FlexBox

Use `--such-notes` if they deserve better treatment.

CSS - under the hood

The Visual Formatting Model

The visual formatting model comprises of two stages:

1. Box Generation
2. Positioning

1. Box Generation
    - visual medium, e.g. web page, is made up of lots of 2D boxes, layered up along a 3rd dimension (z-index is the CSS property used to control the 3D layering order)
    - box generation takes each HTML element and generates one or more boxes. The type of element specifies how many boxes are generated; most elements generate one, e.g.
    - generates 2, one for the content and one for the bullet point icon
    - there are different types of boxes - the display property of the element defines its box's type
    - boxes have both an outer type (how it interacts with its parent and siblings) and an inner type (how it interacts with its children). These behaviours are independent - this will be useful later.
2. Positioning
    - All the boxes are then laid out on a page, taking into account:
        1. the chosen *Positioning Scheme*
        2. the box's dimensions and type,
        3. the relationships between elements in the document tree
        4. external info like viewport size

Positioning:

There are 3 positioning schemes:

1. *Normal flow*: Includes block formatting, inline formatting, table layouts and relative positioning

```
position: static | relative;
```

2. *Floats*: A box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible (to edge of its parent or until it hits another floated element)

```
float: left | right;
```

3. *Absolute positioning (includes fixed)*: Takes the box out of the normal flow, and precisely specifying its location: it is explicitly offset with respect to its containing block. It is removed from the normal flow entirely: it has no impact on siblings, and doesn't flow around other boxes.

```
position: absolute | fixed;
```

Flow:

What does it mean when elements are taken out of "flow"?

When boxes are layed out *independently* of their elements' positions/relationships in the document tree.

E.g. With an absolutely positioned element, the element may be placed anywhere within the visual medium, not respecting its location and relationships in the document tree.

Primer on concept of CSS flow

LAYOUT in CSS

Layout modes

CSS 2.1 defined 4 layout modes:

1. Block layout
2. Inline layout
3. Table layout
4. Positioned layout

1. Block layout
   - For laying out documents
   - Includes float (not originally intended for layouts!)
   - Vertical bias
2. Inline layout
   - For laying out text
   - Horizontal bias
3. Table layout
   - For laying out 2d data in tables
4. Positioned layout
   - For explicit positioning without much regard for other elements

CSS3 layout modes

CSS3 introduced two more: *grid layout* (for another time), and *flex layout*

My goal in doing Flexbox and later Grid was to replace all the crazy float/table/inline-block/etc hacks that I'd had to master as a webdev. All that crap was (a) stupid, (b) hard to remember, and (c) limited in a million annoying ways, so I wanted to make a few well-done layout modules that solved

- Motivation behind these new modes: created explicitly to replace float and table layout hacks.
- makes common design patterns very easy, as flexbox:
  - is Direction agnostic
  - distributes Space around components very easy
  - makes reordering components easily - great for responsive design

the same problems in simple, easy-to-use, and complete ways.

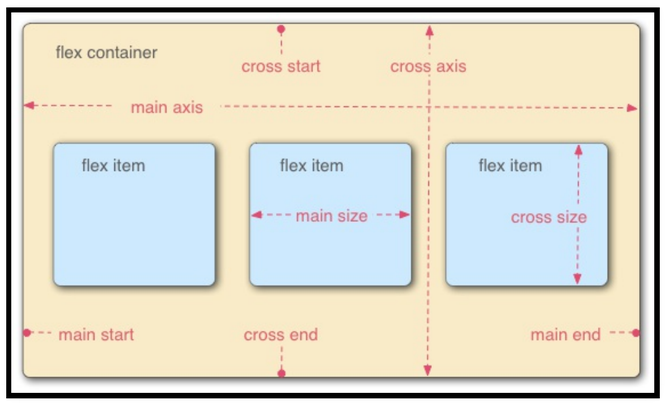
Tab Atkins Jr., author of the Flexbox and Grid specs

FlexBox foundations

flex item 2

flex item 3

Axes



Available Free Space

The inner width/height of the flex container, minus the sum of the preflex sizes of all the relevant lengths of the children.

(the lengths along the relevant axis, of the relevant children). Positive and negative available free space. The free space layout algorithm in Flexbox defines available free space as...

- If ∑preflexSizes < availableLengthOfFlexContainer, there is *positive available space*.
- If ∑preflexSizes) > availableLengthOfFlexContainer, there is *negative available space*.

Flexible lengths

- if positive_free_space -> distribute space amongst flexible lengths that can grow, proportional to their growth factors.
- if negatve_free_space -> shrink flexible lengths that can shrink, proportional to their shrink factors.

Defining aspect of flexbox layout: the ability to make various lengths of the flexbox items flexible - to distribute either positive or negative available free space amongst them, relative to their growth/shrink factors

- width
- height
- padding
- margin

Flex tuple

Each length of a flex item is represented by a *flex tuple*: a 5-tuple of information about a flexible length, containing:

1. Minimum size
2. Maximum size
3. Preferred size - defaults to 0, can be set to a different value with flex-basis: (or 3rd argument of flex:)
4. Positive flexibility
5. Negative flexibility

the minimum and maximum sizes are calculated depending on the type of length:

- margin and padding: Set the minimum size to 0 and the maximum size to infinity.
- Set the minimum size to the value of the ‘min-width’ or ‘min-height’, as appropriate
- borders: inflexible

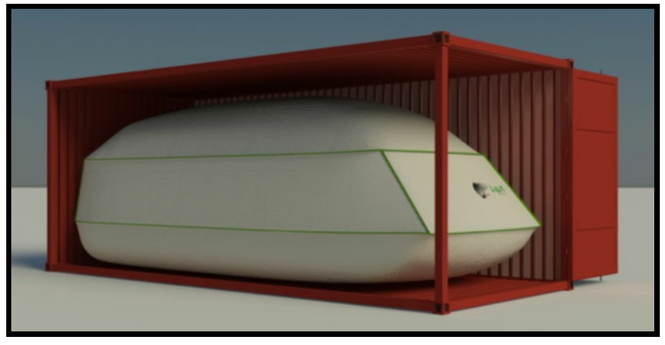
The flex algorithm

CSS resolves a flexible length into a definite length by:

1. Collect all lengths, flexible or inflexible, along a single axis, that will share the space.
2. Sum the *preflex* sizes of the lengths.
3. If there is positive available space, then the space is split up among all the flexible lengths with positive flexibility, according to their grow factors, to make the sum equal to the available length.
4. If there is negative available space, then all the flexible lengths with negative flexibility shrink in proportion to their shrink factors, to make the sum equal to the available length.

1. E.g. for a horizontal flex container, the flex items' left and right margins, left and right borders, left and right paddings, and widths, share the width. Each flex item's vertical margins, borders, padding and height individually share the height of the flex container.
2. *The preflex size of an inflexible length is just the length itself. The preflex size of a flexble length is its preferred size.*

Flex containers - the CSS



Flex container

```
display: flex | inline-flex;
```

- makes an element a flex container
- All the immediate children of the flex container automatically become flex items
- display: flex makes a box whose outer behaviour is block, and inner is a flex container
- display: inline–flex the outer behaviour is inline

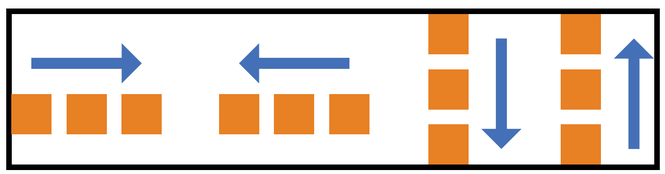
Demo

- start with block parent, block children
- make parent a flex container and children flex items by adding display: flex to container

Flex direction

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

- specify the orientation and direction of the main axis
- default: row, l->r (if direction: ltr;)



play with flex-direction on flex container

1. row (show it's default)
2. column
3. row-reverse
4. column-reverse

Justify content

```
justify-content: flex-start | flex-end | center | space-between | sp
```

- specifies how flex items are laid out along the main axis, i.e. how is free space distributed?

1. flex-start (default)
2. flex-end
3. center
4. space-between (items are evenly distributed along the line; first item touching the start, last item touching the end)
5. space-around (equal space between all flex items)

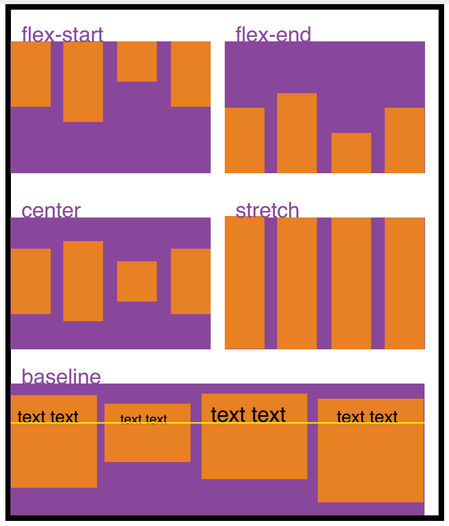
Align items

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

- specifies how flex items are laid out along the cross axis

First: change items 2 and 5 to be twice as tall, to illustrate these better.

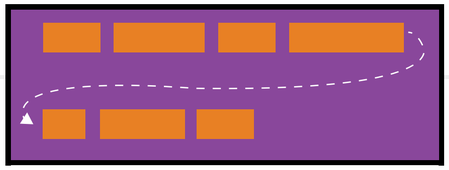
1. flex-start (default)
2. flex-end
3. center
4. baseline - (simplified version) aligns text inside the flex items
5. stretch - fills cross axis length [** doesnt do anything: because of flex items' default flex: values]



Flex wrap

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

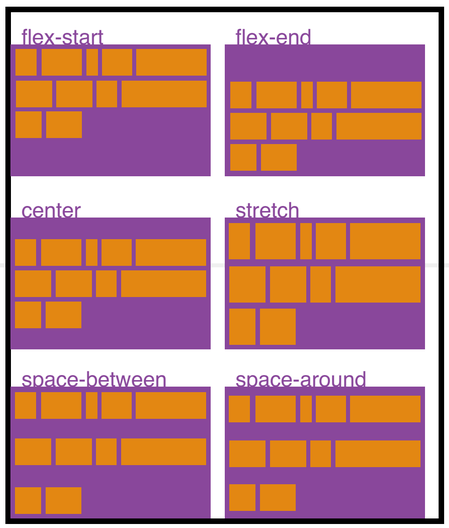
- flex items can wrap onto multiple lines
- default: nowrap



Align content

```
align-content: flex-start | flex-end | center | space-between | spac
```

- when the flex items have wrapped onto multiple lines - how they are positioned along the cross axis



Confusingly named - justifyContent and alignItems are equivalent

Flex items- the CSS

Flex grow and shrink

```
flex-grow: <positive number> /*default: 0*/
flex-shrink: <positive number> /*default: 1*/
```

- unitless value
- e.g. a flex item with flex shrink factor 3 will shrink three times faster than a sibling flex item with shrink factor 1. Both might have the same minimum length.

Demo:

- add flex-grow: 1 to one flex item (default: 0)
- add to a few others, also other values (including non-integer)
    - e.g. a single item having flex-grow: 0.5 means it takes half the available positive space

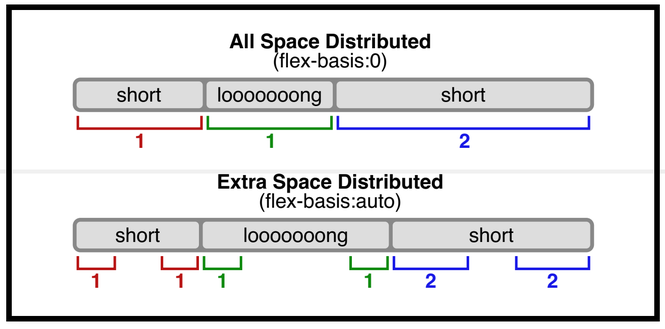
- Specifies how much the flex item will grow (the flex grow factor), or shrink (the flex shrink factor), relative to its sibling flex items, when the flex container's positive or negative free space is distributed among them.

Flex basis

```
flex-basis: <length> | auto; /*default auto*/
```

- auto and 0 mean "look at my width or height property", or if none look at content.
- also defines which 'part' of the length of the element is flexed:
  - 0: similar to auto and the whole length is flexed
  - auto: the space around the content inside the flex item is flexed.

- defines the initial length (along main axis) of the flex item, before free space is distributed (preflex length), as well as which 'part' of the length of the item is flexed.



Flex property shorthand

```
flex: <flex-grow> <flex-shrink> <flex-basis>; /*default 0 auto*/
```

- The flex property is a shorthand for the flex-grow, flex-shrink, and the flex-basis properties.

flex: 0 auto

Common values:

- flex: 0 auto - grow: 0, shrink: 1, basis: auto
- flex: auto - grow: 1, shrink: 1, basis: auto
- flex: none - grow: 0, shrink: 0, basis: auto
- flex: <positive number x> - grow: x, shrink: 1, basis: 0

- default value: grow: 0, shrink: 1, basis: auto
- Item is sized based on its width and height properties, or content if width/height not set
- Cant grow, but can shrink (to its minimum)
- Alignment abilities or auto margins can be used to align items along main axis
- revisit original flexbox demo

flex: auto

- grow: 1, shrink: 1, basis: auto
- Item is sized based on width/height properties, but is also fully flexible (grow and shrink)

flex: none

- grow: 0, shrink: 0, basis: auto
- Sized based on width/height, but fully inflexible

flex: <positive number>

- grow: , shrink: 1, basis: 0
- Flexible and flex basis 0 => item receives the specified proportion of the remaining space
- very common

Order

```
order: <int>; /*default: 0*/
```

- you can manually specify the order to display the flex items

demo

Demo - holy grail layout

RN Flex container properties

```
flexDirection: column | row (default: column)

justifyContent

alignItems
```

- row–reverse and column–reverse aren't supported

RN Flex item properties

flex: <integer>

- When flex is a positive number, it makes the component flexible and it will be sized proportional to its flex value. So a component with flex set to 2 will take twice the space as a component with flex set to 1.
- When flex is 0, the component is sized according to width and height and it is inflexible.
- When flex is -1, the component is normally sized according width and height. However, if there's not enough space, the component will shrink to its minWidth and minHeight.

flexGrow, flexShrink, and flexBasis work the same as in CSS. ** how do they interact with flex??

RN Flex item properties

```
aspectRatio: <number>
```

- Aspect ratio control the size of the undefined dimension of a node
- Unique to RN flexbox