CSS – Recap

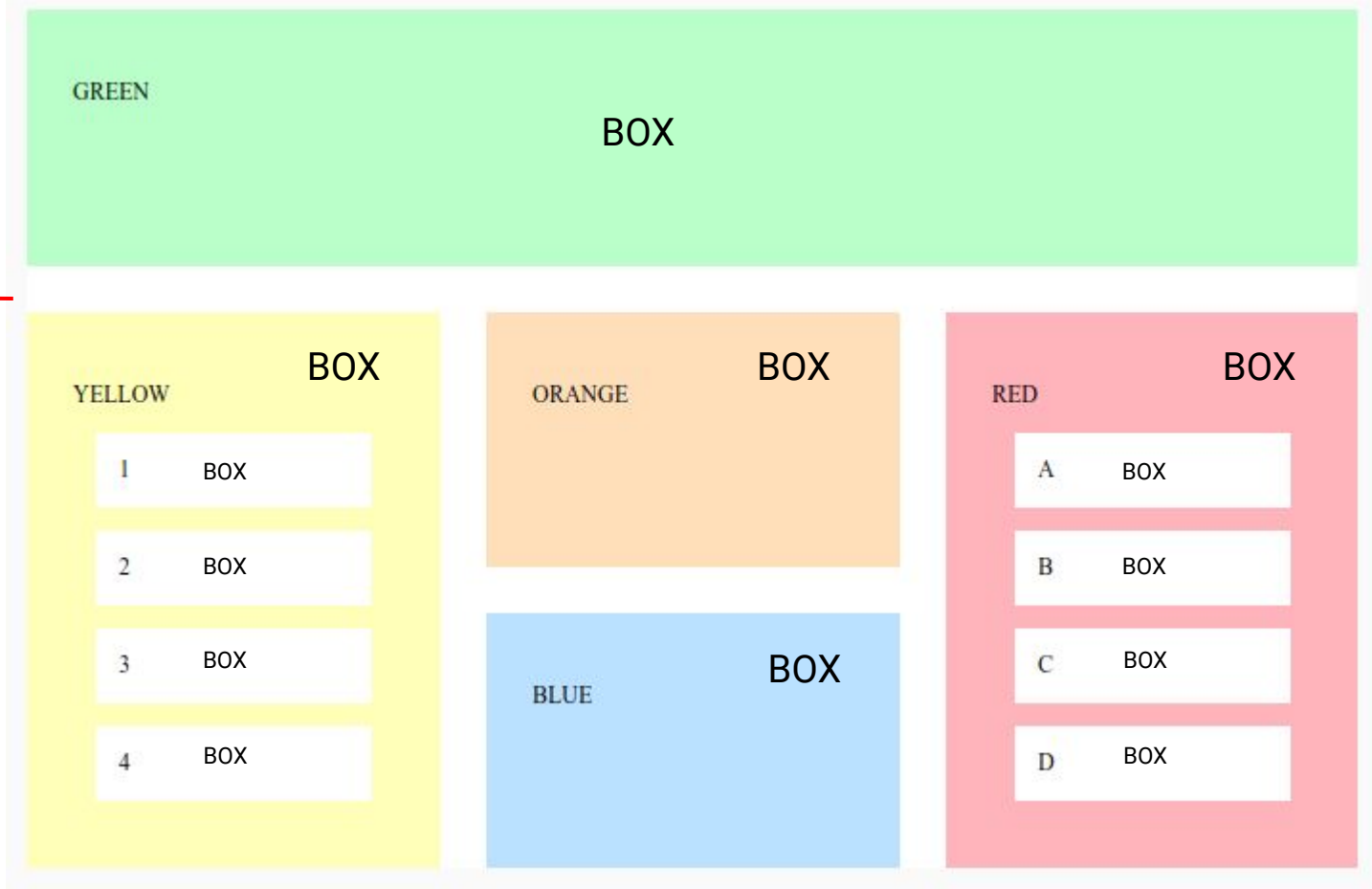
Agenda

- Layouting
- Flexbox
- Modularization
- Questions

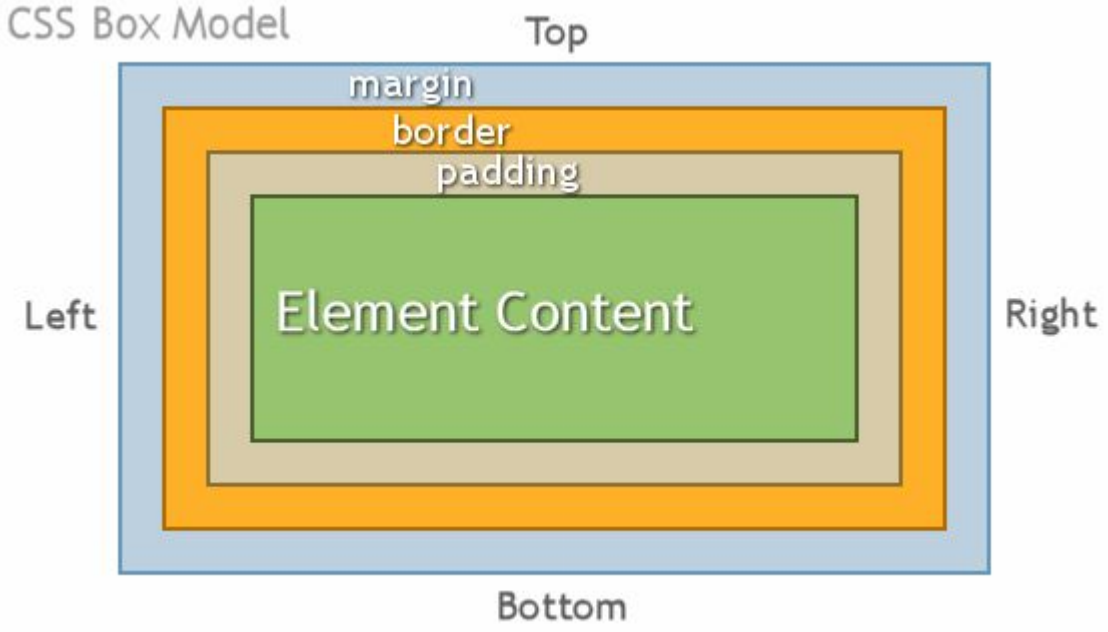
Layouting

The UI Composition

Box Model


GREEN
BOX
BOX
YELLOW
BOX
1
BOX
2
BOX
3
BOX
4
BOX
ORANGE
BOX
BLUE
BOX
RED
BOX
A
BOX
B
BOX
C
BOX
D
BOX

Source: 9Minecraft
MeinKraft Resource Pack 1.17.1/1.16.5 - 9Minecraft.Net



What includes a box?

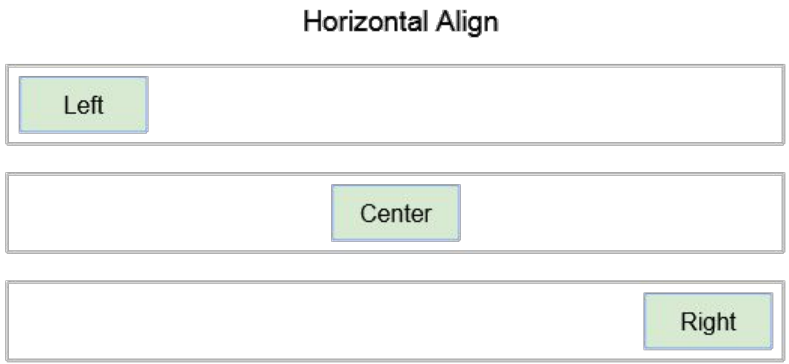
Types of Layouting

Which techniques you have!

- Centering
- Floating
- Positioning
- Using display properties
- Flex Layout
- CSS Grid
- Multi-column layout

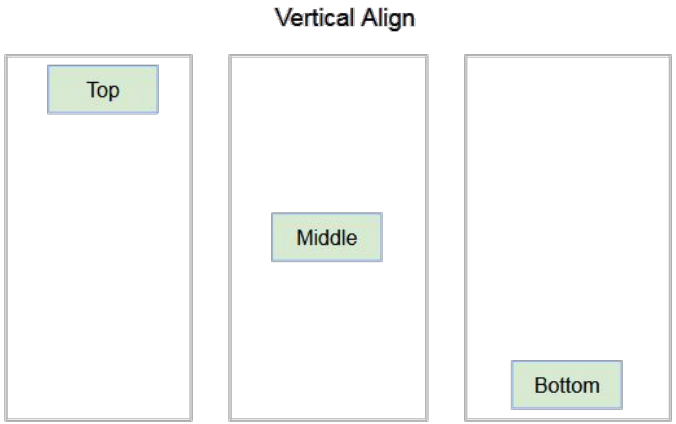
Horizontal centering

- Use auto margin
- Use text-align
- Use positioning
- Use flexbox



Vertical centering

- Use padding
- Use line-height
- Use positioning
- Use vertical-align:middle
- User transforms
- Use flexbox



Floating

Aside from the simple example of wrapping text around images, floats can be used to create entire web layouts.

- CSS Tricks

- Default layout/flow: Positioning elements to top-left
- Float property enables changing the default behaviour
- Possible values: none, left, right
- Float can be neutralized/cleared.

CSS

- float: left;
- float: right;
- clear: both;

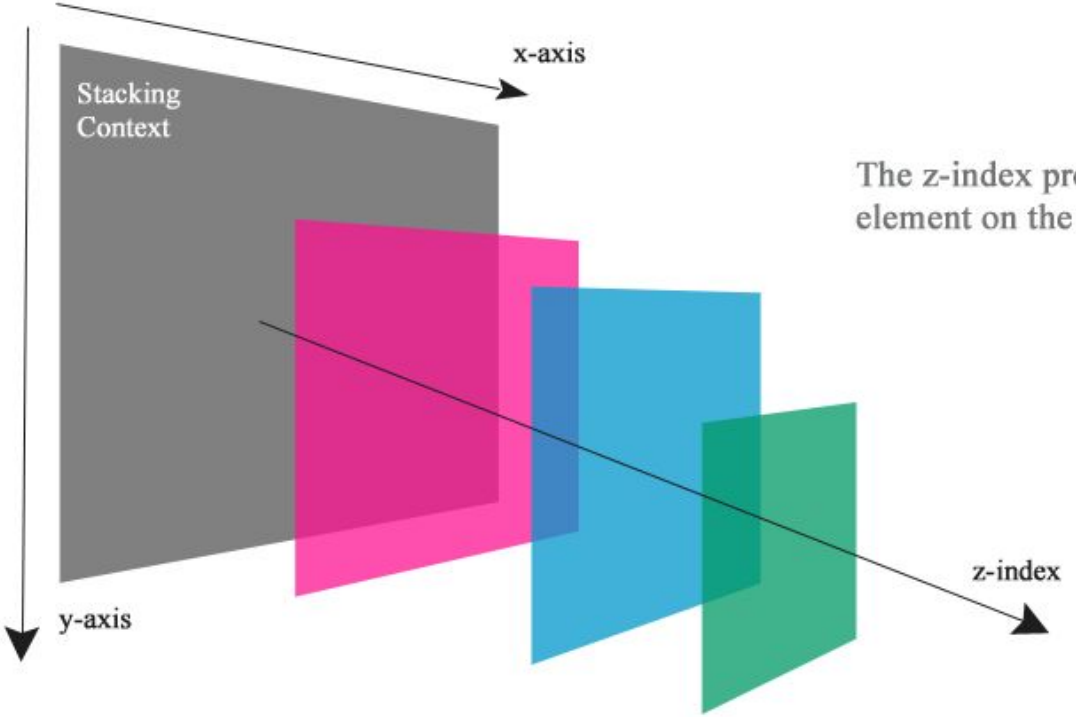
Positioning

Put the elements using a coordinate-system (x, y).

- Put elements out of order or flow
- Different types
  - relative,
  - static
  - absolute,
  - fixed,
  - sticky

CSS

- position: relative | …
- top, right, bottom, left, and z-index



The z-index property determines the order of the element on the z-axis of the stacking context.

Dimensions

Display property

- Block, Inline, Inline-block
- Table, row, cell
- Flex
- Grid
- List

Flexbox

Why FlexBox?

Here Flexbox offers a solution that is flexible compared to the block layout of the box model.

With Flexbox ...

- the flow direction can be freely determined
- elements can be arranged within one line or in several lines by using breaks
- elements are given flexible sizes that are aligned with the viewport
- can be used to position elements in a different order in the markup.

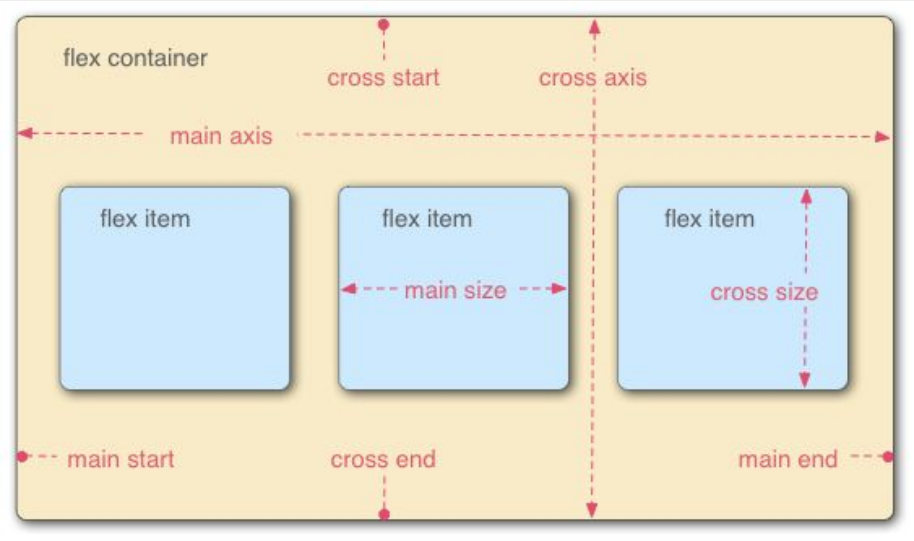
If the width of the viewport is no longer sufficient for a horizontal multi-column layout, media queries can be used to change the direction of the layout to a vertical layout.

FlexBox

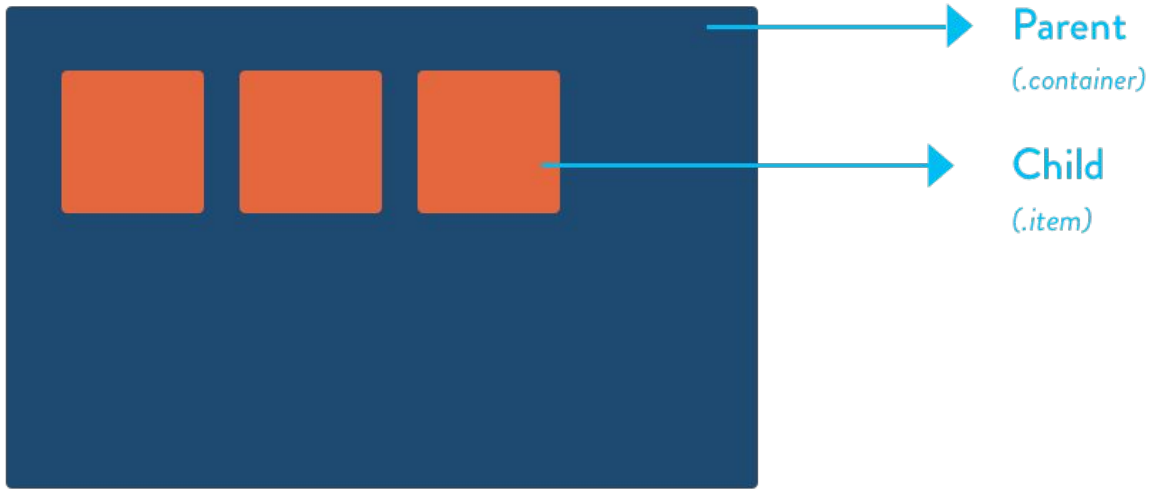
The main concepts.

Learn them before you go in deeply!

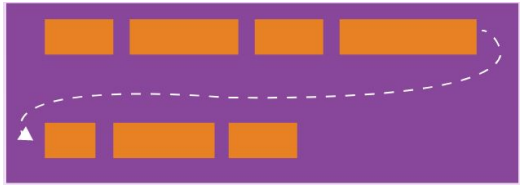
- Container und Items
  - Or: parent element and child elements
- https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox
- With **display:flex**

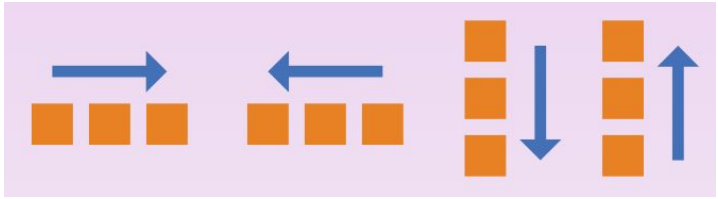


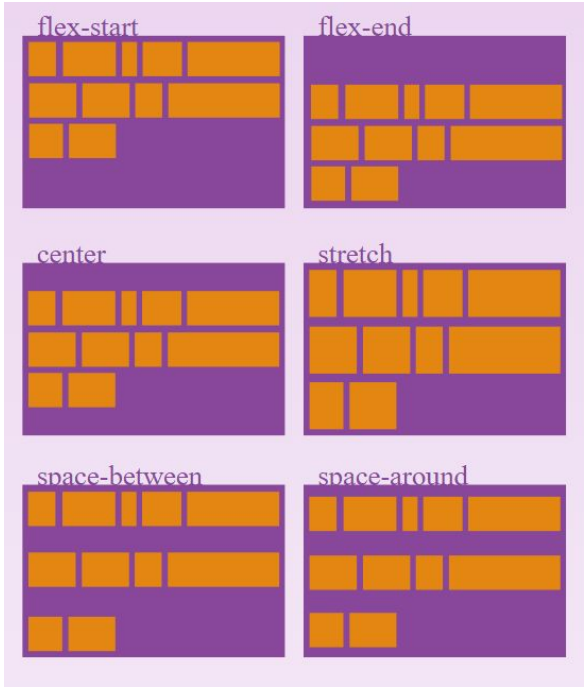
CSS Flexbox

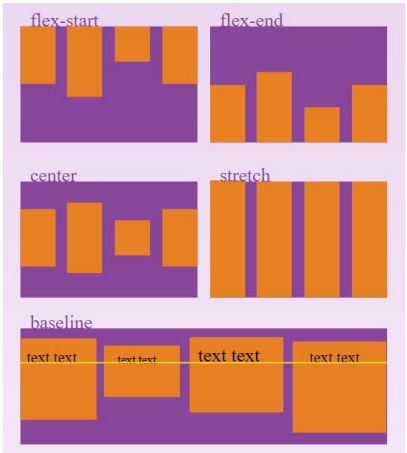


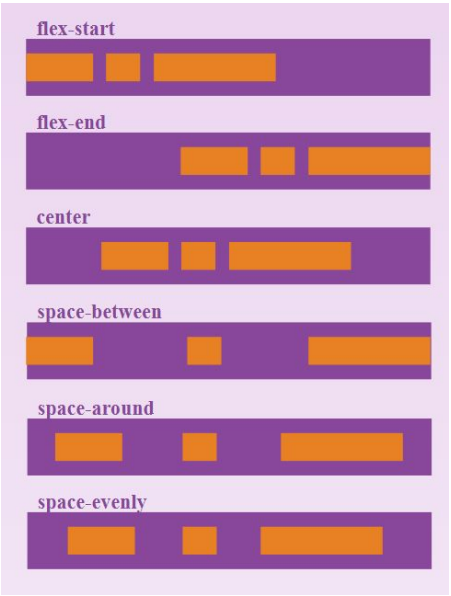
flexbox parent, child






flex-start
flex-end
center
stretch
space-between
space-around

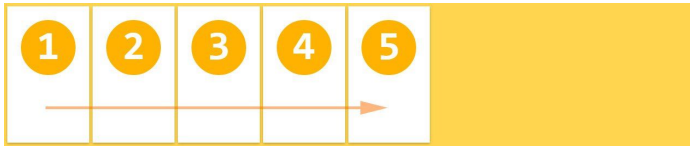

flex-start
flex-end
center
stretch
baseline
text text
text text
text text
text text


flex-start
flex-end
center
space-between
space-around
space-evenly

flex container

the parent

Flexbox Container Properties



- **flex-direction**
  - With row direction the flex items are stacked in a row from left-to-right in ltr context

```
.flex-container {
  display: flex;
  flex-direction:  row;
}
```

https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties

Default value: row

Flexbox Container Properties

- **flex-direction**
  - With column direction the flex items are stacked in a column from top-to-bottom

```
.flex-container {
  display: flex;
  flex-direction:  column;
}
```

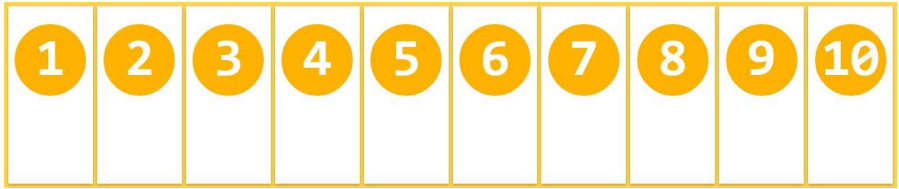
Flexbox Container Properties

- **flex-direction**

  Values:

  - `row`
  - `row-reverse`
  - `column`
  - `column-reverse`

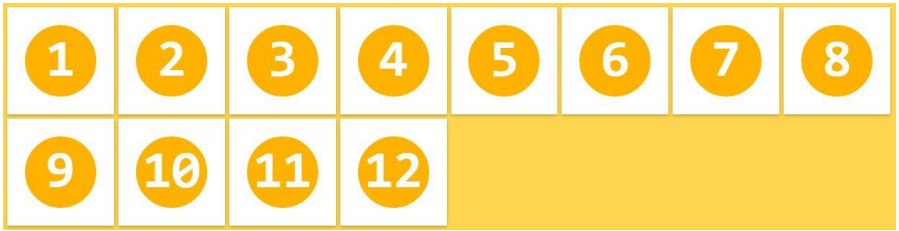
Flexbox Container Properties



- **flex-wrap**
  - Flex items are displayed in one row, by default they are shrunk to fit the flex container's width

```
.flex-container {
  display: flex;
  flex-wrap: nowrap;
}
```

Flexbox Container Properties



- **flex-wrap**
  - Flex items are displayed in multiple rows if needed from left-to-right and top-to-bottom

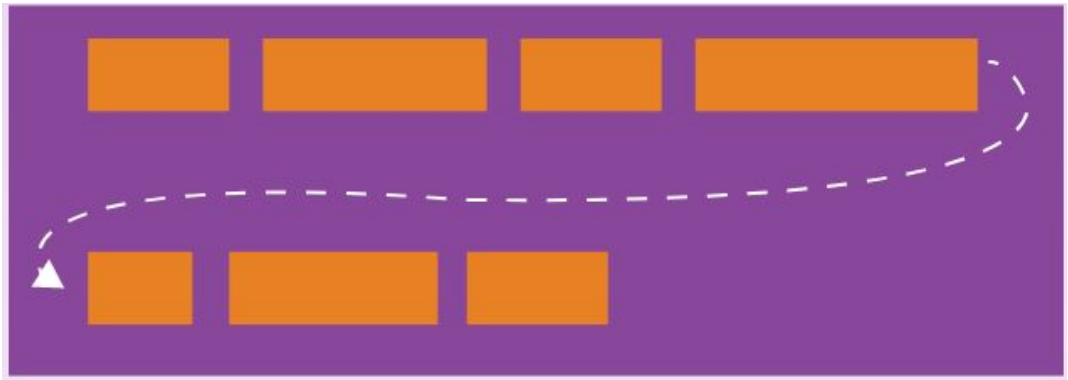
```
.flex-container {
  display: flex;
  flex-wrap: wrap;
}
```

Flexbox Container Properties

- **flex-wrap**

  Values:

  - *nowrap*
  - *wrap*
  - *wrap-reverse*



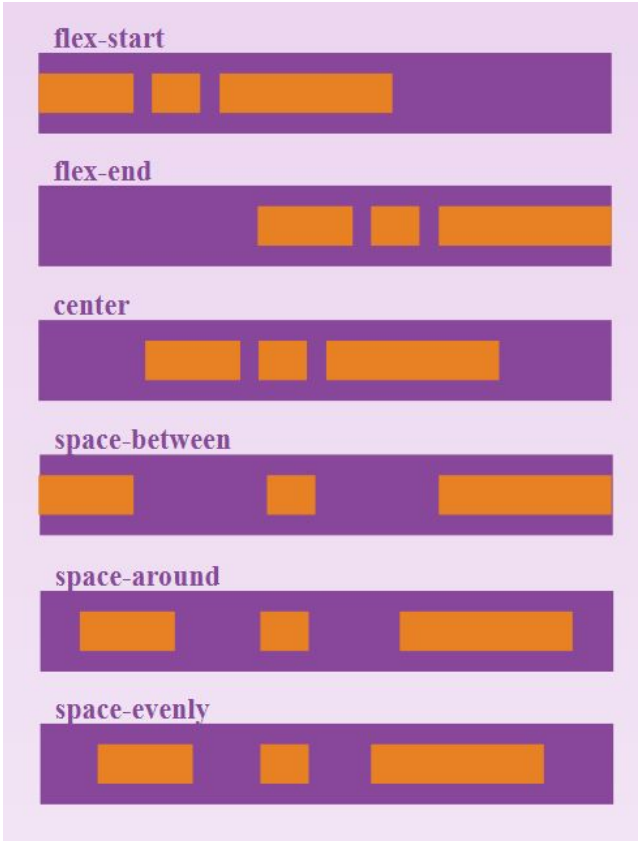
flex-wrap

Flexbox Container Properties

- **justify-content**

  Values:

  - *flex-start*
  - *flex-end*
  - *center*
  - *space-between*
  - *space-around*
  - *space-evenly*



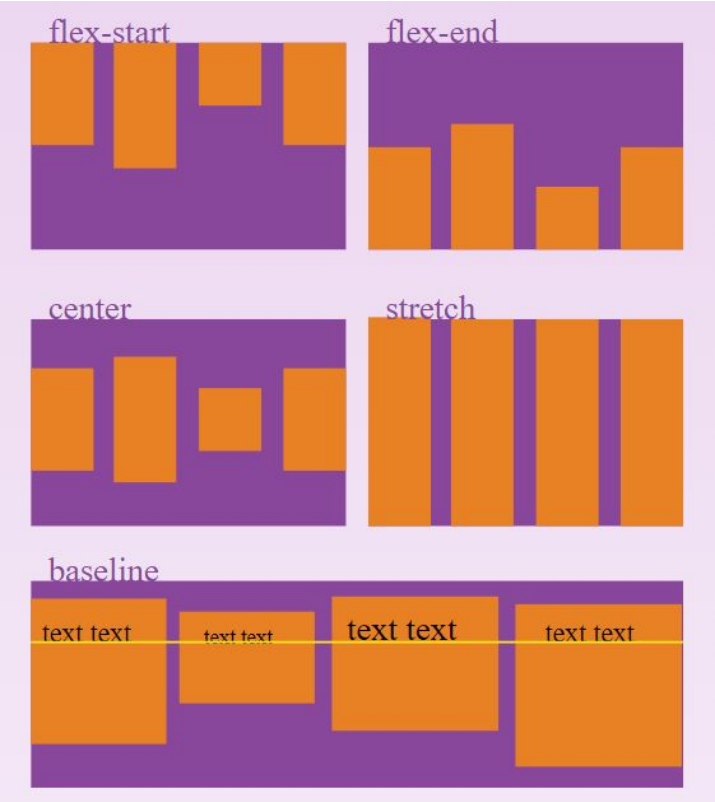
justify-content

Flexbox Container Properties

- **align-items**

  Values:

  - *stretch*
  - *flex-start*
  - *flex-end*
  - *center*
  - *baseline*


flex-start
flex-end
center
stretch
baseline
text text
text text
text text
text text

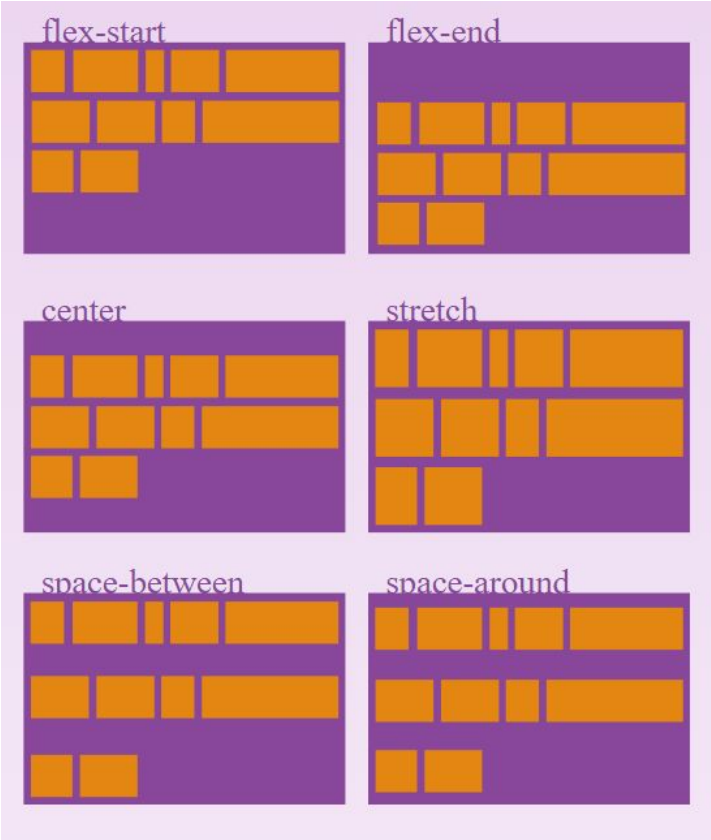
align-items

Flexbox Container Properties

- **align-content**

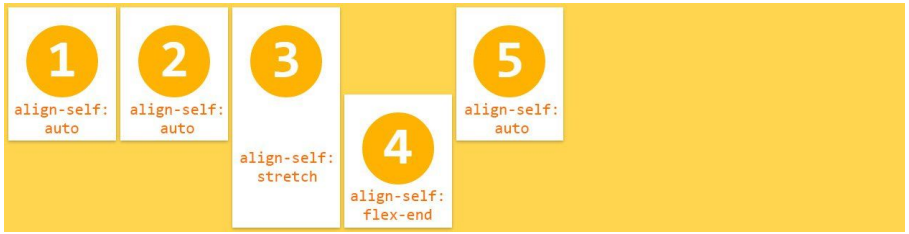
  Values:

  - *flex-start*
  - *flex-end*
  - *center*
  - *space-between*
  - *space-around*
  - *space-evenly*



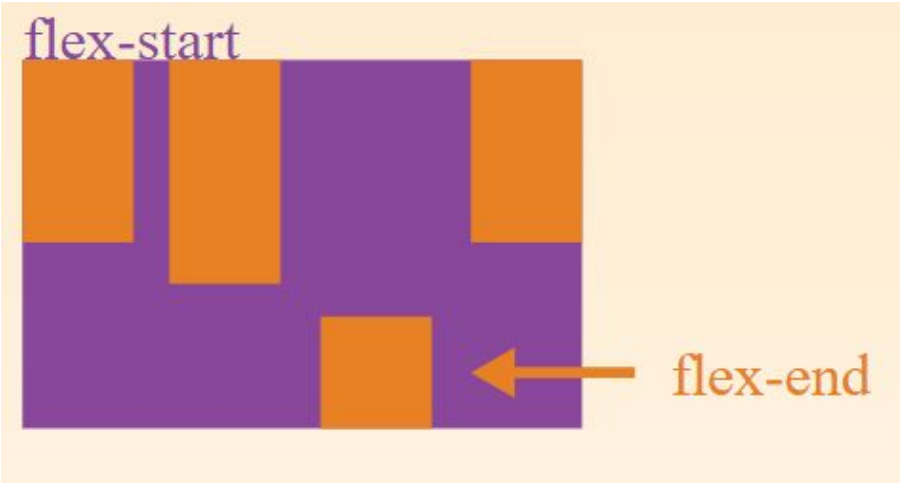
align-content

Flexbox Item Properties



- **align-self**
  - This align-self property allows the default alignment (or the one specified by align-items) to be overridden for individual flex items. Refer to align-items explanation for flex container to understand the available values.

```
.flex-item {
align-self: auto | flex-start |
flex-end |
center | baseline | stretch;
}
```



align-self

Modularization

Modular CSS

Modularization

Keep everything grouped in a logical topic.

- Modularization
  - Breaking a system into simple reusable & manageable components
- Every component in a modularized system has only one reason to exist.
- And the same reason to be changed
- Try to use to separate
  - Code in topics (component)
  - Us separate files for each topic
  - Try to reduce your code
    - use classes
    - use css variables

Why

Keep everything green and clean.

- CSS can be become quickly messy just like any other projects coded by any other language
- A big project should have some principles and guidelines
- You need to consider a team of programmers working on project together
- You have to understand your code after 6 months

Approaches

Keep an eye on the quality of your work.

- You need to have housekeeping as you code, before you code!
- Try to find common concepts in the code, promote them reusable component
- Start with the designing first
- Start with a design system
  - Define colors
  - Define fonts
  - Define layout
  - Define Components
- Always strive for quality

Techniques

How to achieve it.

- Use separate files for concepts or components
  - smacss
- Use @rules to import files
- Use a consistent naming convention (domain language, ubiquitous language)
- Use classes
- Create reusable components
- Use css variables

Additional Resources

- https://www.w3schools.com/css/css_positioning.asp
- https://developer.mozilla.org/en-US/docs/Web/CSS/position
- https://www.w3schools.com/css/css_boxmodel.asp
- https://csslayout.io/

Flexbox

Links & Pointers

- Theorie
  - https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties#align-items
  - https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox
  - https://css-tricks.com/snippets/css/a-guide-to-flexbox/
  - https://caniuse.com/#search=flex
- Exercise
  - https://github.com/Zmote/it-club-css-layouts
  - http://flexboxfroggy.com/
- More about the FlexBox (self-study)
  - https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Questions