# CSS Selector Review
# Layout with CSS & Flex

Lecture 4

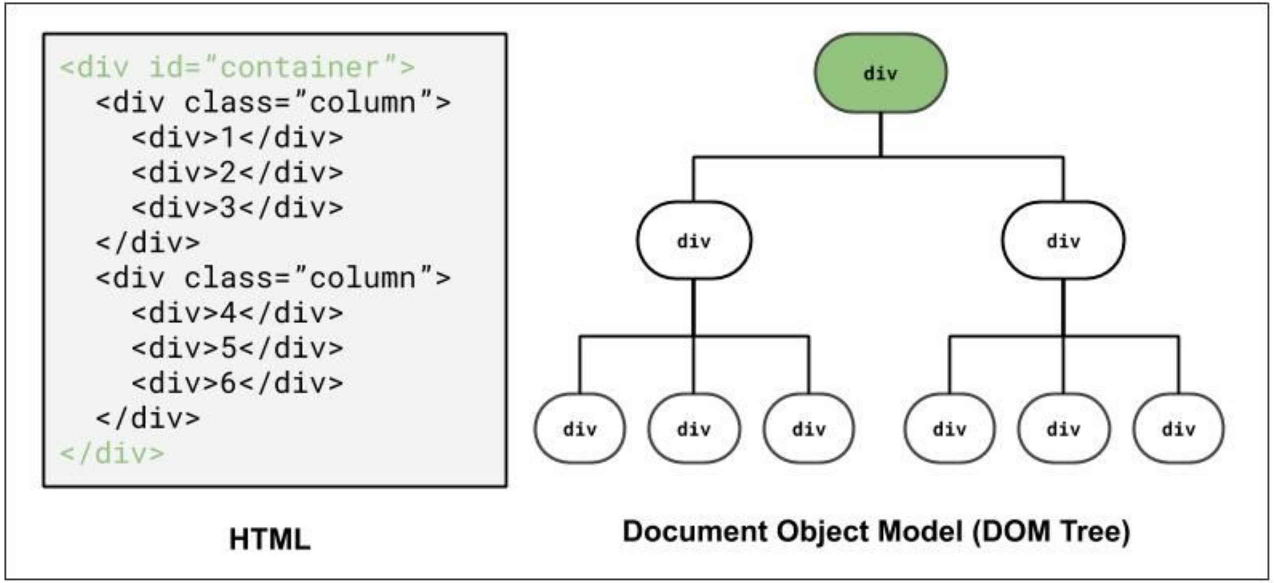
## CSS Selectors Review

| Classification | Description | Example |
|---|---|---|
| Type selector | Selects an element type | `p` |
| Class selector | Selects all elements with the given class attribute value | `.koala-face` |
| ID selector | Selects the element with the unique id attribute value | `#scientific-name` |

We can also combine selectors:

- `.bordered` selects all elements with the bordered class
- `h2.bordered` selects only h2 elements with the bordered class

## DOM and Selectors: Q1

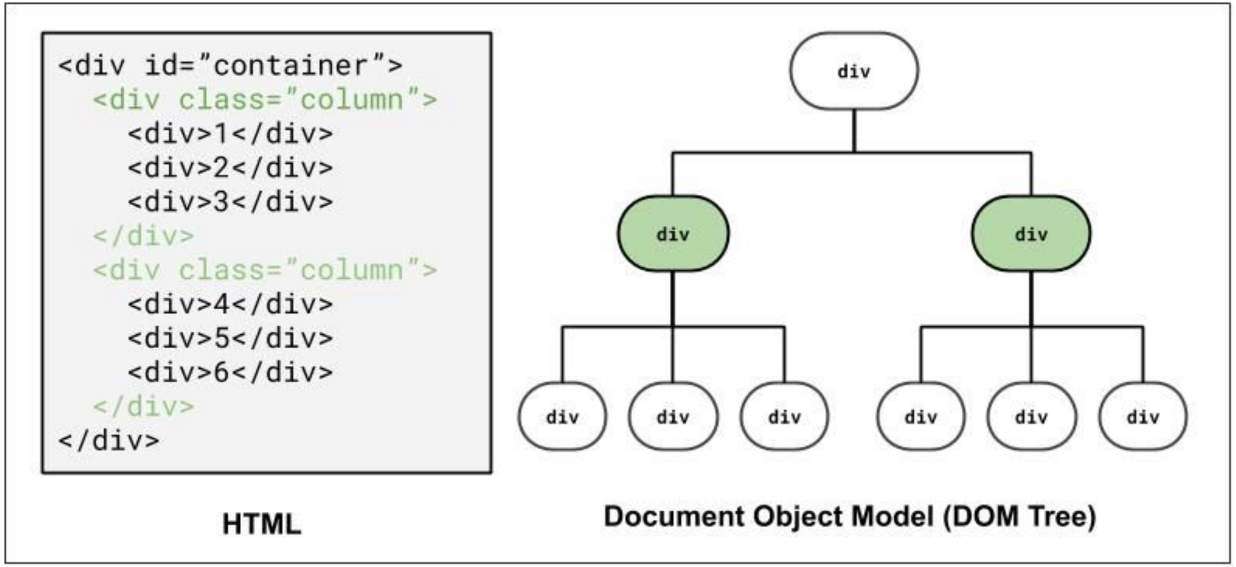


```
<div id="container">
  <div class="column">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>
  <div class="column">
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>
</div>
```

How to select the highlighted elements?

Using ID selector: `#container`

## DOM and Selectors: Q2

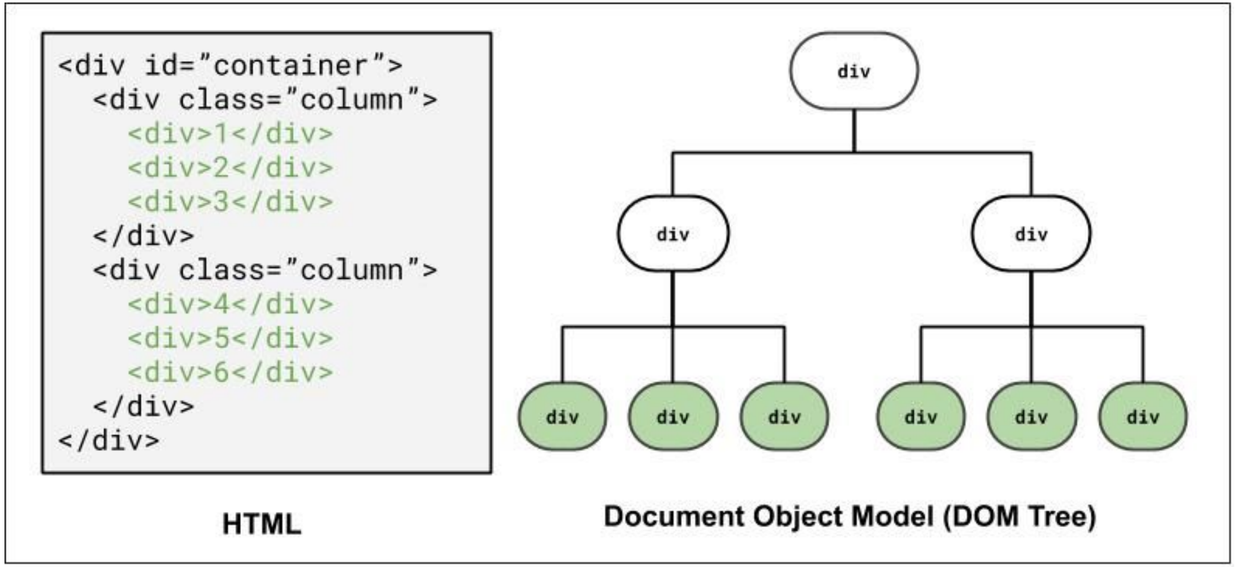


```
<div id="container">
  <div class="column">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>
  <div class="column">
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>
</div>
```

**How to select the highlighted elements?**

Using class selector: `.column`
Using combinator selector: `#container > div`

## DOM and Selectors: Q3

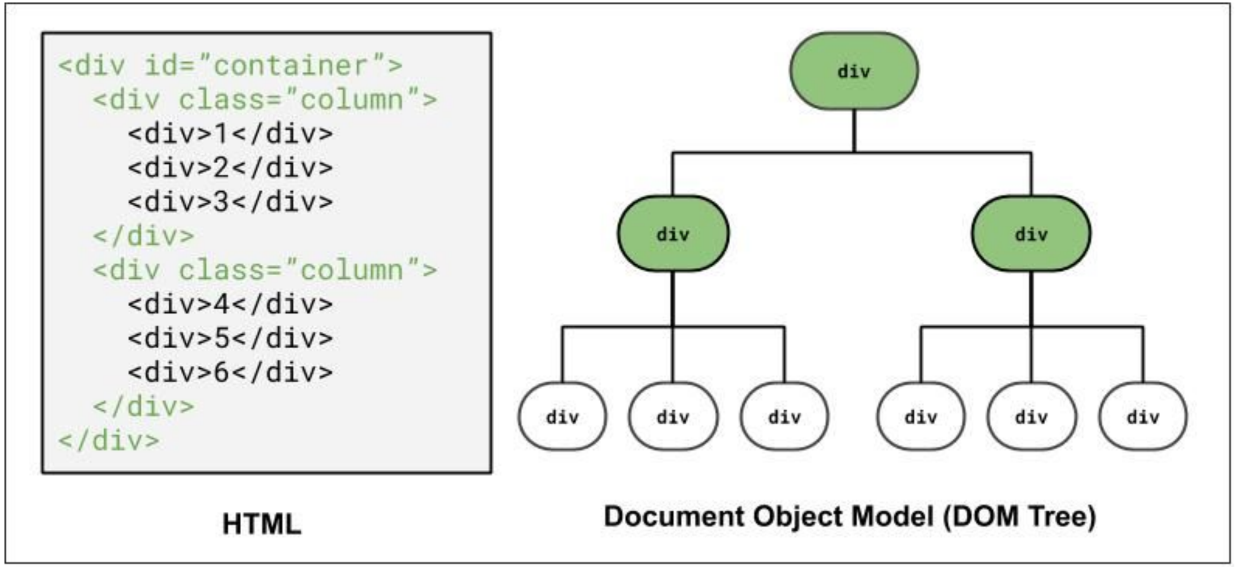


```
<div id="container">
  <div class="column">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>
  <div class="column">
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>
</div>
```

**How to select the highlighted elements?**

Using class and descender combinator selector: `.column div`
Using a class and child combinator selector: `#container > div > div`

## DOM and Selectors: Q4



```
<div id="container">
  <div class="column">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>
  <div class="column">
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>
</div>
```

**How to select the highlighted elements?**

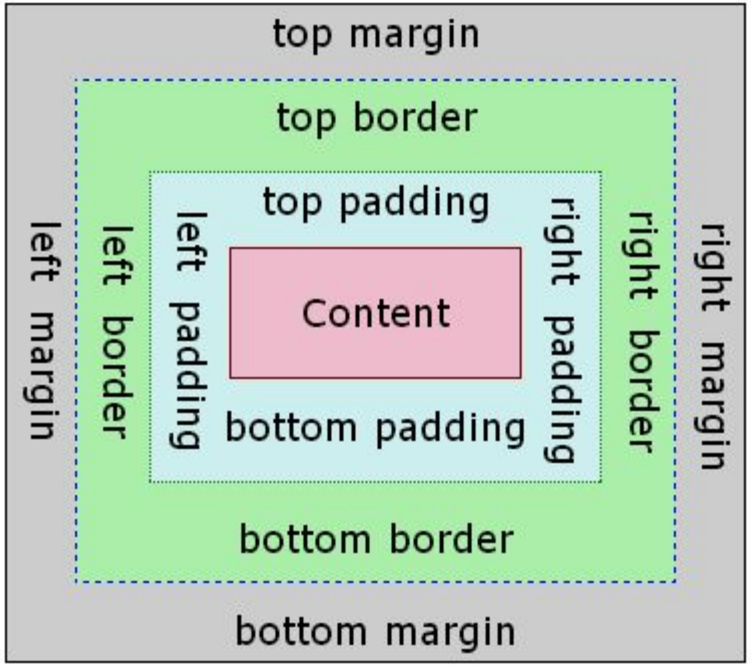
Grouping elements: `#container, .column`

## The Box Model

**Margin:** (*outside*) space between different elements

**Border:** (*optionally visible*) line that separates elements

**Padding:** (*inside*) space between element content and border



## Why page layout is important

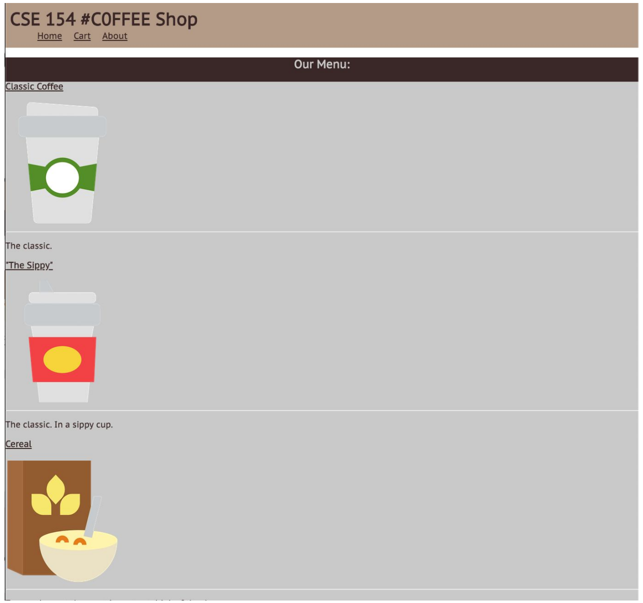
- **Example 1**: Click here for an example of poor HTML tags, layout, and accessibility (try resizing the page). The “old days” of layout.
  - Inspect?: table, table, table, table…

- **Example 2:** This is the official HealthCare.gov webpage from 2018, where all kinds of users rely on for health care information.
  - Some visually-impaired users need larger font sizes on the screen.
  - What happens to the search bar when you increase the font size?

- As a user, have you ever left a website (that may be useful) because of the layout or accessibility?
- There's a lot of very cool research on verifying page layout!

# The Goal: Clean Layout, Responsive Design

Today, we'll learn the fundamentals of various layout techniques to go from the left initial product (no layout CSS) to the final product (to the right, link TBD)



CSE 154 #COFFEE Shop

Home Cart About

Our Menu:

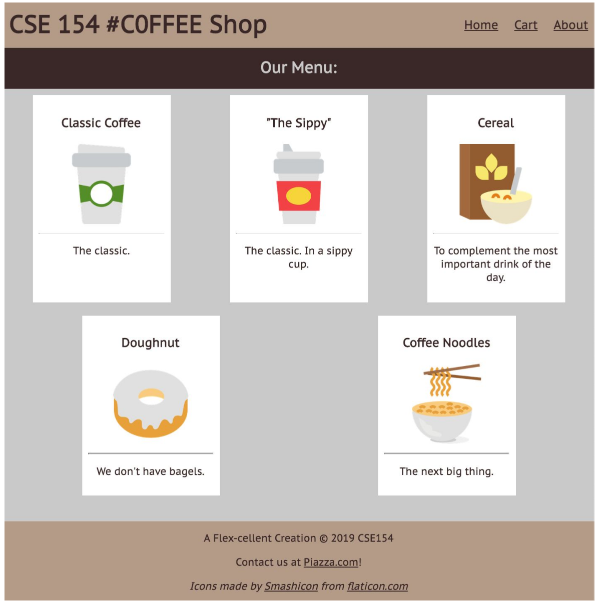
Classic Coffee

The classic.

"The Sippy"

The classic. In a sippy cup.

Cereal



CSE 154 #COFFEE Shop

Home Cart About

Our Menu:

Classic Coffee

The classic.

"The Sippy"

The classic. In a sippy cup.

Cereal

To complement the most important drink of the day.

Doughnut

We don't have bagels.

Coffee Noodles

The next big thing.

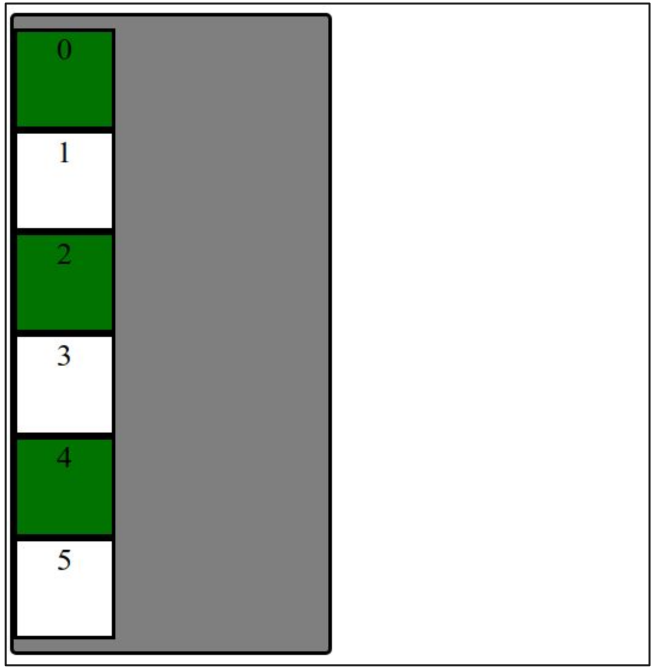
A Flex-cellent Creation © 2019 CSE154

Contact us at Piazza.com!

*Icons made by Smashicon from flaticon.com*

## Starting with Building Blocks

```
<div id="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```



## Why are we playing with boxes?

When learning CSS layout, you'll find there are many ways to layout your pages.
"Boxes" are great to practice with for comparing different layout strategies and better understanding the box model.
We are also working with text inside of each div to demonstrate block vs. inline layout.

In practice, it's useful to:

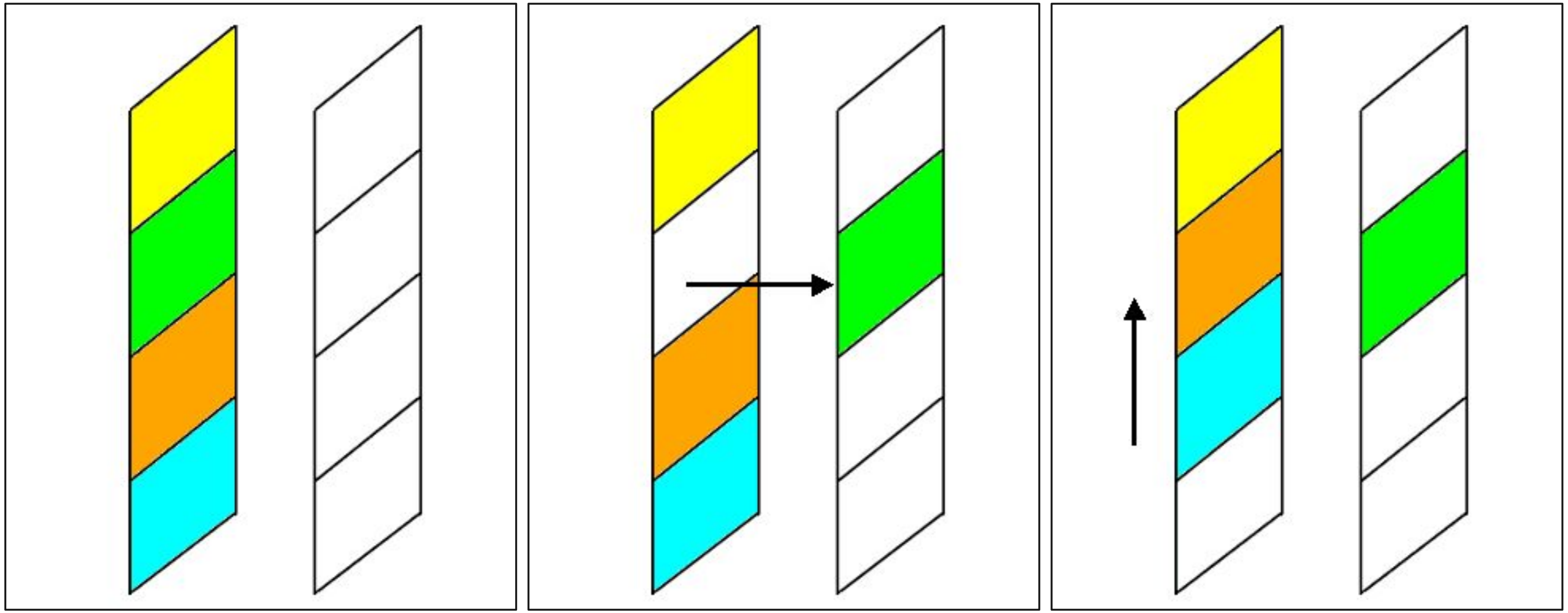
1. Treat page elements as boxed regions to figure out the page layout
2. Then focus on more specific CSS styling

## (Older) Layout Method: Floating Elements

A way to remove elements from the normal document element flow, usually to get other elements to "wrap" around them.

Can float `left`, `right` (no `center`)

## An analogy: Page Layers as Sheets of Paper



## Example with Float

There is a blue block has the `float` CSS property set to `left`. This tells the browser to give the element as much space as it needs, and then start bringing the next content up from below and fill in remaining space.

See the Pen - Box Float by @mehovik on Codepen.

Add `overflow: auto;` to make the parent of a floating element contain the floating element.

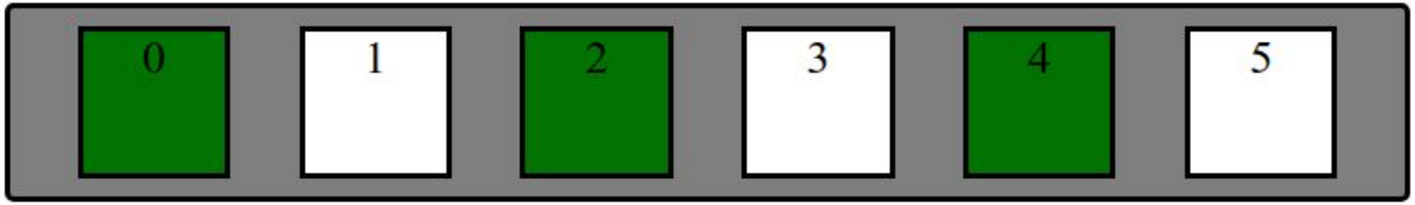
## And that's a wrap on float :)

This is not an exhaustive introduction to `float`. There is **SO** much more to learn about `float` as well as some other good use cases for `float` as well. However, our focus for layout will be on the box model and using `flex`. If you'd like to learn more about float post on Ed, ask in OHs or go to WPL.

## Distributing Boxes Evenly in a Container

How could we distribute boxes across box container evenly (equal space between each box)?.

... what should we do about the margins of the boxes?
... what value do we put?
... how many screen sizes will we try on?



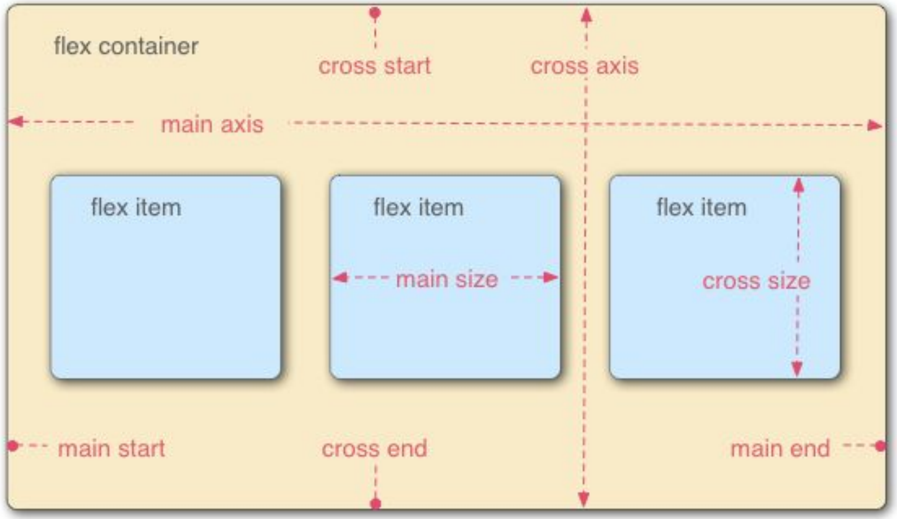
## `display` Property

The display property specifies the display behavior (the type of rendering box) of an element. The four values you most often will see are:

- `inline`: Displays an element as an inline element, spanning the width/height of its content. Any height and width properties will have no effect.
- `block`: Displays an element as a block element. It starts on a new line, and takes up the width of the parent container.
- `none`: The element is completely removed.
- `flex`: Displays an element as a block-level flex container.
- `grid`: Displays elements in a 2-dimensional grid.

## Flexbox

- Flexbox is a set of CSS properties for aligning block level content.
- Flexbox defines two types of content - "containers" and "items".
- Anything directly nested inside of a flex container becomes a flex item.
- Various properties on the container can be set to determine how its items are laid out.



## Basic properties for the flex container

```
display: flex;
```

- makes an element a "flex container", items inside automatically become "items" - by default, starts as a row

```
justify-content: flex-end; (flex-start, space-around,...)
```

- indicates how to space the items inside the container along the main axis

```
align-items: flex-end; (flex-start, center, baseline,...)
```

- indicates how to space the items inside the container along the cross axis

```
flex-direction: row; (column)
```

- indicates whether the container flows horizontally or vertically (default row)

```
flex-wrap: wrap; (no-wrap, ...)
```

- indicates whether the container's children should wrap on new lines

# Basic properties for the flex container

There are also cases when you will need to add flex properties to flex *items* rather than the flex *container*

**`flex-grow: <number>`**

- Defines a proportional value to determine whether a flex items can grow (what amount of the available space inside the container it should take up).

**`flex-basis: 20%; (3em, 50px,...)`**

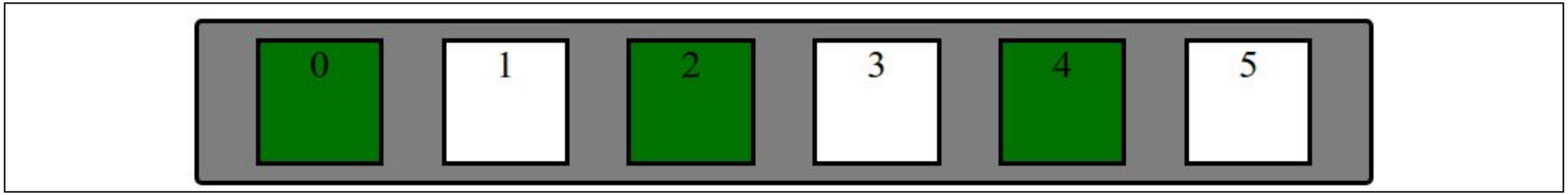
- indicates the default size of an element before the extra space is distributed among the items

**`align-self: flex-end; (flex-start, center, stretch,...)`**

- indicates where to place this specific item along the cross axis

## Back to the boxes

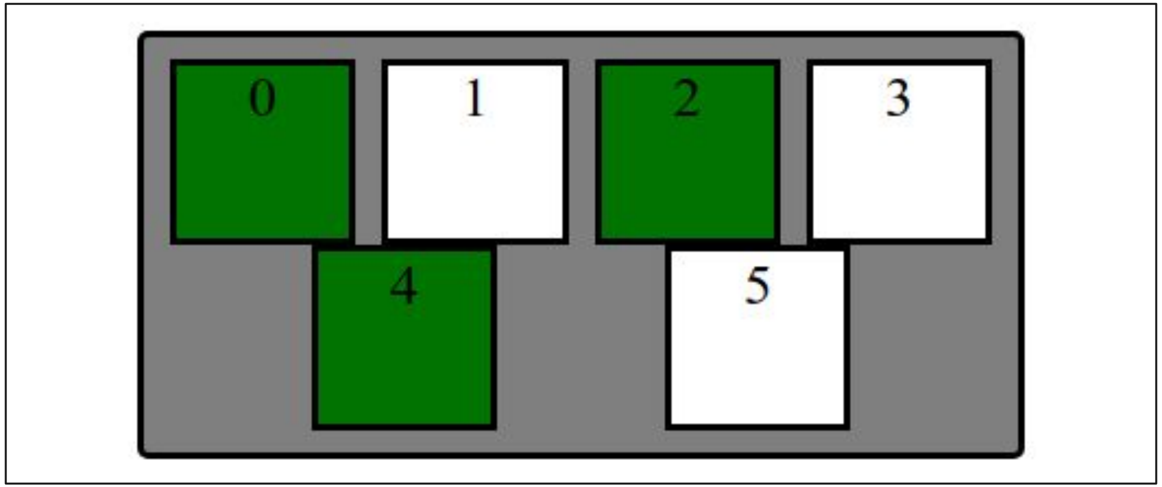
Exercise: distribute boxes across box container evenly (equal space between each box) using flex.



solution

## Exercise: Responsive Page Layout - Wrapping

Set boxes to wrap when box container gets too small in the browser so that they keep their square widths (what happens when you shrink the browser width in the previous exercise?).

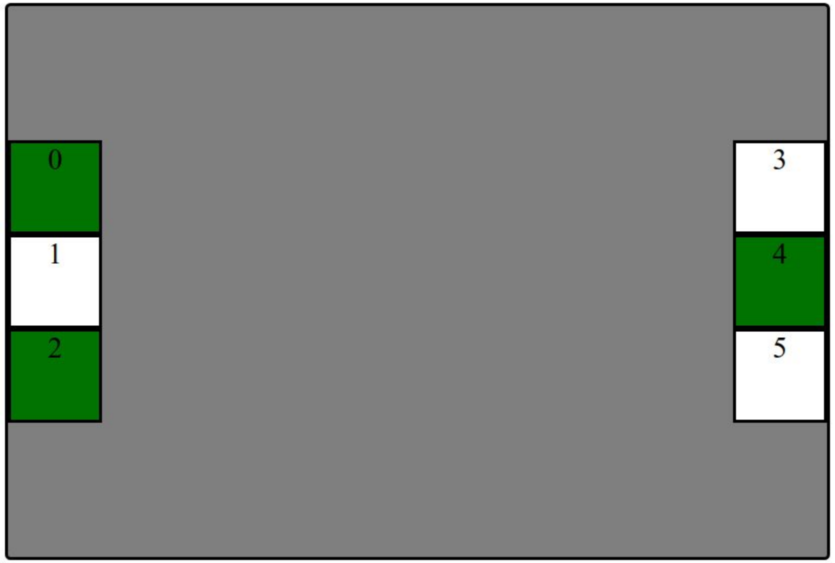


solution

## Extra Practice (for flex masters): more fancy stuff

Layout boxes into two 3-box columns using flex (use screenshot with given details). Note: don't rely too much on previous CSS solutions - you'll need to change the HTML slightly as well to get the columns grouped)

- In the HTML, make boxes grouped in two 3-box columns (hint: add a class to both groupings called "`column`").
- Change height of `#container` to `500px` and center the columns vertically
- Distribute the two columns on both left/ends of the `#container`.



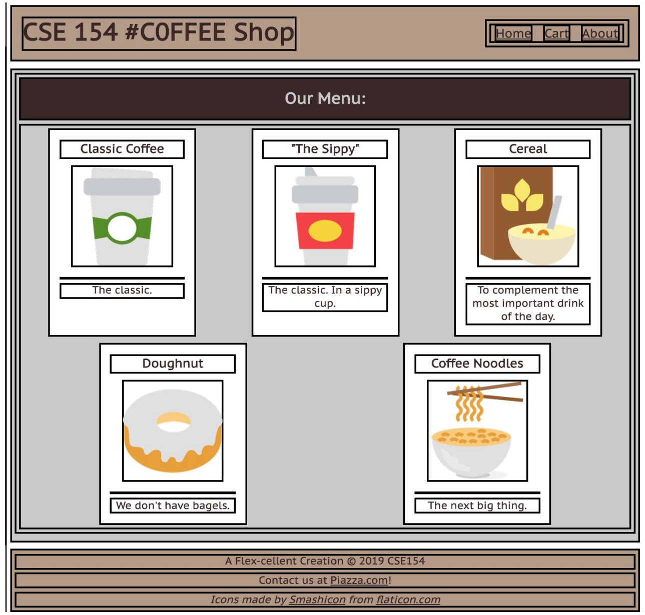
solution

## From Boxes to a “Real” Example

How can we use these different layout methods in pages with components like header, main, footer? What about side-by-side sections? Inline navigation with lists?

## What columns & rows exist in the cafe page?

What are the parent "containers" distributing items in a row/column?


CSE 154 #COFFEE Shop

Home Cart About

Our Menu:

Classic Coffee

The classic.

"The Sippy"

The classic. In a sippy cup.

Cereal

To complement the most important drink of the day.

Doughnut

We don't have bagels.

Coffee Noodles

The next big thing.

A Flex-cellent Creation © 2019 CSE154

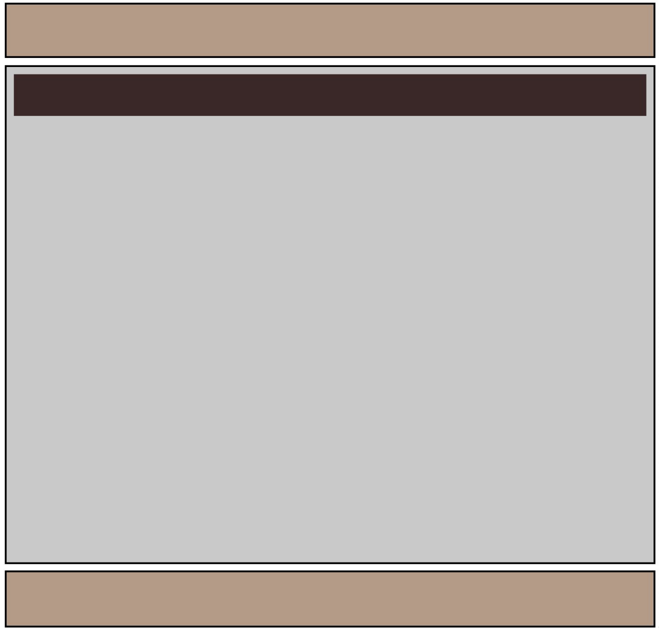
Contact us at Piazza.com!

*Icons made by Smashicon from flaticon.com*

- `body` (column with 3 children)
- `#top-bar` header (row with 2 children)
- `#item-container` (row with 5 children)
- `footer` (column with 3 children)

We'll take an "outside-in" approach, starting with the body

## Body Layout: A Column

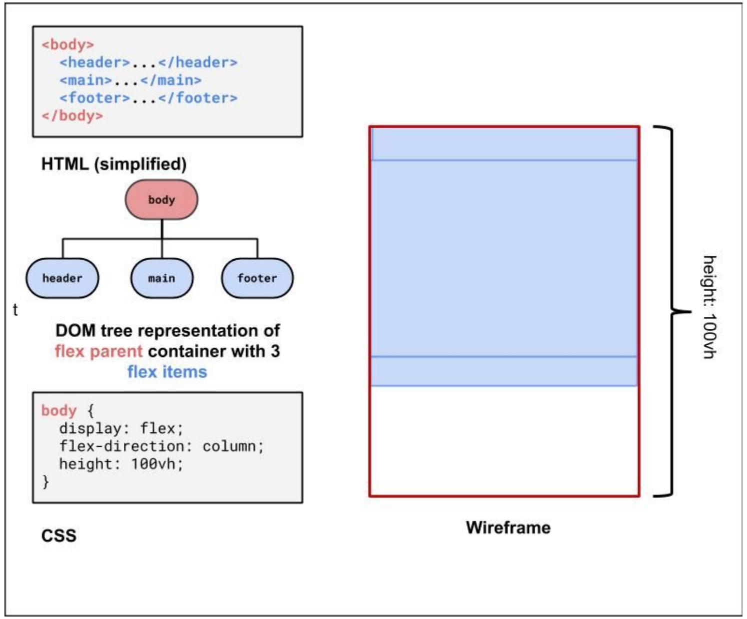


## Figuring out the Flex Layout

- For the body, we know we want a column.
- We already get a column layout from the default block display for `header`, `main`, `footer`.
- But by default, these elements will have a height defined by their contents. This will result in whitespace at the bottom of the page.
- We can use flex to control the distribution of the body's children to fill the entire page!

# Making the Body a Flex Column

Many page layouts desire a full viewport height (vh) with a footer at the bottom.. To set the body to be 100% of the viewport height, use the vh size unit.



```
<body>
  <header>...</header>
  <main>...</main>
  <footer>...</footer>
</body>
```

HTML (simplified)

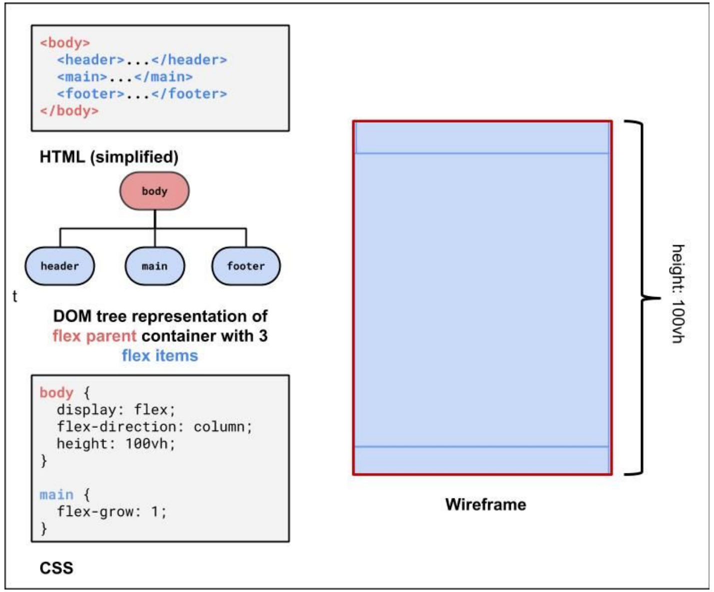
DOM tree representation of flex parent container with 3 flex items

```
body {
  display: flex;
  flex-direction: column;
  height: 100vh;
}
```

CSS

## Using flex-grow with column page layout

Next, we can use `flex-grow: 1` on the child element of the body flex container to have that child fill any remaining whitespace (the default for flex-grow for all items is 0). Let's make the `main` child fill the rest of the whitespace of the parent



```
<body>
  <header>...</header>
  <main>...</main>
  <footer>...</footer>
</body>
```

HTML (simplified)

DOM tree representation of flex parent container with 3 flex items

```
body {
  display: flex;
  flex-direction: column;
  height: 100vh;
}

main {
  flex-grow: 1;
}
```

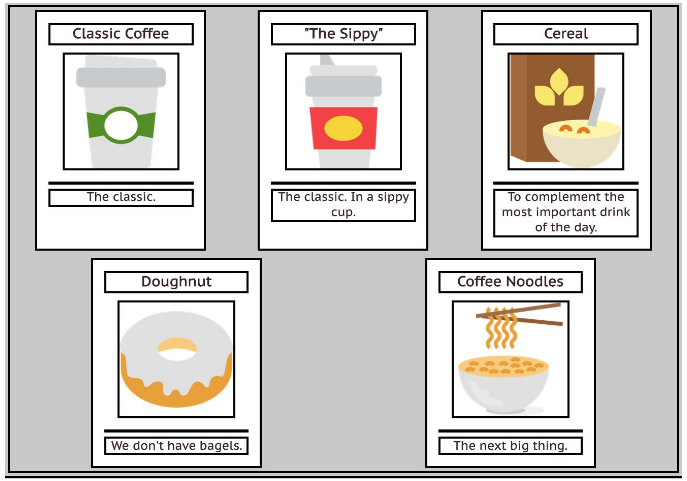
CSS

Wireframe

## Second Flex Container: `#item-container`

This is the `div` that holds all of the product article "cards". It would be nice to have some control over their distribution, and wrap them when the screen gets smaller.

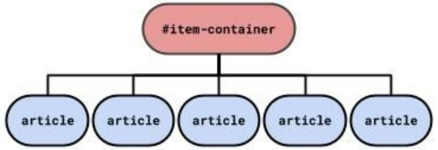
```
<div id="item-container">
  <article>...</article>
  <article>...</article>
  <article>...</article>
  <article>...</article>
  <article>...</article>
</div>
```



## #item-container Solution

```html
<div id="item-container">
  <article>...</article>
  <article>...</article>
  <article>...</article>
  <article>...</article>
  <article>...</article>
</div>
```

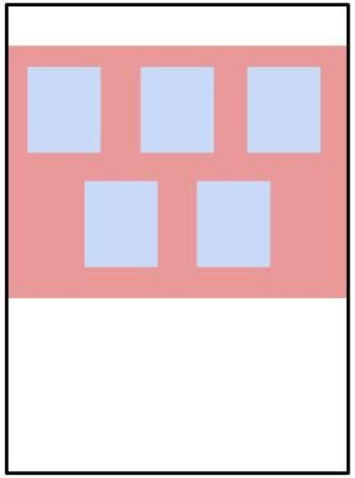
HTML (simplified)



DOM tree representation of flex parent container with 5 flex items

```css
#item-container {
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
}
```

CSS



Wireframe

## Third Flex Container: The `#top-bar`

```
<header id="top-bar">
  <h1>...</h1>
  <nav>...</nav>
</header>
```

- This is a bit of a trickier one, so it's good to do it last. We want to make it a flex row so we can get a nice distribution of space between the h1 and the nav.
- We'll also make the #top-bar a sticky nav bar, so it sticks to the top when you scroll down!
- With careful planning, we can combine different layout techniques like display: flex; and position: sticky.
- Let's take a look more closely at the CSS position property.

## positioning Elements

**`position: static`**

- Default value. Elements render in order, as they appear in the document flow

**`position: fixed`**

- Puts an element at an exact position within the browser window

**`position: absolute`**

- Puts an element at an absolute position based on the location of the element's parent container

**`position: relative`**

- Makes children positioned relative to the parent container
- Handy for sticking a footer to the bottom of a page, for example

**`position: sticky`**

- A "hybrid" - toggles between relative and fixed depending on scroll position

Another good explanation is here

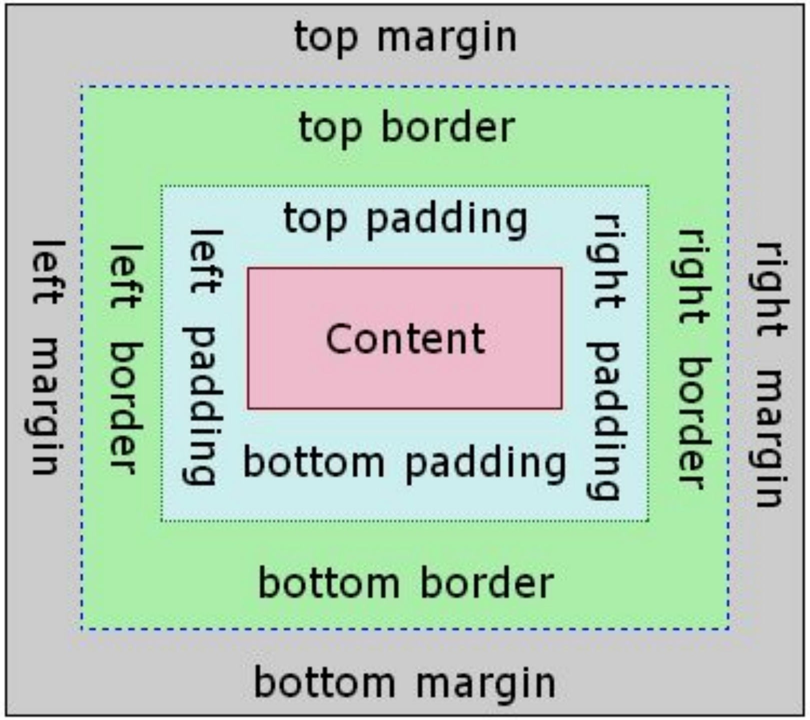
## Using `position: sticky` for the header/footer

- A sticky element toggles between relative and fixed depending on the scroll position - is fixed when a given offset position (e.g. top of 0) is met in the viewport
- See the Pen Sticky Examples by @mehovik on CodePen

## Review: Box Model Properties

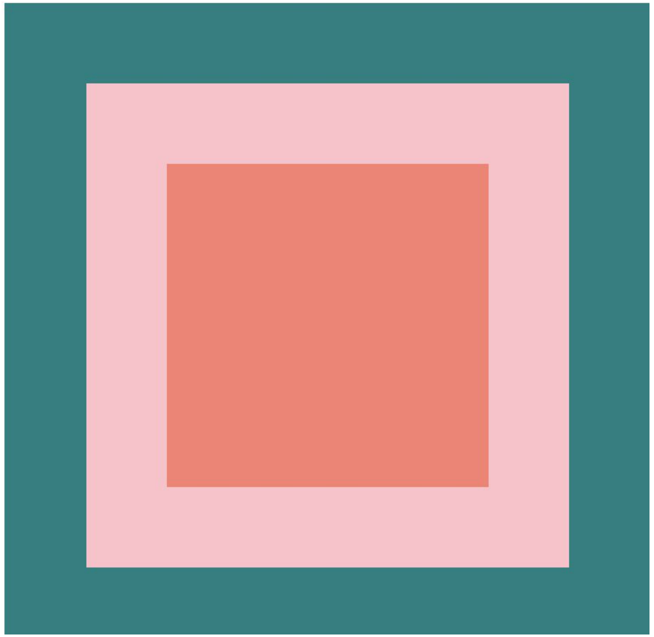
The box model is a way of representing the space that an element takes up on a page, and how it relates to other elements.

- Width/Height
- Padding
- Border
- Margin



## Boxes

Given boxes.html, write boxes.css to make the appearance below

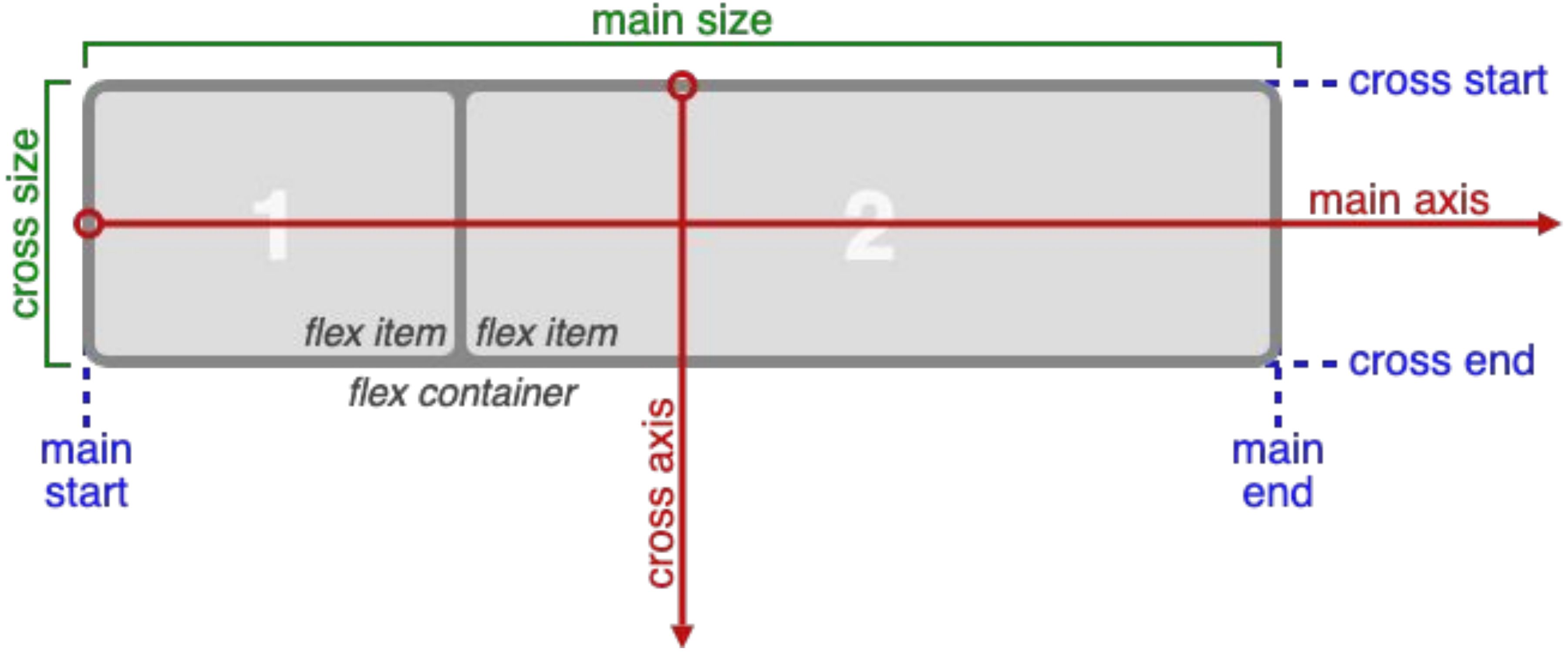


- The padding of the outer box is teal with a width of 50 pixels.
- The border of the inner box is pink with a width of 50 pixels.
- The background color of the inner box is salmon with a width and height of 200 pixels.
- The overall box has a total width and height of 400 pixels.

(solution)

## Introduction to Flex

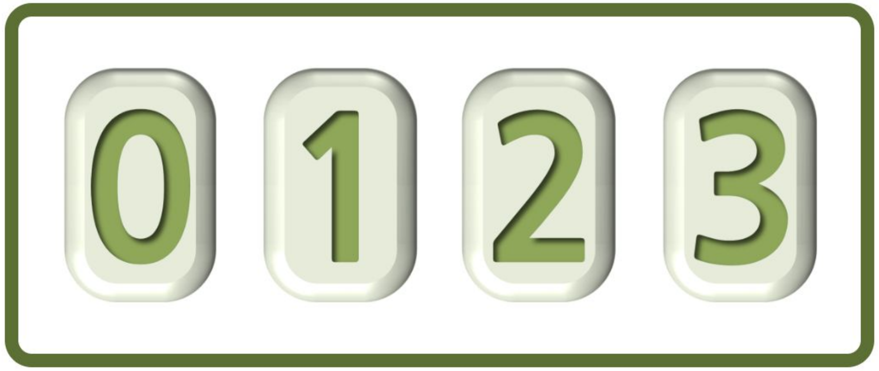
- Flexbox is a set of CSS properties for aligning block level content.
- Flexbox defines two types of content
  - "Container": The parent block element
  - "Items": anything directly nested inside of a flex container becomes a flex item.
- Various properties on the container determine how its items are layed out.


main size
cross start
cross size
1
2
main axis
flex item
flex item
cross end
flex container
cross axis
main start
main end

## Basic Properties of Flex Container

- `display: flex;`
    - makes an element a "container", items inside automatically become "items"
- `justify-content: flex-end; (flex-start, space-around,...)`
    - indicates how to space the items inside the container along the main axis
- `align-items: flex-end; (flex-start, center, baseline,...)`
    - indicates how to space the items inside the container along the cross axis
- `flex-direction: row; (column)`
    - indicates whether the container flows horizontally or vertically

## Flex with Number Card



- The card container is represented using a `div` and has 4 images, each representing a number. This card should be `500px` wide and `200px` tall with a `solid #698733 border` of `0.5em width` and a `border-radius` of `1em`.
- The four images in the card should take up `70%` of the `height` of the card, be centered `vertically` within, and `spaced-evenly` horizontally.

(starter files) & (solution)

## Flex with Number Card (challenge!)

- Once you've finished this exercise to get the expected output, how can you modify your CSS to get the numbers in the order "3 2 1 0"? How about "2 3 1 0"? For the second ordering, try to use the order property.

## Extra Practice: more flexbox resources

- Weird-flex
  - A visual aid for learning and seeing flex properties in action. Made by Chao, a CSE 154 TA!
- Flexbox ducky
  - a CSS game for learning the basics of Flexbox. It's fairly self-contained, but feel free to talk to your neighbors with any questions or let your TA know if you run into anything you're not sure about along the way!
- CSS-Tricks Guide to Flexbox
  - goes into a deeper explanation of the flex properties and has some great examples to compare each - it's a great bookmark resource to reference for this class!