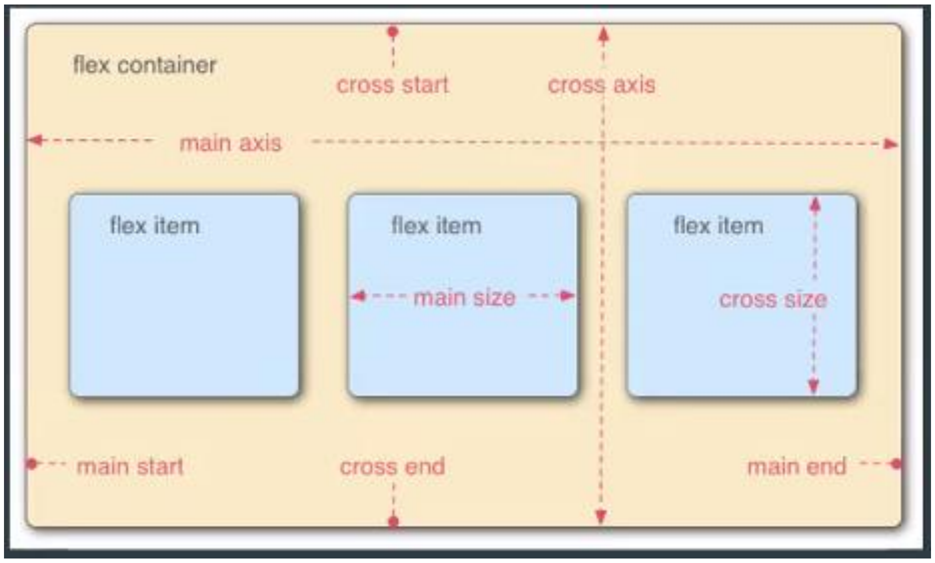


All the element are placed horizontally and the box-container is taking the height so that the longest child element fits in it.

There are two axes defined in the container once we put in display: flex.

- Main axis
- Cross axis

Everything related to orientation happens on main axis and specifi element positioning happens on cross axis. The main axis is horizontal and cross axis is vertical.



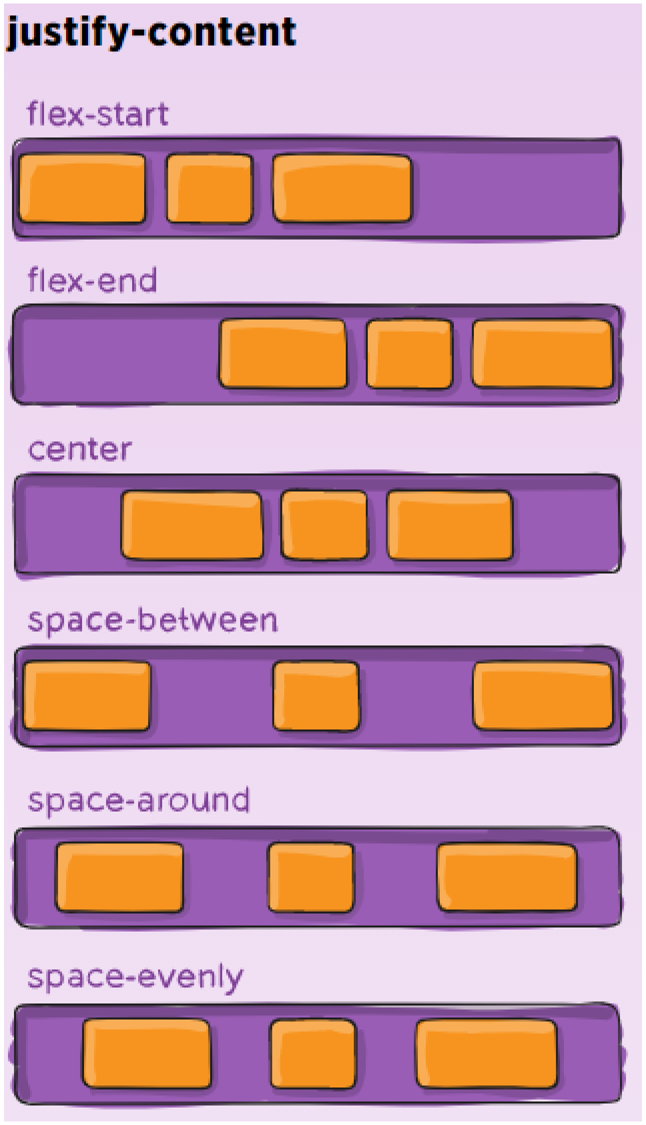
Terminology:

- The **main axis** is the axis running in the direction the flex items are being laid out in (e.g. as row across the page, or column down the page). The start and end of the axis are called the **main start** and **main end**.
- The **cross axis** is the axis running perpendicular to the direction the flex items are being laid out in. The start and end of this axis are called the **cross start** and **cross end**.
- The parent element that has (display: flex) is called the **flex container**.

- ➢ The items being laid out as flexible boxes inside the flex container are called **flex items**.

justify-content:

This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items are on line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

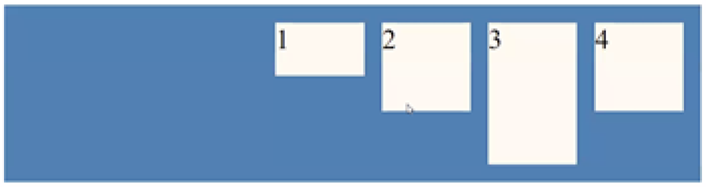


- ✓ ***`flex-start`*** (default): items are packed toward the start of the flex-direction.
- ✓ ***`flex-end`***: items are packed toward the end of the flex-direction.
- ✓ ***`start`***: items are packed toward the start of the writing-mode direction.
- ✓ ***`end`***: items are packed toward the end of the writing-mode direction.
- ✓ ***`left`***: items are packed toward left edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like start.
- ✓ ***`right`***: items are packed toward right edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like end.
- ✓ ***`center`***: items are centered along the line
- ✓ ***`space-between`***: items are evenly distributed in the line; first item is on the start line, last item on the end line
- ✓ ***`space-around`***: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two

units of space between the next item because that next item has its own spacing that applies.

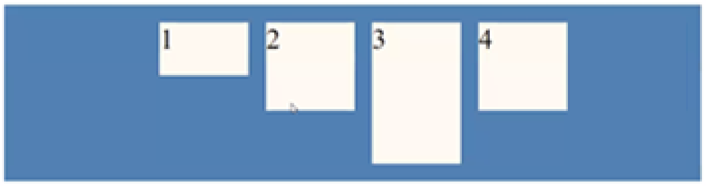
- ✓ ***space-evenly***: items are distributed so that the spacing between any two items (and the space to the edges) is equal.

If we want to the elements at the end of the main axis and not at the starting, then we need to use, **`justified-content: end;`**, then all the elements would shift to the right-hand side.



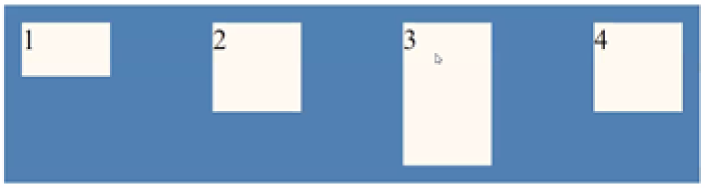
If we want them at the center, we need to use,

`justify-content: center;`



If we use the

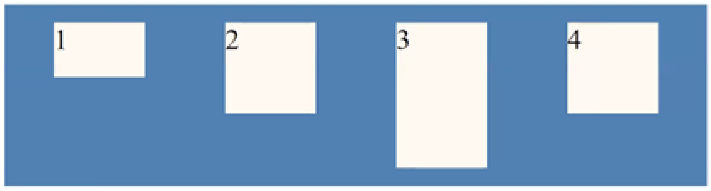
`justify-content: space-between;`



will put all the elements in the box-container equidistant right away, so that they have occupied the complete space of the box-container.

If we use the

`justify-content: space-around;`



it will put the extra space around all the elements instead of the elements that are between the first and the last element.

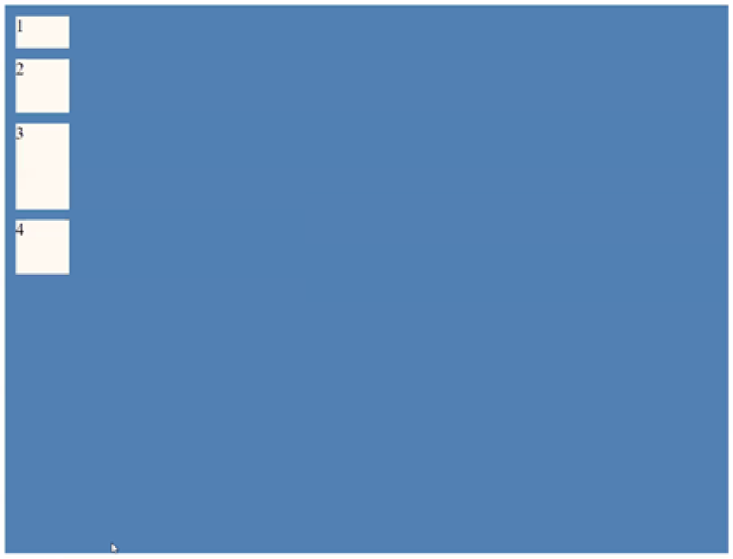
flex-direction:

This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

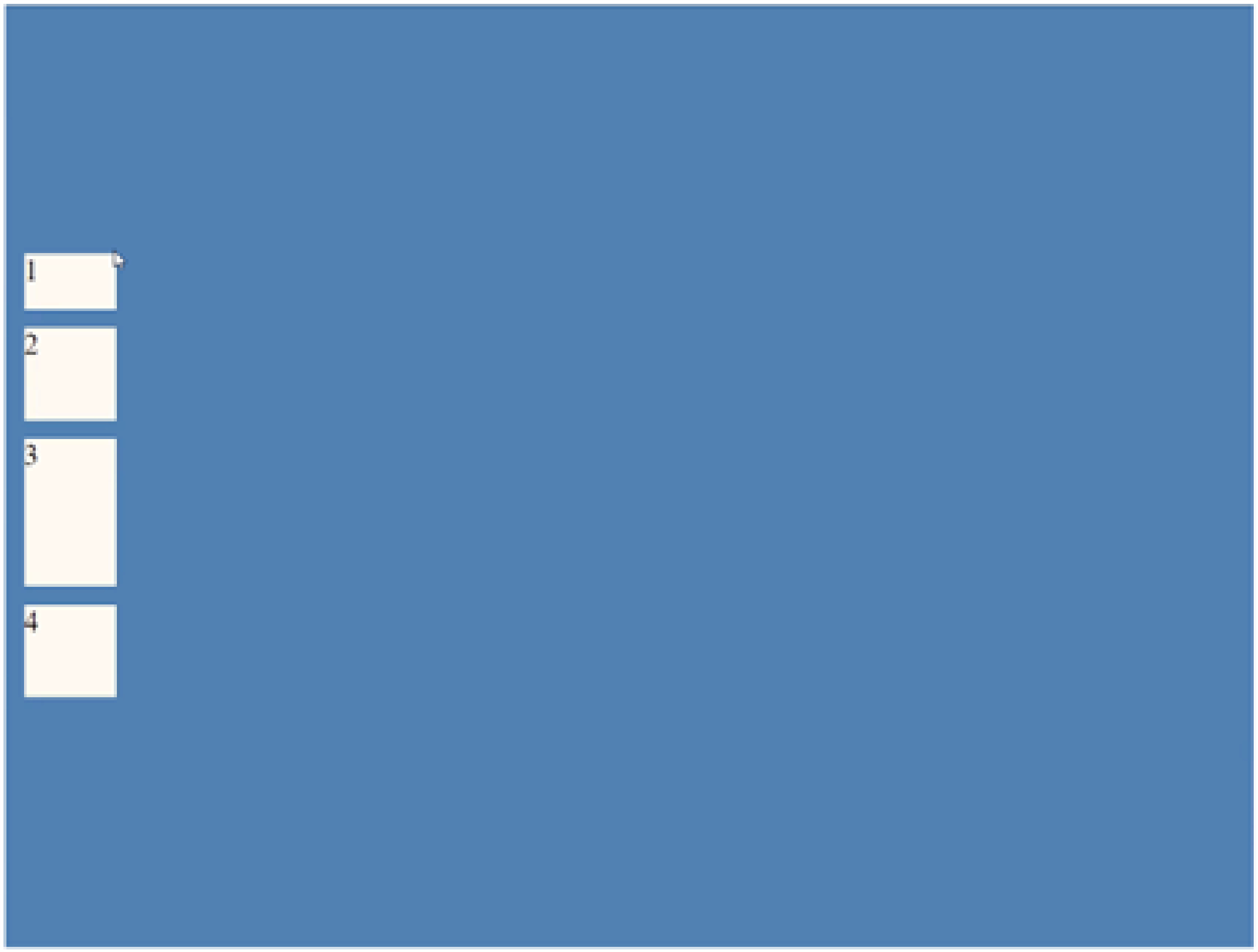
- ✓ ***row (default)***: left to right in `ltr`; right to left in `rtl`
- ✓ ***row-reverse***: right to left in `ltr`; left to right in in `rtl`

- ✓ ***`column`***: same as row but top to bottom
- ✓ ***`column-reverse`***: same as `row-reverse` but bottom to top

The main axis and the cross axis are both in the vertical direction as we have used **`flex-direction: column;`** In other words, we have come back to the same positions that the elements were present in before using the flexbox (**`display: flex;`**)



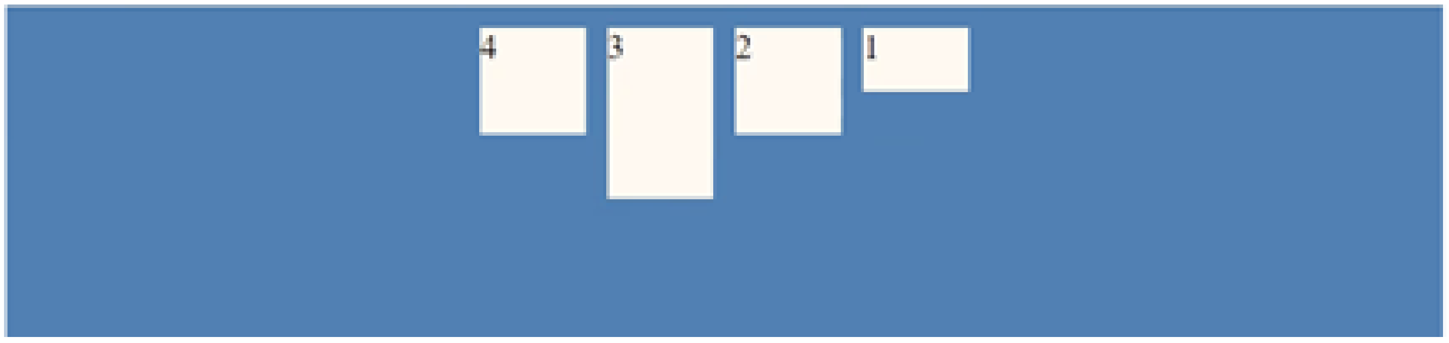
`justify-content: center;` will put the elements at the center as per the vertical axis of the flexbox.



Now, if we change the

```
flex-direction: row-reverse;
```

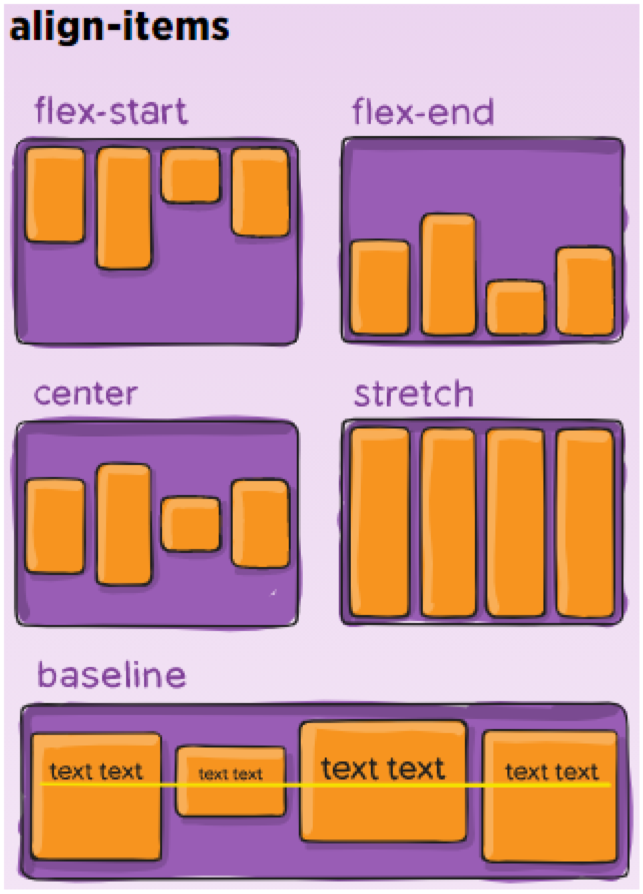
instead of getting the elements from 1 to 4, it will display the element from 4 to 1 (reverse direction).



Now, if we change the **`flex-direction: column-reverse;`** then the element would be displayed in the veritucal direction starting from last element to the first element (4 to 1).

align-items:

This defines the default behavior for how flex items are laid out along the **cross axis** on the current line. Think of it as the **`justify-content`** version for the cross-axis (perpendicular to the main-axis).



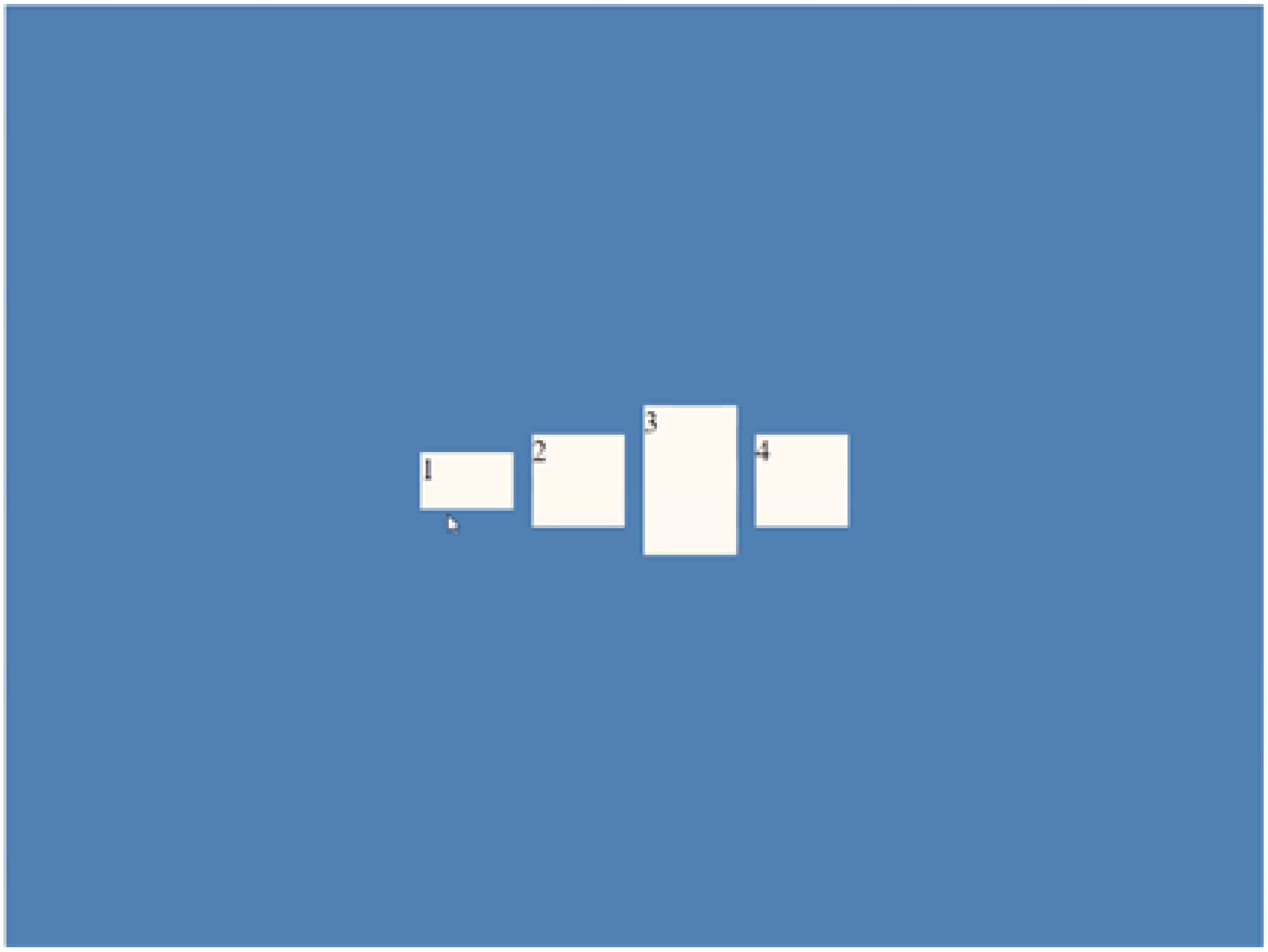
- ✓ ***`Stretch`***: stretch to fill the container (still respect min-width/ max-width)
- ✓ ***`Flex-start/start/self-start`***: items are placed at the start of the cross axis. The difference between these is subtle, and is about respecting the `flex-direction` rules or the `wirting-mode` rules.
- ✓ ***`Flex-end/end/self-end`***: items are placed at the end of the cross axis. The difference again is subtle and is about respecting **`flex-direction`** rules vs **`writing-mode`** rules.
- ✓ ***`Center`***: items are centered in the cross-axis
- ✓ ***`Baseline`***: items are aligned such as their baselines align

If the main axis is in horizontal direction and the cross axis is in the vertical direction, then **`align-items`** would put the elements based on the property we give.

If we put

```
justify-content: center;
align-items: center;
```

then all the elements would be at the middle of the parent element (flex box container).

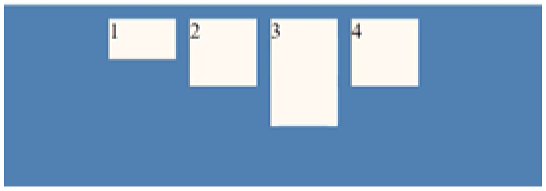


*We have all the elements (**children elements**) at the middle of the flex box (**parent element**).*

If we use

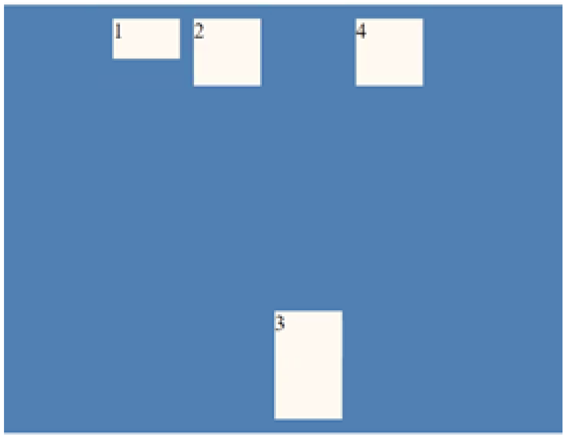
`align-items: baseline;`

the base of the text is aligned now and not the bottom or top of the element being in the same line.



align-self:

All the properties are applied to the flexbox itself. So, all the elements of the flex are being organized together. But, if we want one particular element in the flexbox to be placed in a particular position, then we need to apply changes to the individual element. By using **`align-self`**,

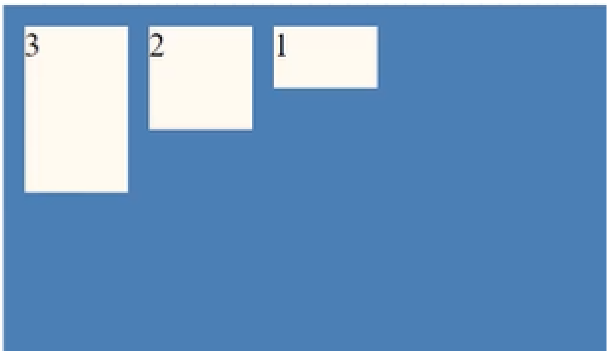


```
.box-3{
align-self: bottom;
}
```

order:

By default, flex items are laid out in the source order. However, the order property controls the order in which they appear in the flex container.

```
.box-1{
order: 2;
}
.box-2{
order: 1;
}
.box-3{
order: 0;
}
```



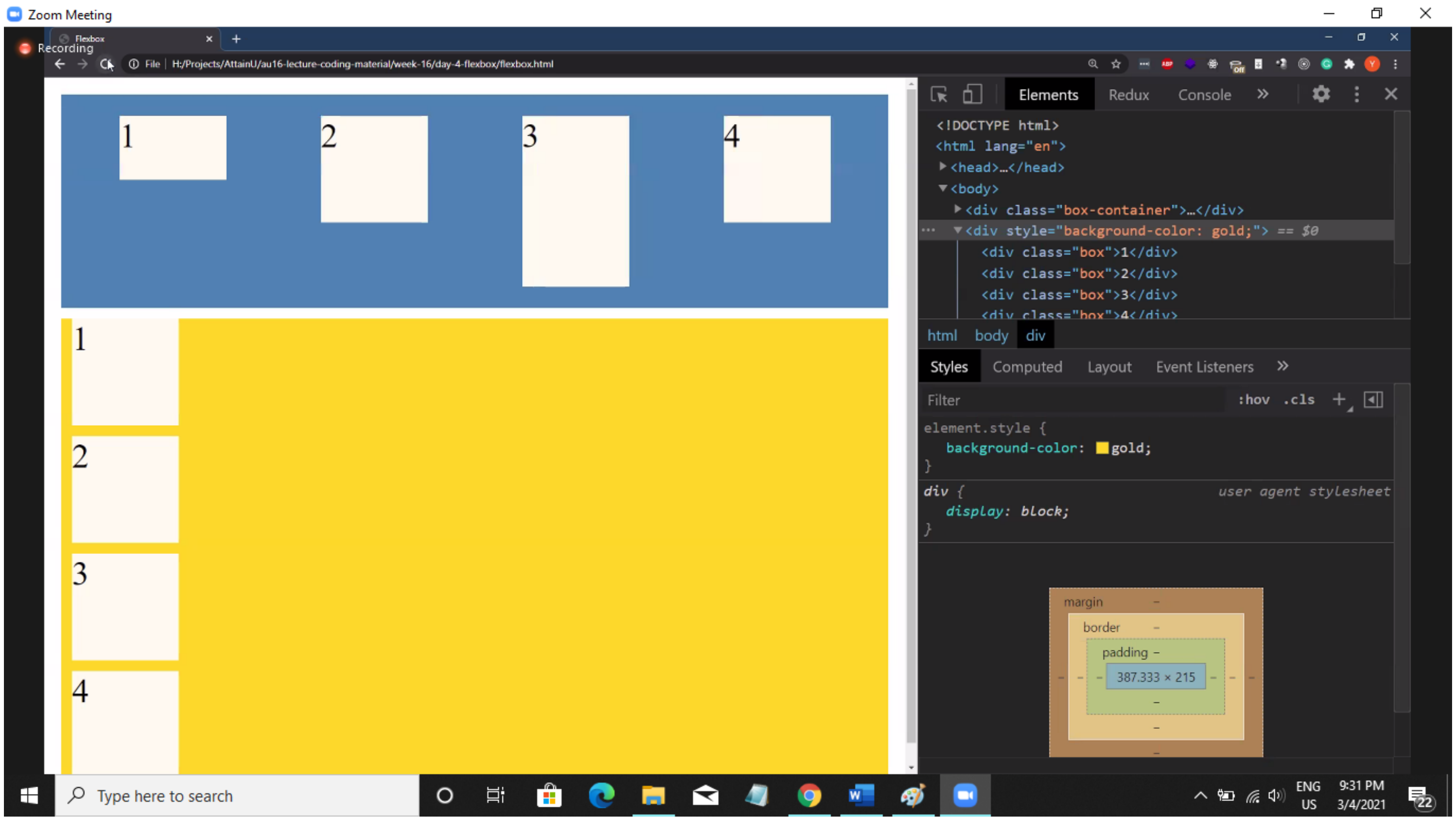
We need to give order to all the elements inside the flexbox.

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

http://flexboxfroggy.com/

http://www.flexboxdefense.com/

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox



```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▶<div class="box-container">…</div>
  ▼<div style="background-color: gold;"> == $0
      <div class="box">1</div>
      <div class="box">2</div>
      <div class="box">3</div>
      <div class="box">4</div>
```

```
element.style {
  background-color: gold;
}
div {                                user agent stylesheet
  display: block;
}
```