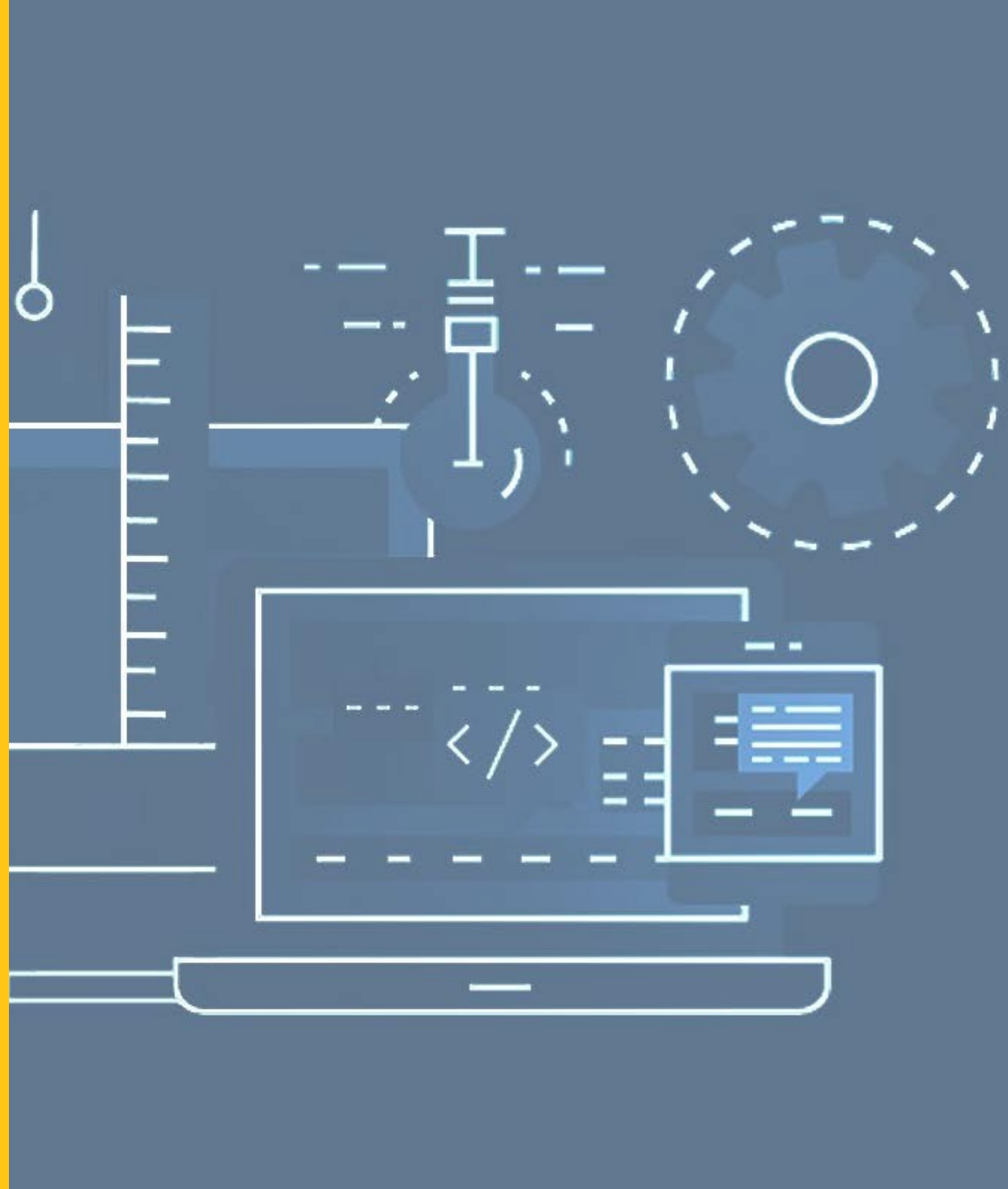
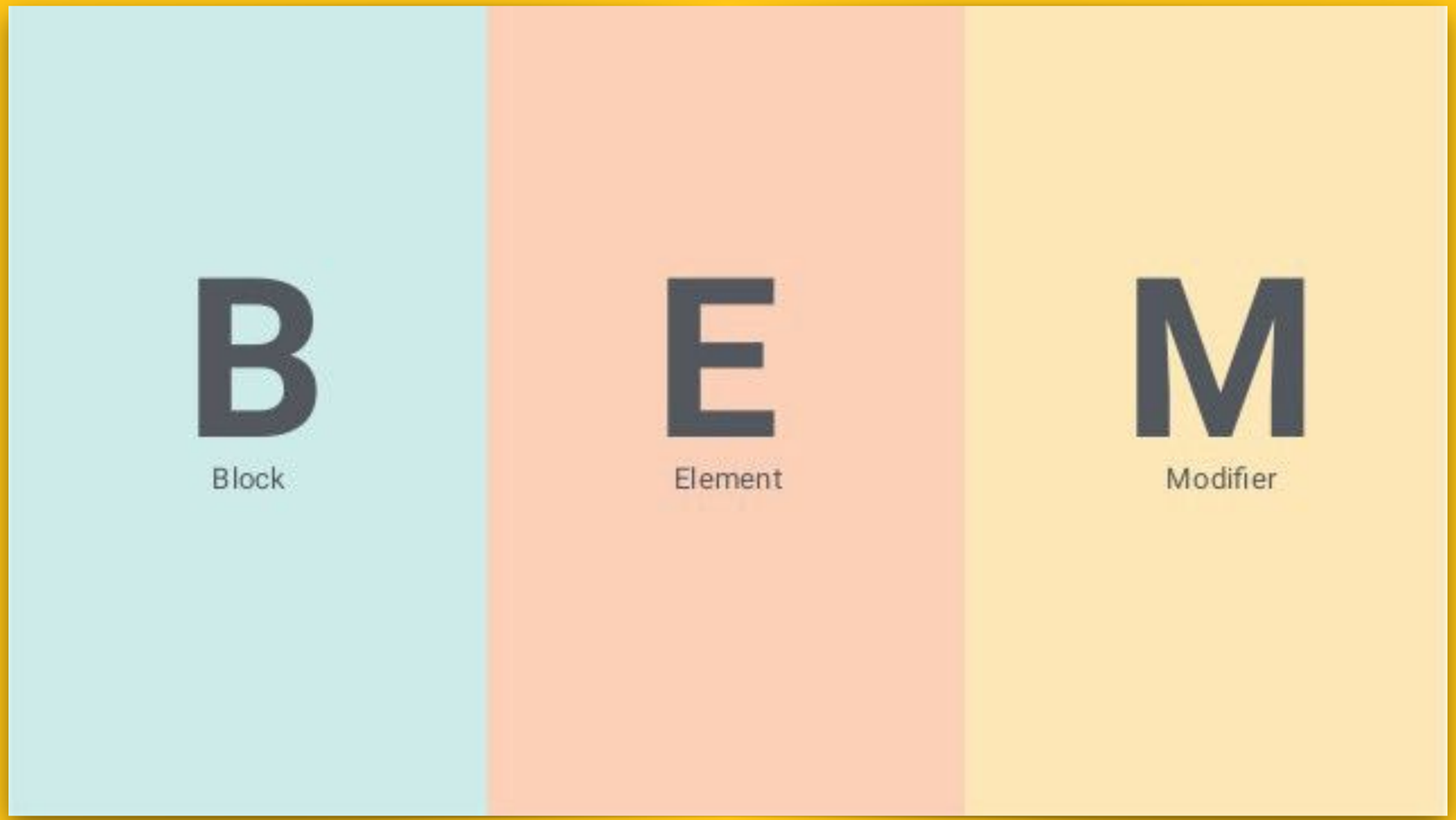
SHELLY GRAHAM, 05/05/2022

WEB DEV 3 SPRING 2022

Week5: BEM & Flexbox

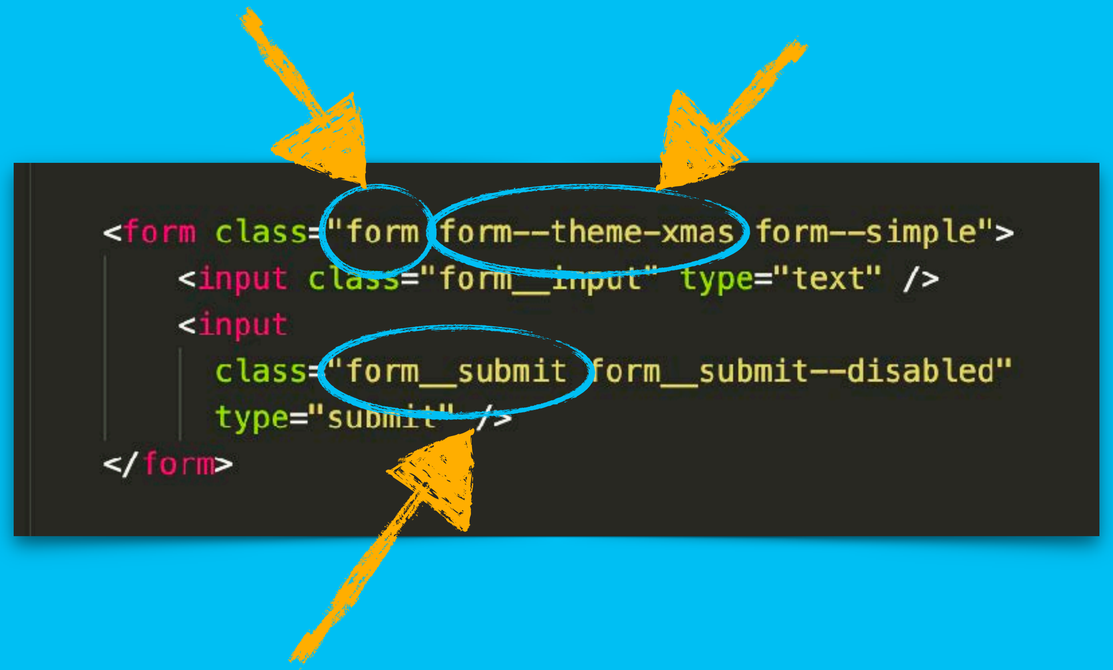


BEM



BEM

- Block Element Modifier
- Highly useful, powerful, and simple naming convention that makes your code easier to read and understand, easier to work with, easier to scale, more robust and explicit, and a lot more strict
- It’s easy, modular and flexible
- No additional library needed



Block

Standalone entity that is meaningful on its own.

Examples

`header` , `container` , `menu` , `checkbox` , `input`

Element

A part of a block that has no standalone meaning and is semantically tied to its block.

Examples

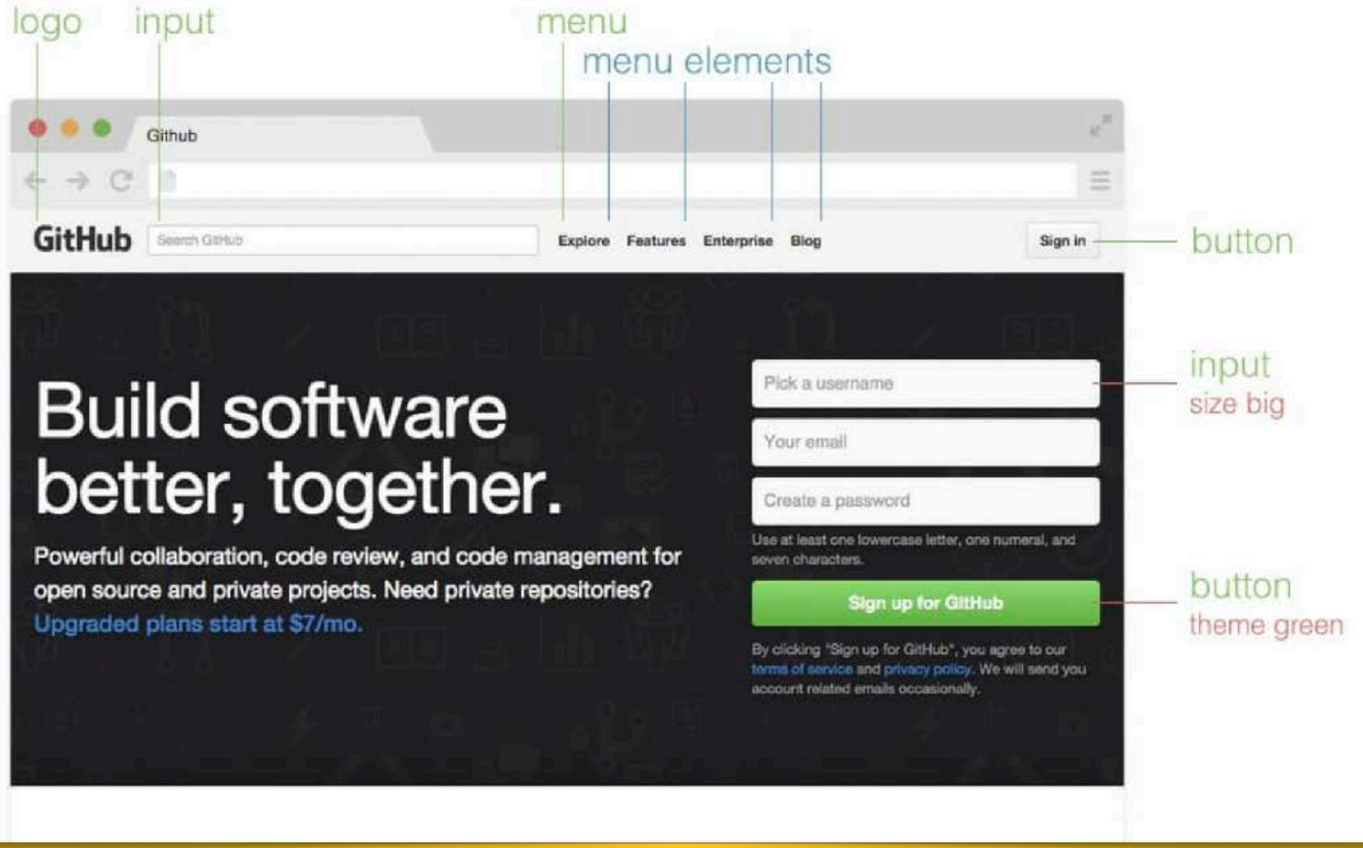
`menu item` , `list item` , `checkbox caption` , `header title`

Modifier

A flag on a block or element. Use them to change appearance or behavior.

Examples

`disabled` , `highlighted` , `checked` , `fixed` , `size big` , `color yellow`



BLOCK

- A standalone entity that is meaningful on its own
- While blocks can be nested and interact with each other, semantically they remain equal; there is no precedence or hierarchy
- Block names may consist of Latin letters, digits, and dashes
- CSS class is a short prefix for namespacing

```
<div class="block">...</div>
```

ELEMENT

- Part of a block and have no standalone meaning.
- Any element is semantically tied to its block.
- Element names may consist of Latin letters, digits, dashes and underscores.
- CSS class is formed as block name plus two underscores plus element name

```
<div class="block">
    ...
    <span class="block__elem"></span>
</div>
```

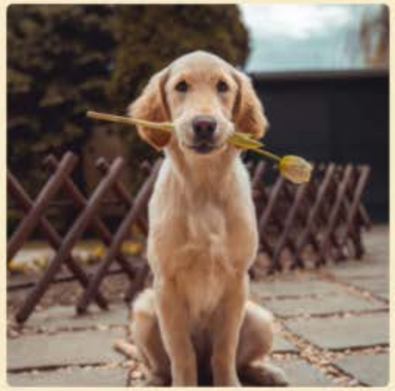
MODIFIER

- Flags on blocks or elements
- Use them to change appearance, behavior or state
- Modifier names may consist of Latin letters, digits, dashes and underscores. CSS class is formed as block's or element's name plus two dashes:
  - .block—mod
  - .block__elem—mod

```
<div class="block block--mod">...</div>
<div class="block
            block--size-big
            block--shadow-yes">
            ...
</div>
```

OBJECT FIT

fill



cover

contain

none

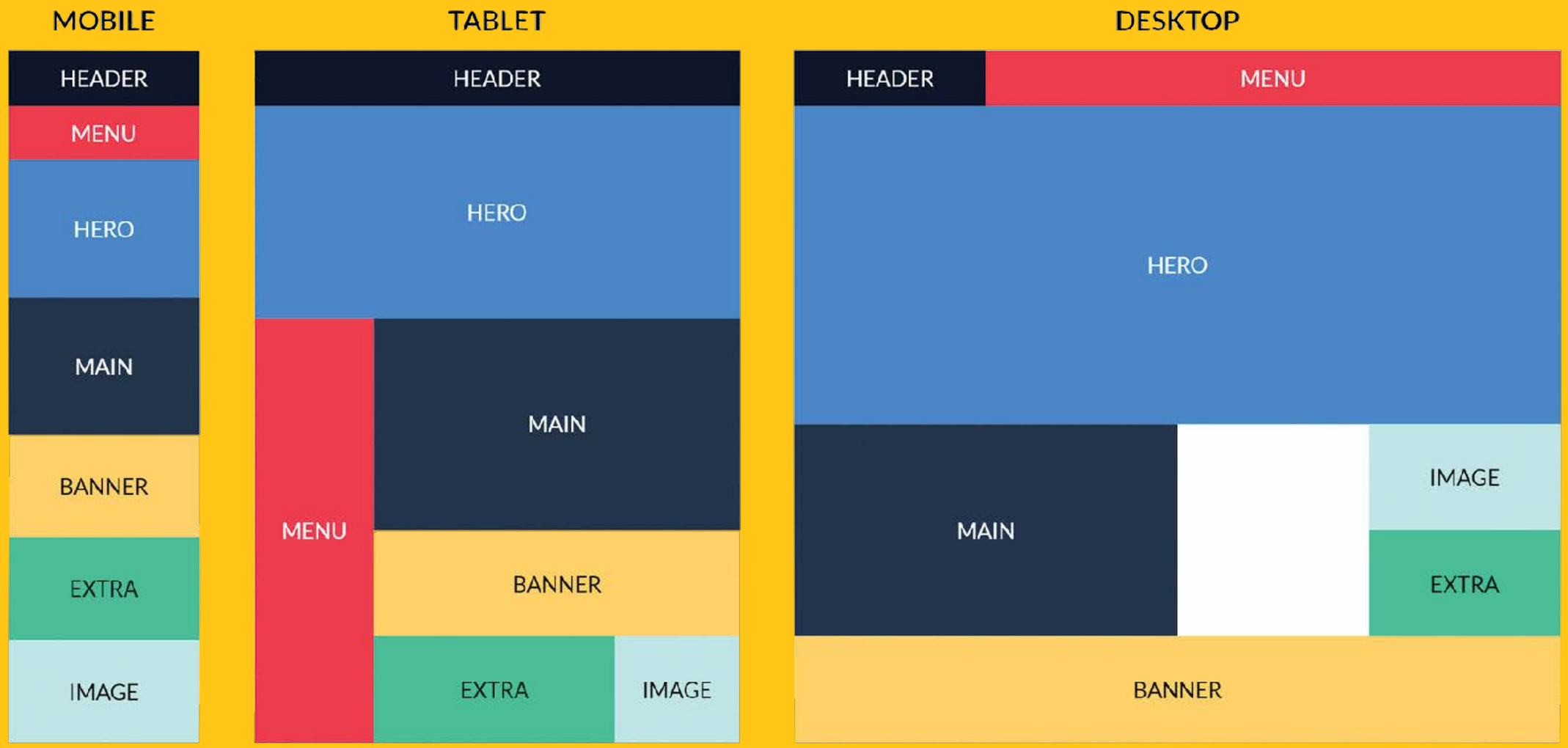
scale-down

OBJECT-FIT

- Let’s us treat images just like background images!
- It is set on the image itself, not the parent!
- You’d be using this while setting a fixed width/*height* on an image (usually in flexible values like vh or % but also px)
- none is similar to cover but keeps the dimensions of image intact. It crops sides to fit image
- scale-down is similar to contain but won’t grow the image larger than the original dimensions

```
.container {

    .item {
        object-fit: none
                    fill
                    contain
                    cover
                    scale-down;
        object-position: top|right|bottom|left
                         30% 40%
                         25vh
                         2em 40px;
    }
}
```

LAYOUTS


MOBILE
HEADER
MENU
HERO
MAIN
BANNER
EXTRA
IMAGE
TABLET
HEADER
HERO
MENU
MAIN
BANNER
EXTRA
IMAGE
DESKTOP
HEADER
MENU
HERO
MAIN
IMAGE
EXTRA
BANNER

THE EVOLUTION OF LAY OUTING

- Initially, websites were mainly text and images.
- You didn’t really mess with the HTML flow and were happy you could place some text on the internet!
- Designing beautiful layouts wasn’t even in the picture!
- But the internet evolved and with it it’s appearance...
- Introducing: The Table Layout!

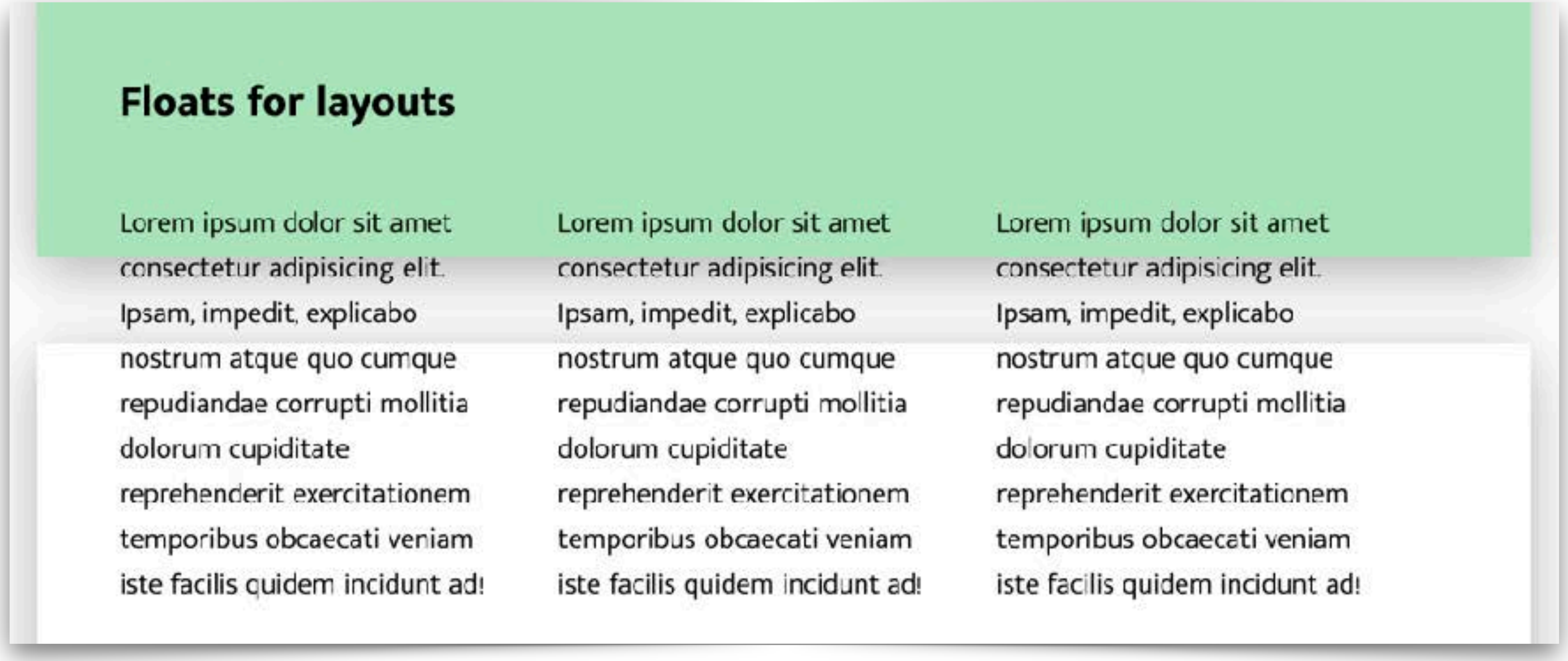
```
<table border="1" width="980" cellspacing="0" align="center">
    <tr>
        <td width="150" height="150">
            <img src="#" alt="test">
        </td>
        <td></td>
        <td></td>
    </tr>
    <tr>
        <td height="50" colspan="2">
            <table border="1" width="980" cellspacing="0"
            align="center">
                <tr>
                    <td height="50"></td>
                </tr>
            </table>
        </td>
        <td></td>
        <td></td>
    </tr>
</table>
```

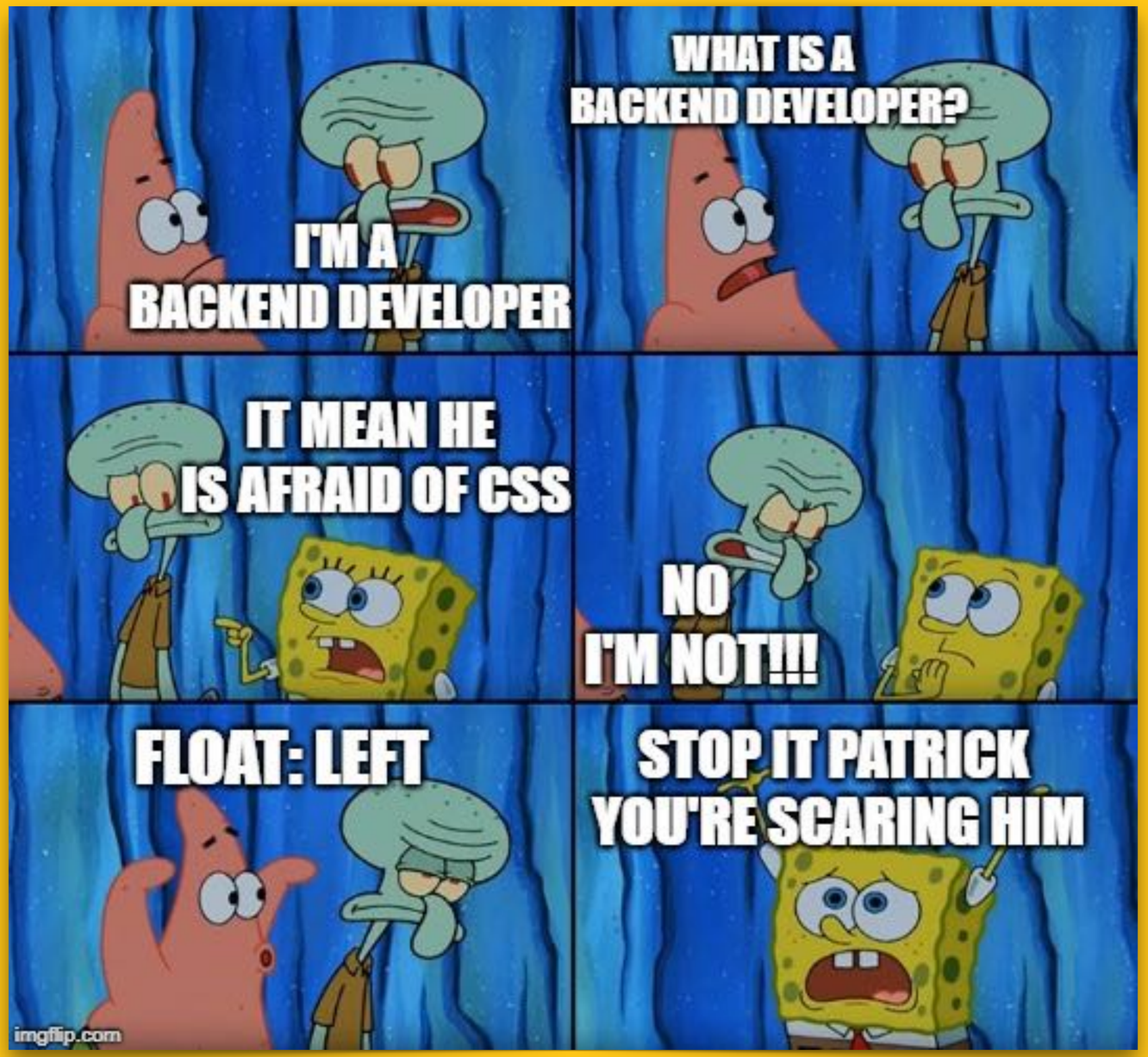
FROM TABLES TO FLOATS

- In a far away land...long before our time... developers used **tables** to create a layout with rows and columns...
- It was very complicated and tedious, so people started using ***floats*** for layout purposes because it made things a lot easier!
- Initially, floats were created to wrap text around images - not to layout entire pages!
- And since floats weren't meant to be used for layouts, stuff went wrong left and right...

FLOATS

- We can use float on multiple items and they will wrap next to each other and behave like flex: row-wrap in Flexbox
- But if you use floats you lose the height of block elements!
- In order to fix this, people used clearfixes
- The clearfix would be added to the parent element to add the lost height back in with a pseudo element
- By using floats, we would intentionally break the HTML flow to behave the way we want it to but along the way we had to use fixes to fix the things that happened due to us breaking it in the first place - I already have a headache!

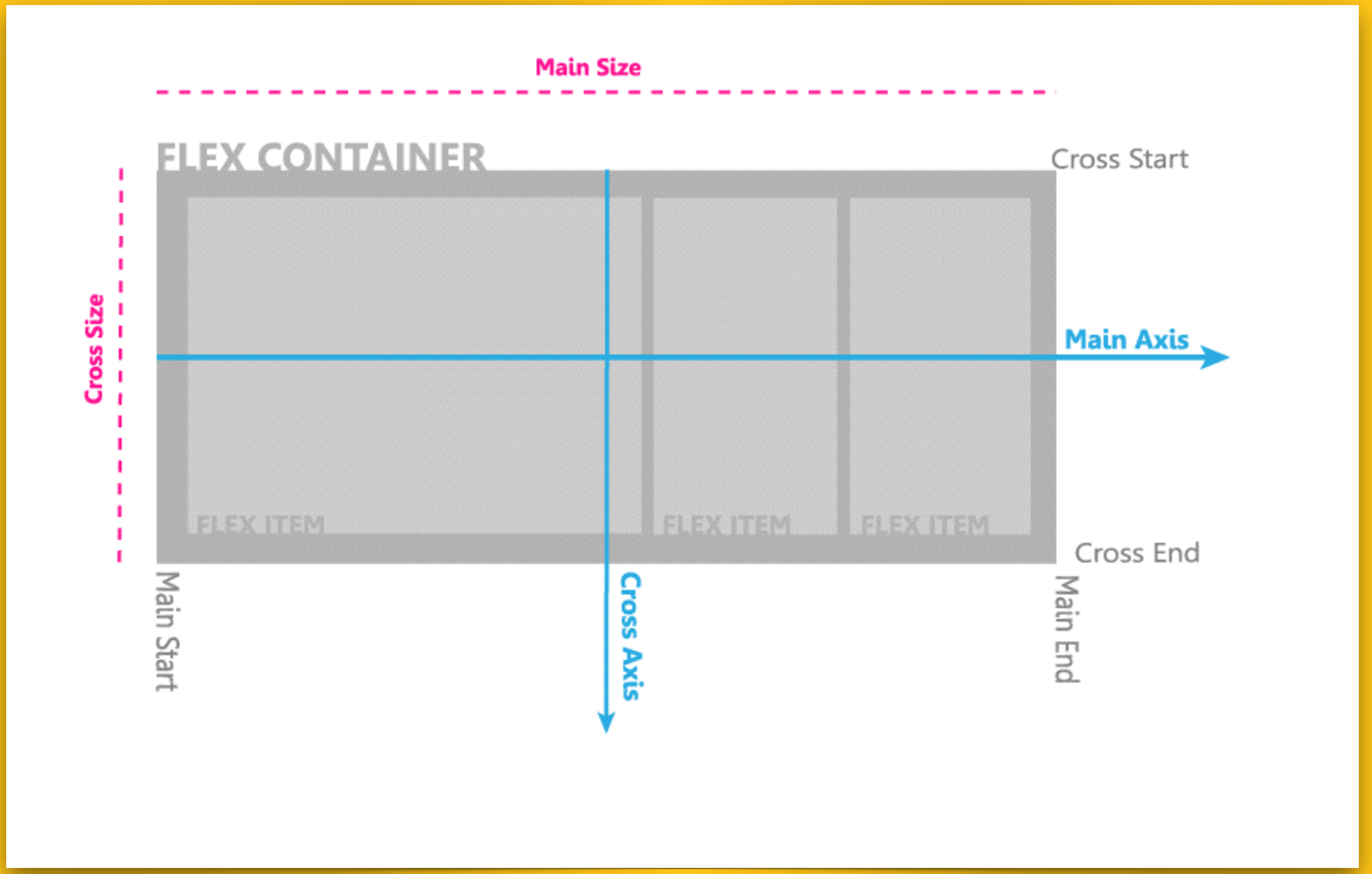



I'M A
BACKEND DEVELOPER
WHAT IS A
BACKEND DEVELOPER?
IT MEAN HE
IS AFRAID OF CSS
NO
I'M NOT!!!
FLOAT: LEFT
STOP IT PATRICK
YOU'RE SCARING HIM
imgflip.com

FROM FLOATS TO FLEXBOX

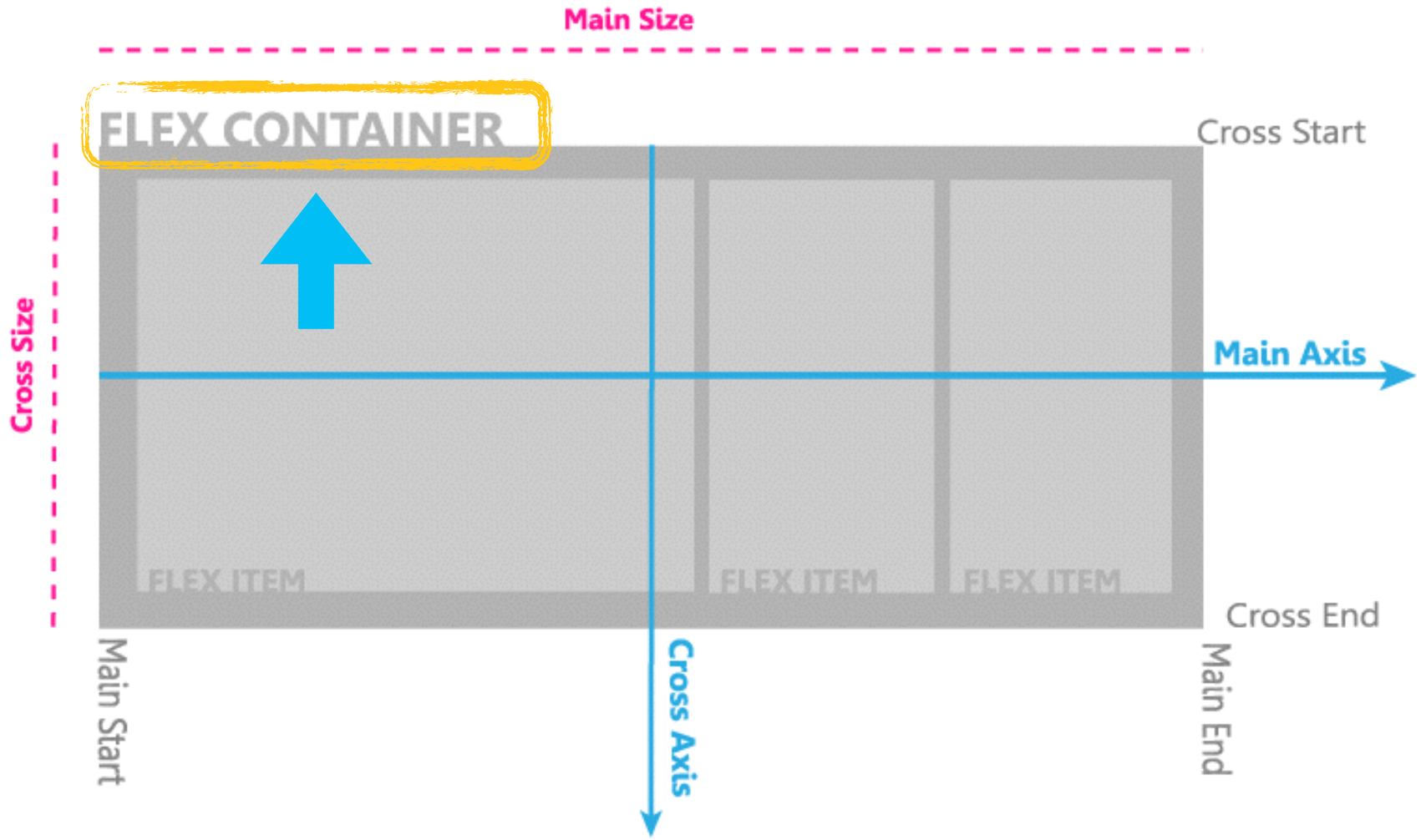
- Flexbox was the first attempt in CSS to give us a tool to create layouts
- It was created so we wouldn't have to use floats anymore and break every site
- It changes the natural HTML block element flow
- But in reality, it was created to help us create rows OR columns!
- Floats were used directly on the element you wanted to float itself whereas Flexbox needs a parent container to tell the flex elements within how to behave
- Flexbox is by default responsive so content will shrink and grow based on screen size

FLEXBOX


Main Size
FLEX CONTAINER
Cross Start
Cross Size
Main Axis
FLEX ITEM
FLEX ITEM
FLEX ITEM
Cross End
Main Start
Cross Axis
Main End

FLEXBOX

- In order to use Flexbox, you need:
  - a flex container aka parent and
  - flex item(s) aka direct children
- The behavior of the children is dictated by the parent
- Floats, clears and vertical-align have *no effect* on flex items!



FLEX CONTAINER

The Parent Container

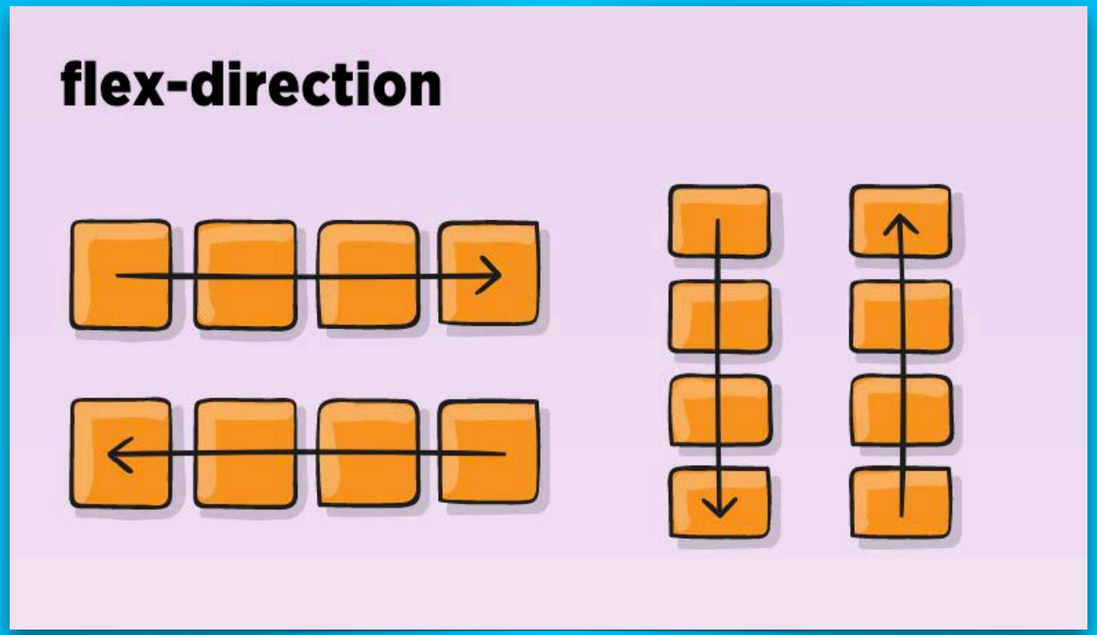
1. FLEX CONTAINER / PARENT

- Flex-direction
- Flex-wrap
- Flex-flow
- Justify-content
- Align-items
- Align-content
- New: Gap

```
.container {
    display: flex; /* or inline-flex */
}
```

FLEX-DIRECTION

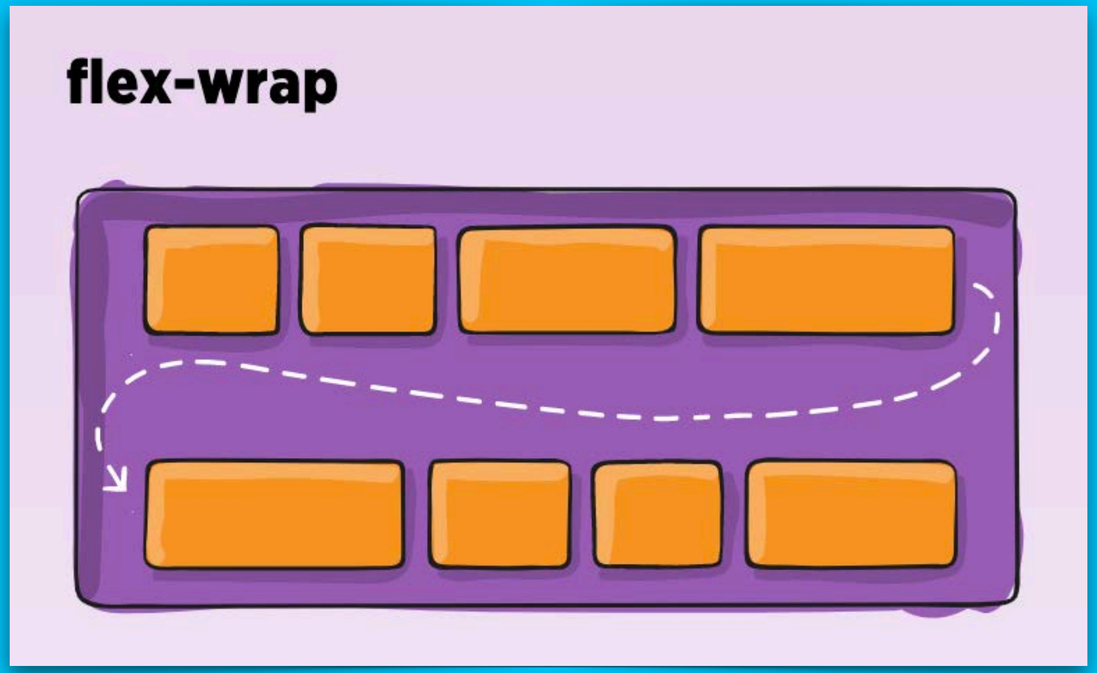
- Default is set to row, so all items will try to sit next to each other instead of stacking on top one another
- The block element still remains for the flex-children but is extended over the entire flex container
- Row replaces float: left
- Row-reverse replaces float: right
- Column: basically forcing it to behave “normal” by stacking children
- Direction changes are really helpful for navigation bars in media queries




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

FLEX-WRAP

- Flex-items will automatically adjust to the content that's inside them, but will eventually spill out of container if too many flex elements fill up the container
- Default: nowrap - but better would be a default of flex-wrap: wrap;
- Items will get as small as possible if wrap is *not* turned on
- If flex-wrap is not turned on, it will *ignore* the width set by a child element!




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

FLEX-FLOW

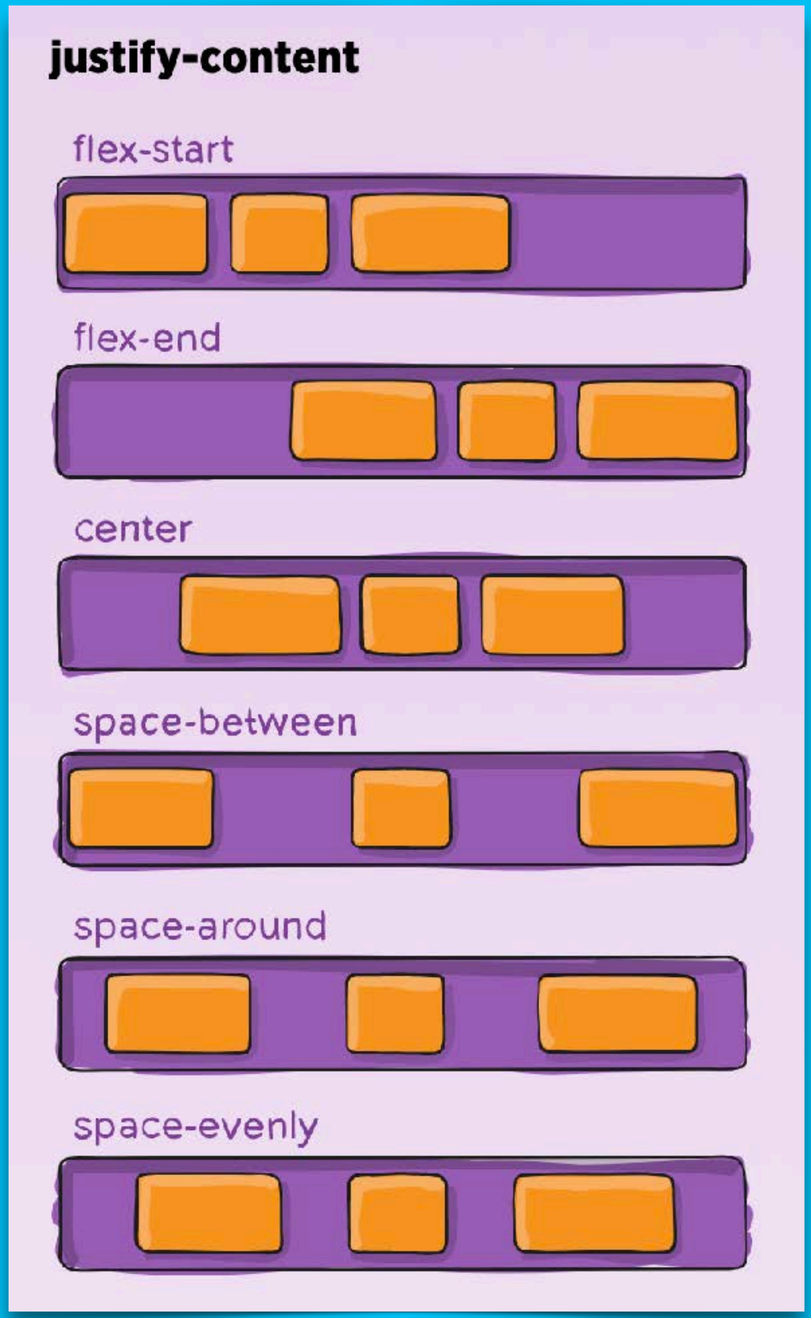
- Shorthand for:
  - flex-direction
  - flex-wrap
- Default:
  flex-flow: row nowrap;

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

    // flex-direction: column;
    // flex-wrap: wrap;
}
```

JUSTIFY-CONTENT

- Defines the alignment along the main axis
- Main axis can be reversed with flex-direction!
  - flex-direction: row = X axis is main axis
  - flex-direction: column = Y axis is main axis
- Default is flex-start

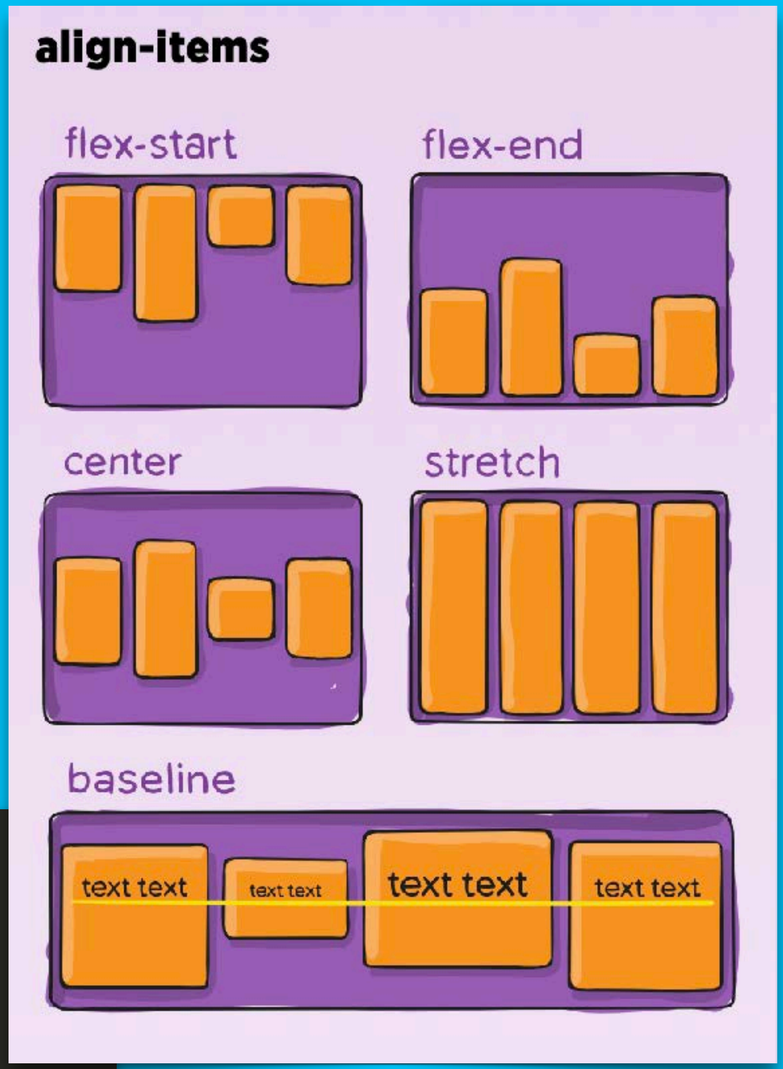
```
.container {
    display: flex;
    justify-content: flex-start
                     flex-end
                     center
                     space-between
                     space-around
                     space-evenly;
}
```



ALIGN-ITEMS

- Defines the alignment along the cross axis
- Default is stretch: content stretches across the entire container width or height
- If taken off of default, they will shrink to minimum size
- Baseline to align items all on one line/same height

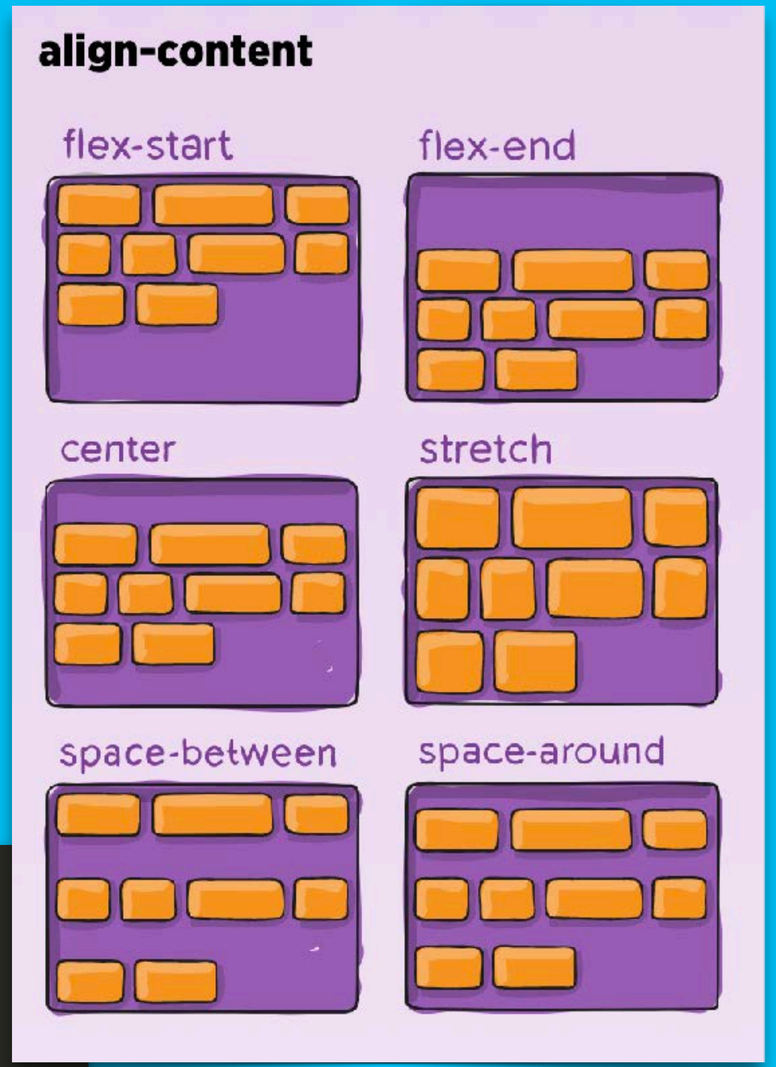
```
.container {
    display: flex;
    align-items: stretch
                 flex-start
                 flex-end
                 center
                 baseline
                 first baseline
                 last baseline;
}
```



ALIGN-CONTENT

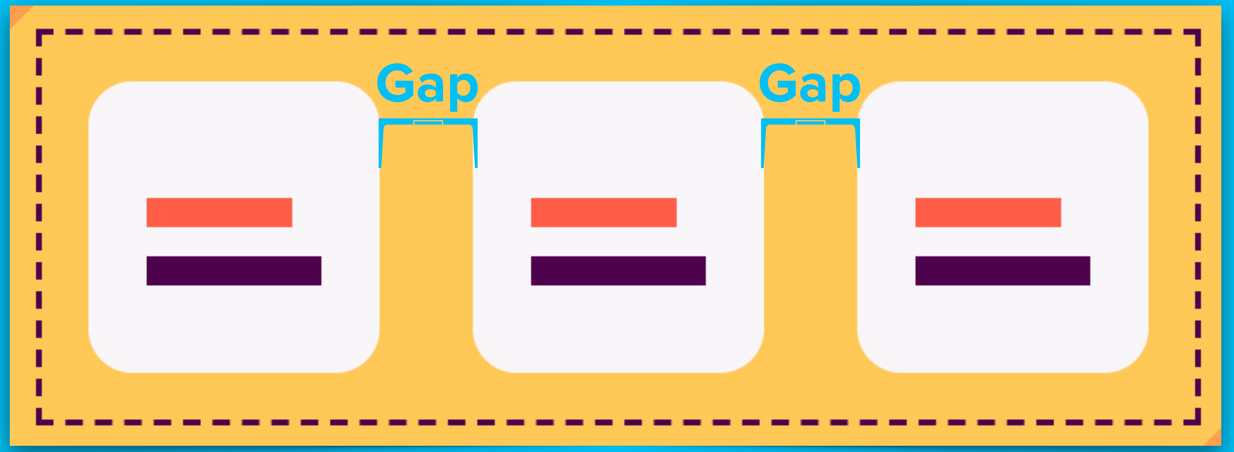
- Works like justify-content but not on items but entire flex content
- Only works if you have multiple lines of content and flex-flow is set to wrap or wrap-reverse

```
.container {
    display: flex;
    align-content: flex-start
                   flex-end
                   center
                   space-between
                   space-around
                   space-evenly
                   stretch;
}
```

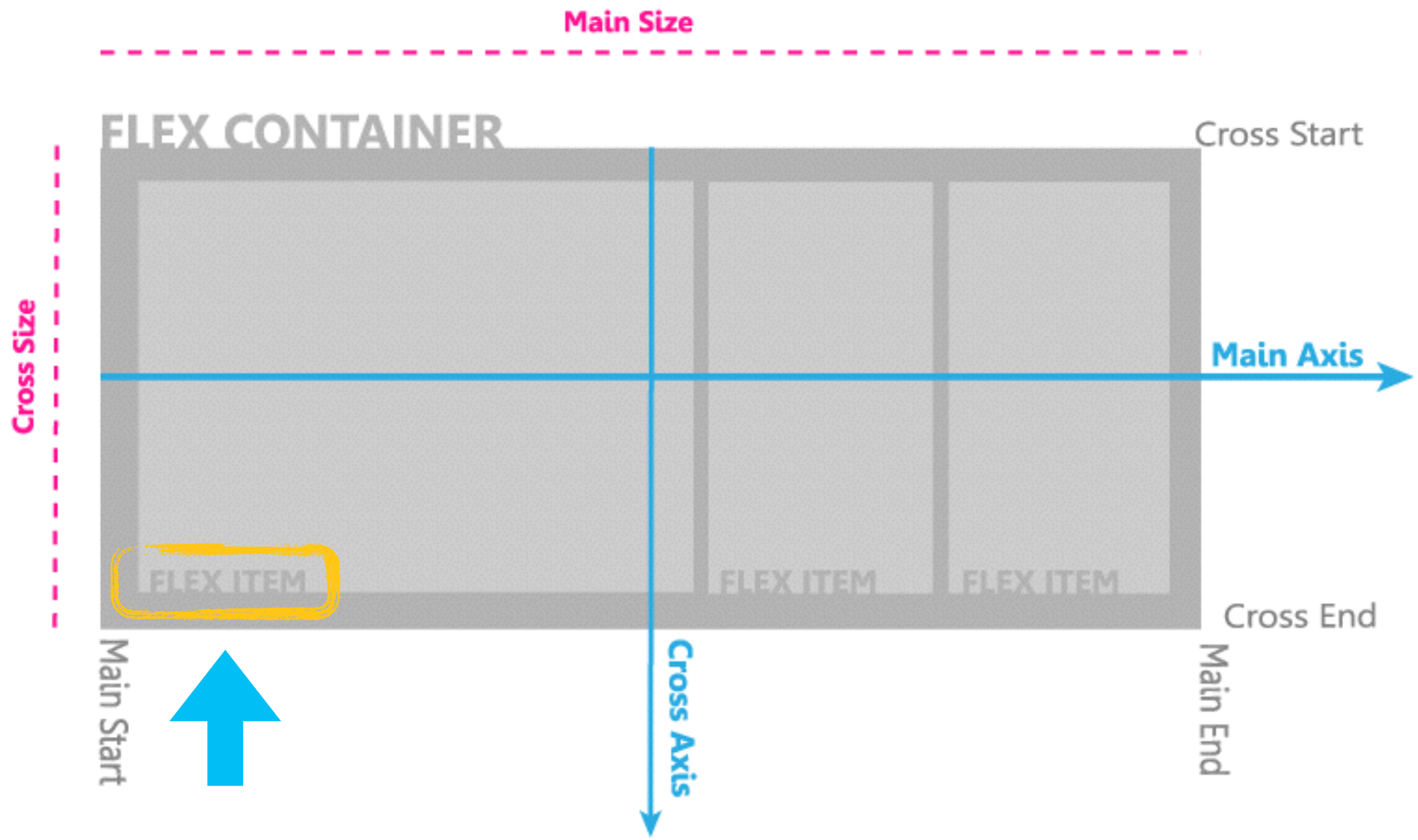


GAP

- You can now use Gap with Flexbox!
- Set the gap in the flex container/ parent
- It will only act *between* the flex element, not at start or end to not cause weird margins
- Also works with media queries if you switch the flex direction from column to row!
- Use gap instead of margin to create space between flex items




```
.container {
    display: flex;
    // row-gap: 1em;
    // column-gap: 1em;

    gap: 1em;
    gap: 10px 15%;
}
```



FLEX ITEMS

The Child Element

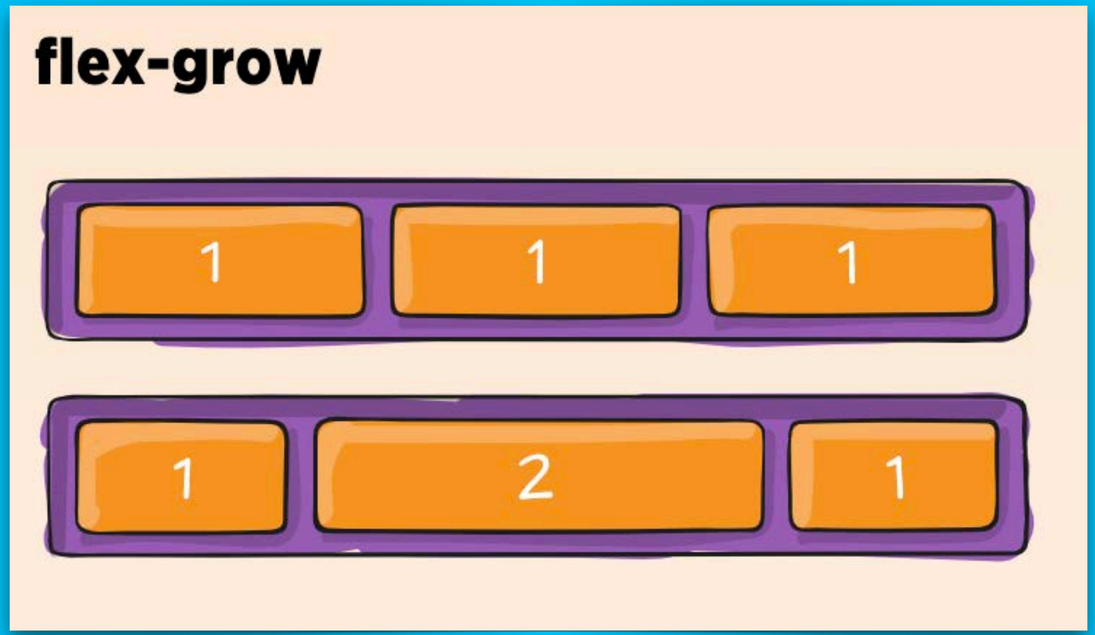
2. FLEX ITEMS / CHILDREN

- Flex-grow
- Flex-shrink
- Flex-basis
- Flex
- Align-self
- Order

```
.container {
    display: flex;

    .item {
        flex-grow:
        flex-shrink:
        flex-basis:
        flex:
        align-self:
        order:
    }
}
```

FLEX-GROW

- Default is 0
- Flex-grow dictates what amount of the available space inside the flex container the item should take up
- This defines the ability for a flex item to grow if necessary.
- Unitless value that serves as a proportion




```
.container {
    display: flex;

    .item {
        flex-grow: 2;
    }
}
```

FLEX-SHRINK

- Default is 1
- Flex-shrink defines the ability for a flex item to shrink if necessary
- Negative numbers are invalid
- Unitless value that serves as a proportion

```
.container {
    display: flex;

    .item {
        flex-shrink: 3;
    }
}
```

FLEX-BASIS

- Default is auto
- Defines the default size of an element before the remaining space is distributed
- Use this instead of giving your flex-items a width
- It’s the ideal width, behaves like max-width but will get smaller OR bigger if it has to due to resizing —> depending on what grow and shrink are set to
- If you want all columns to always have the same size, regardless of amount of text inside, give all children a flex-basis: 100%
- This works because flex-shrink is turned on by default so they are ok to shrink but they really want to be as big as possible (100%)

```
.container {
    display: flex;

    .item {
        flex-basis: 250px;
        flex-basis: 100%;
    }
}
```

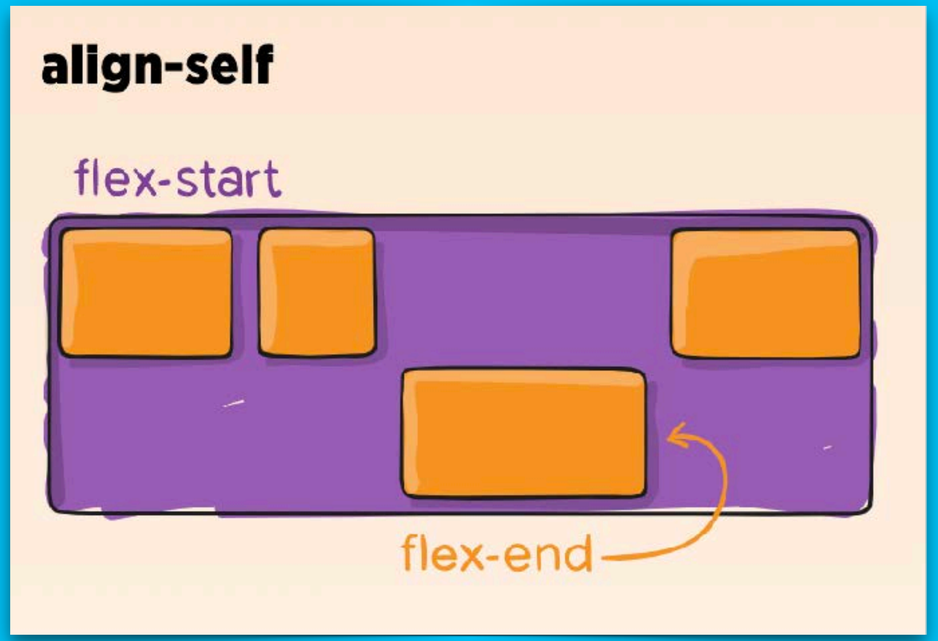
FLEX

- Shorthand for:
  - flex-grow
  - flex-shrink
  - flex-basis
    —> in that order!
- Default:
  flex: 0 1 100%;

```
.container {
    display: flex;

    .item {
        flex: 1 1 250px;
    }
}
```

ALIGN-SELF

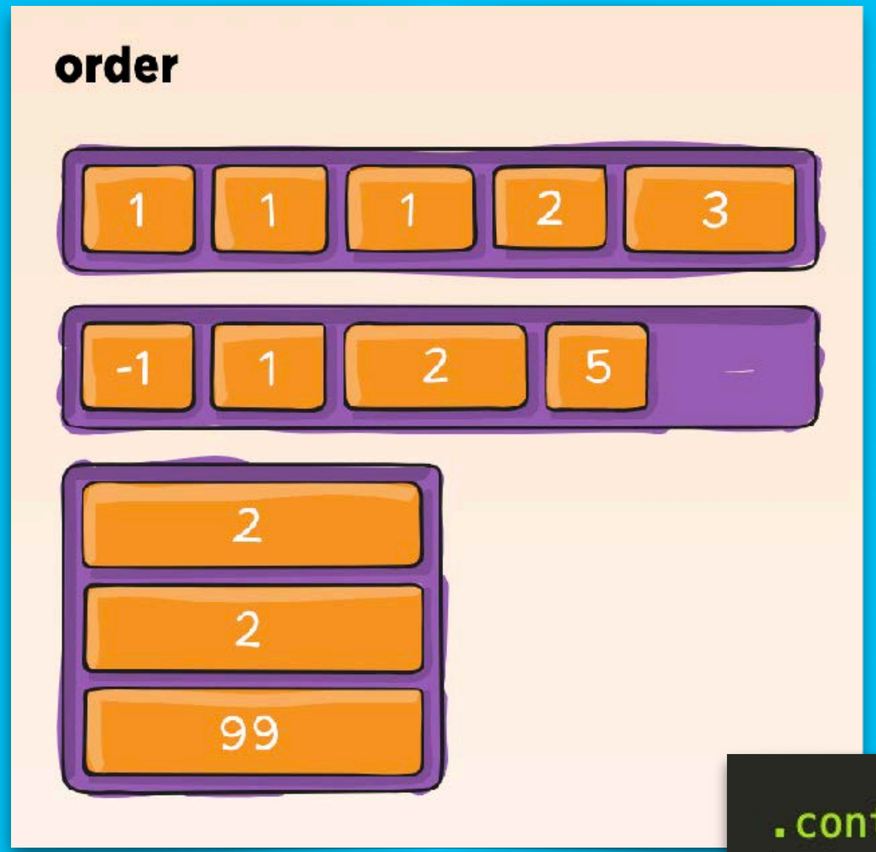
- Default is auto
- Aligns said item on secondary axis and overwrites the default alignment




```
.container {
    display: flex;

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

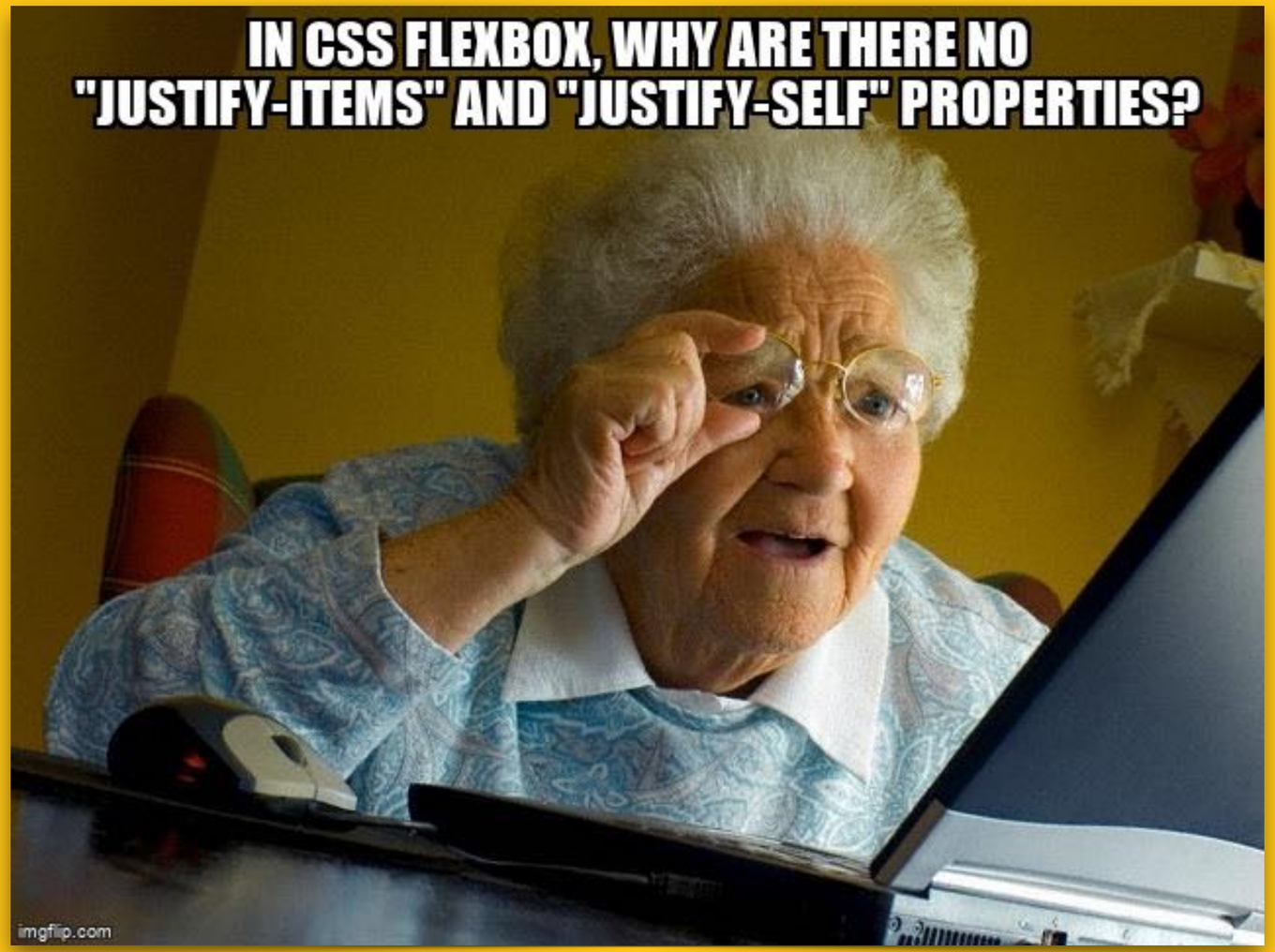
ORDER

- Default is 0
- order: -1 puts an individual item to the FRONT
- order: 1 puts an individual item to the END



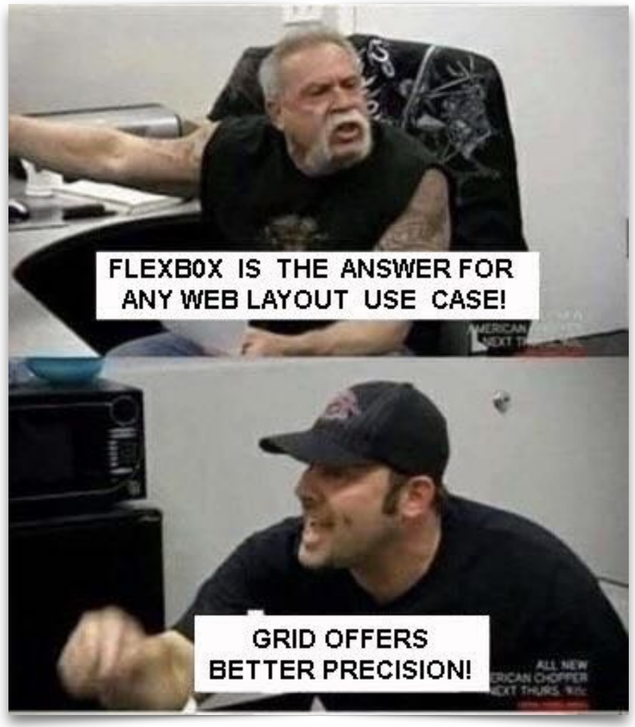

```
.container {
    display: flex;

    .item {
        order: -1;
        order: 4;
    }
}
```

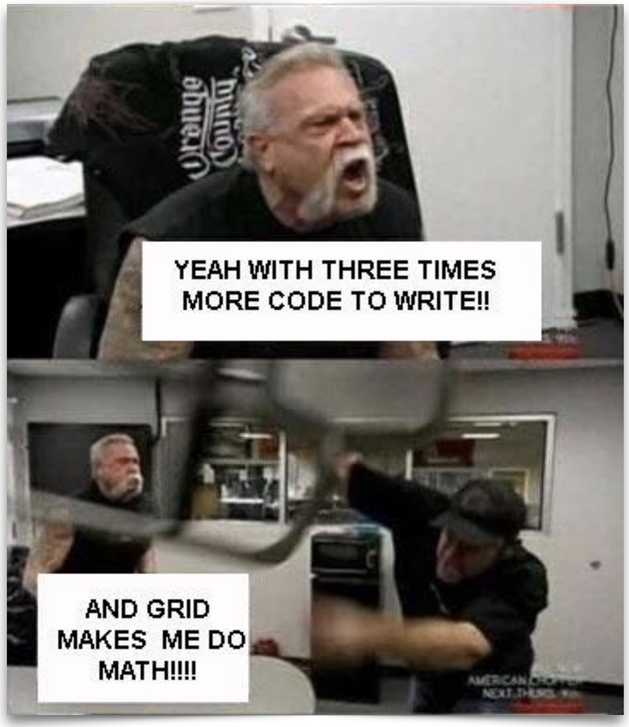
SOMETHING IS MISSING...

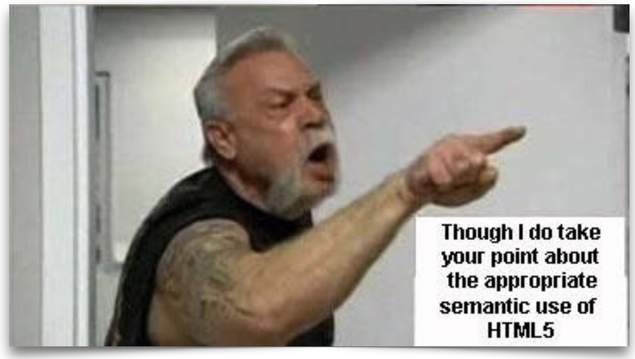


ROW OR COLUMN - YOU CAN'T HAVE BOTH

- Flexbox lets you layout either in row OR column form.
- You can't define a specific row and column amount but are rather forced to make the content fit in either row or column formats







QUESTIONS?

RESOURCES

- Not using BEM: https://codepen.io/michellejames/pen/bGBLGXX?editors=1100
- Using BEM: https://codepen.io/michellejames/pen/zYoRxvz
- Using BEM even better: https://codepen.io/michellejames/pen/jOVZErW?editors=1100
- Object Fit: https://codepen.io/michellejames/pen/abBOpoR?editors=1100

HOMEWORK

- Use the provided Adobe XD file and recreate the layout using Flexbox.
- Export all the images you need from the file and optimize them with the tools and techniques we learned in last week's class.
- As usual, the page must be fully responsive. I'd also like to see usage of Sass and BEM.
- Take a look at the provided transitions and recreate them. Also, add at least 1 other transition to the website to make it “your own”!
- Submit a link to your GitHub repo.

FIN