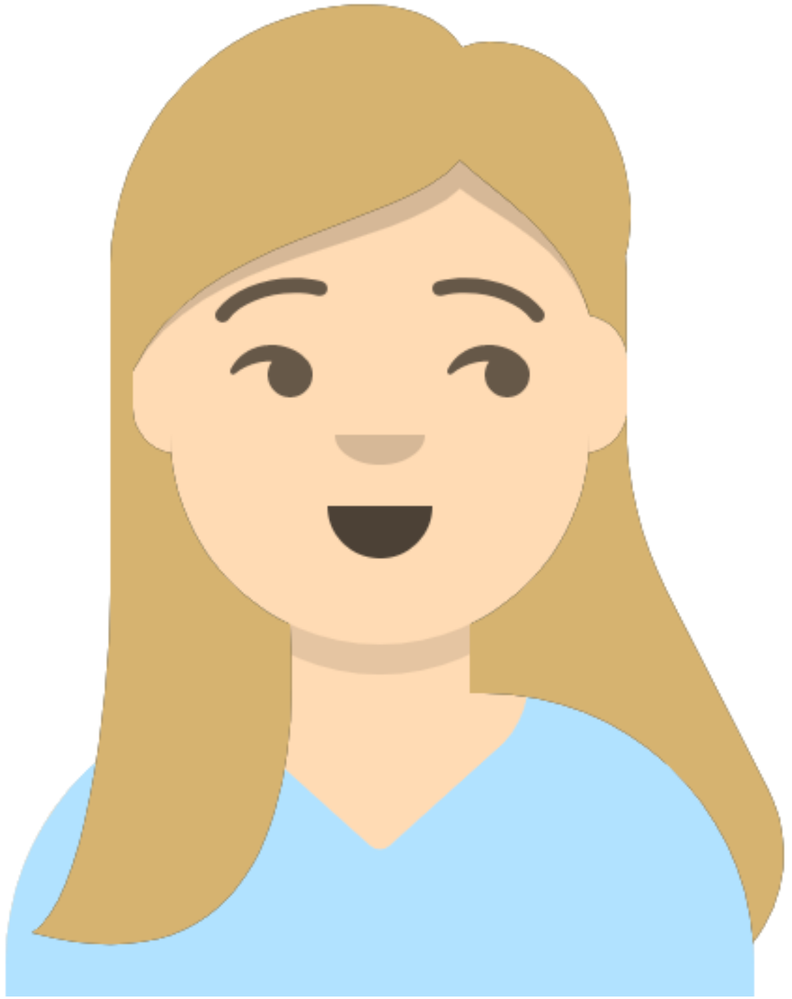
howest
hogeschool

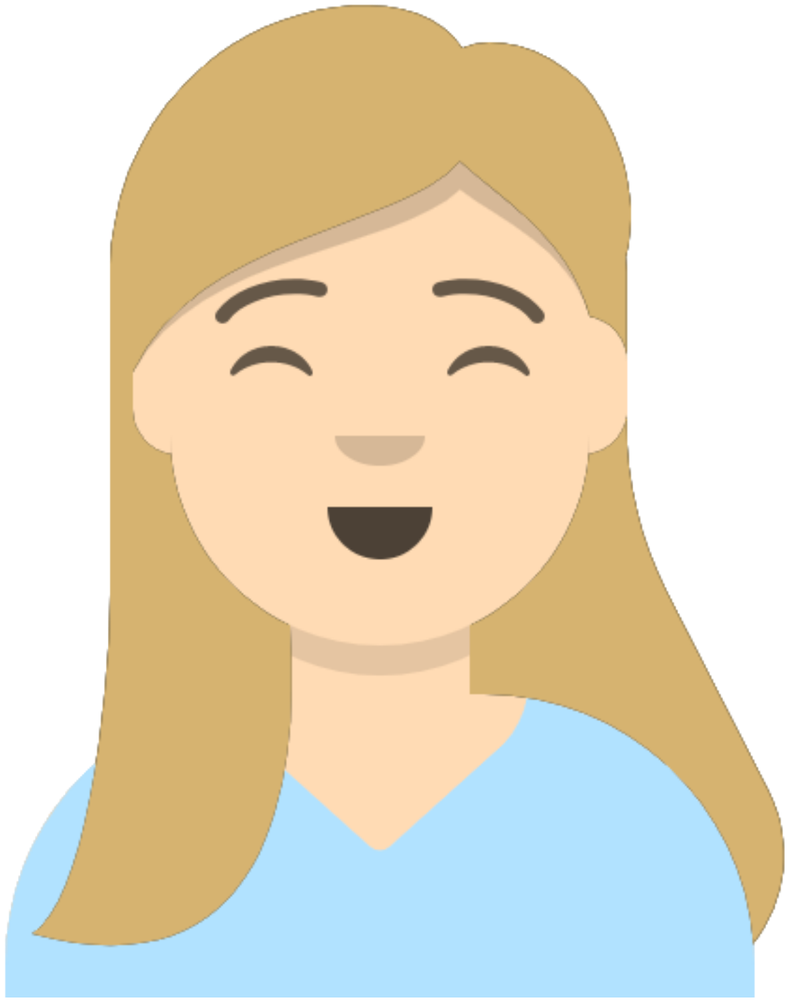
CSS Positioning

Web, Mobile and Security

Frédéric Vlummens



Guest lecture: Jill VandenDriessche



Yeah that means it’s CSS time, sorry!

Overview

- CSS Positioning
- Responsive design techniques (different video)
- CSS Animation techniques (different video)

Overview

- CSS Positioning
  - Static positioning
  - Absolute positioning
  - Relative positioning
  - z-index
  - Fixed positioning
  - Sticky positioning
- Responsive design techniques (different video)
- CSS Animation techniques (different video)

Positioning, what?

- One of the different layout techniques
  - Box model
  - Flexbox
  - Grid
  - Float
- Use with care!

howest
hogeschool

Static positioning

Static positioning is the **default behaviour** of the element. It is contained within the page flow, interacting with other elements as « normal »

howest
hogeschool

Absolute positioning

An **absolutely positioned element** is an element whose computed `position` value is `absolute` or `fixed`. The `top`, `right`, `bottom`, and `left` properties specify offsets from the edges of the element's containing block. (The containing block is the ancestor relative to which the element is positioned.) If the element has margins, they are added to the offset. The element establishes a new block formatting context (BFC) for its contents.

An **absolutely positioned element** is an element whose computed `position` value is `absolute` or `fixed`. The `top`, `right`, `bottom`, and `left` properties specify offsets from the edges of the element's containing block. (The containing block is the ancestor relative to which the element is positioned.) If the element has margins, they are added to the offset. The element establishes a new block formatting context (BFC) for its contents.

An **absolutely positioned element** is an element whose computed `position` value is `absolute` or `fixed`. The `top`, `right`, `bottom`, and `left` properties specify offsets from the edges of the element's containing block. (The containing block is the ancestor relative to which the element is positioned.) If the element has margins, they are added to the offset. The element establishes a new block formatting context (BFC) for its contents.

This indicates the position of the element cannot be changed by other elements, they do not affect the box

An **absolutely positioned element** is an element whose computed `position` value is `absolute` or `fixed`. The `top`, `right`, `bottom`, and `left` properties specify offsets from the edges of the element's containing block. (The containing block is the ancestor relative to which the element is positioned.) If the element has margins, they are added to the offset. The element establishes a new block formatting context (BFC) for its contents.

Understanding BFCs is crucial to successful CSS strategies:
https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Block_formatting_context

When to **use** absolute positioning?

When to **use** absolute positioning?

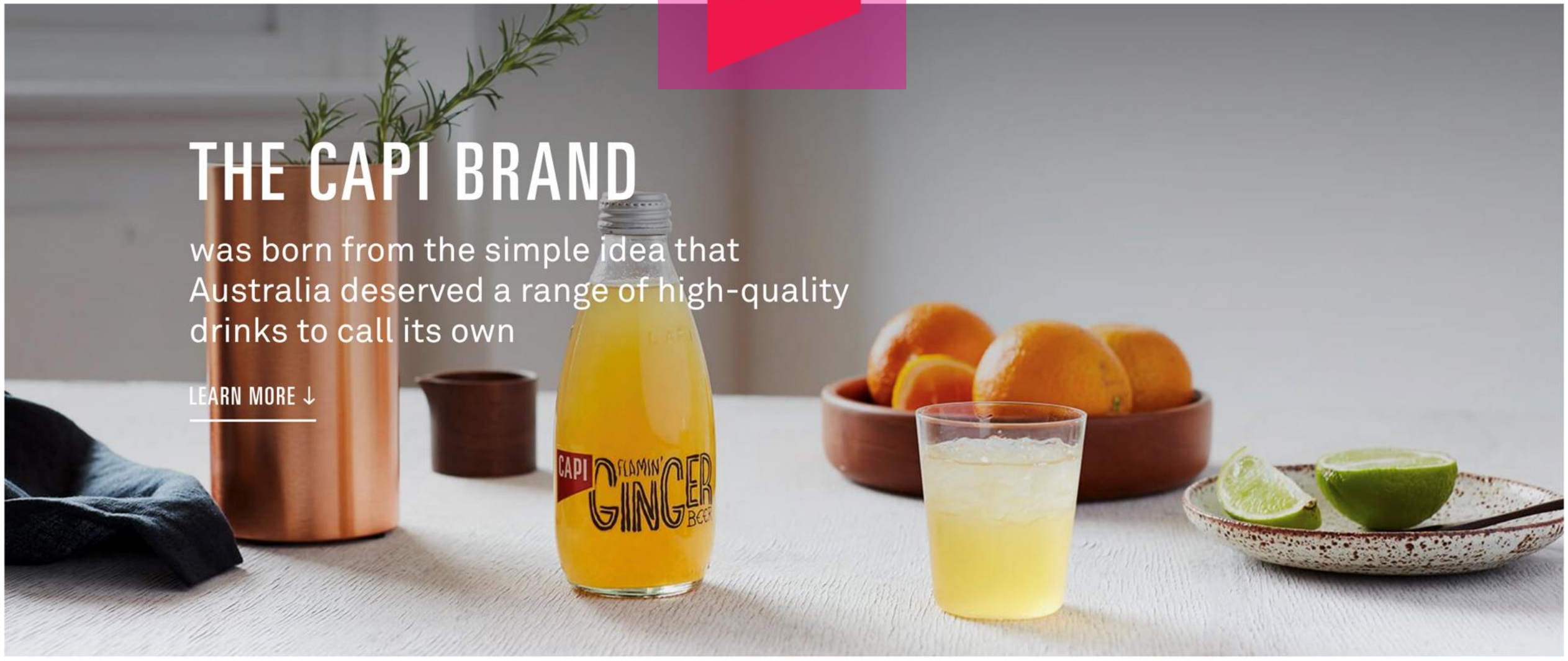
When an element **does not** fit the grid

THE CAPI BRAND

was born from the simple idea that Australia deserved a range of high-quality drinks to call its own

LEARN MORE ↓

THE RANGE



THE CAPI BRAND

was born from the simple idea that Australia deserved a range of high-quality drinks to call its own

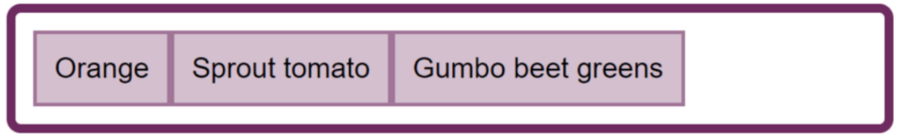
LEARN MORE ↓

THE RANGE

[a flex container] and explored [alignment as it works in Flexbox]. This time we are going to take a look at sizing. How do we control the size of our flex items, and what choices is the browser making when it controls the size?

Initial Display Of Flex Items

If I have a set of items, which have variable lengths of content inside, and set their parent to `display: flex`, the items will display as a row and line up at the start of that axis. In the example below my three items have a small amount of content and are able to display the content of each item as an unbroken line. There is space at the end of the flex container which the items do not grow into because the initial value of `flex-grow` is `0`, *do not grow*.

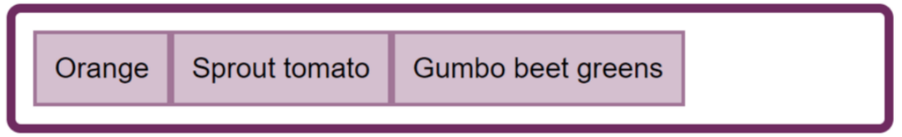


The flex items have room to each be displayed on one line (Large preview)

[a flex container] and explored [alignment as it works in Flexbox]. This time we are going to take a look at sizing. How do we control the size of our flex items, and what choices is the browser making when it controls the size?

Initial Display Of Flex Items

If I have a set of items, which have variable lengths of content inside, and set their parent to `display: flex`, the items will display as a row and line up at the start of that axis. In the example below my three items have a small amount of content and are able to display the content of each item as an unbroken line. There is space at the end of the flex container which the items do not grow into because the initial value of `flex-grow` is `0`, *do not grow*.

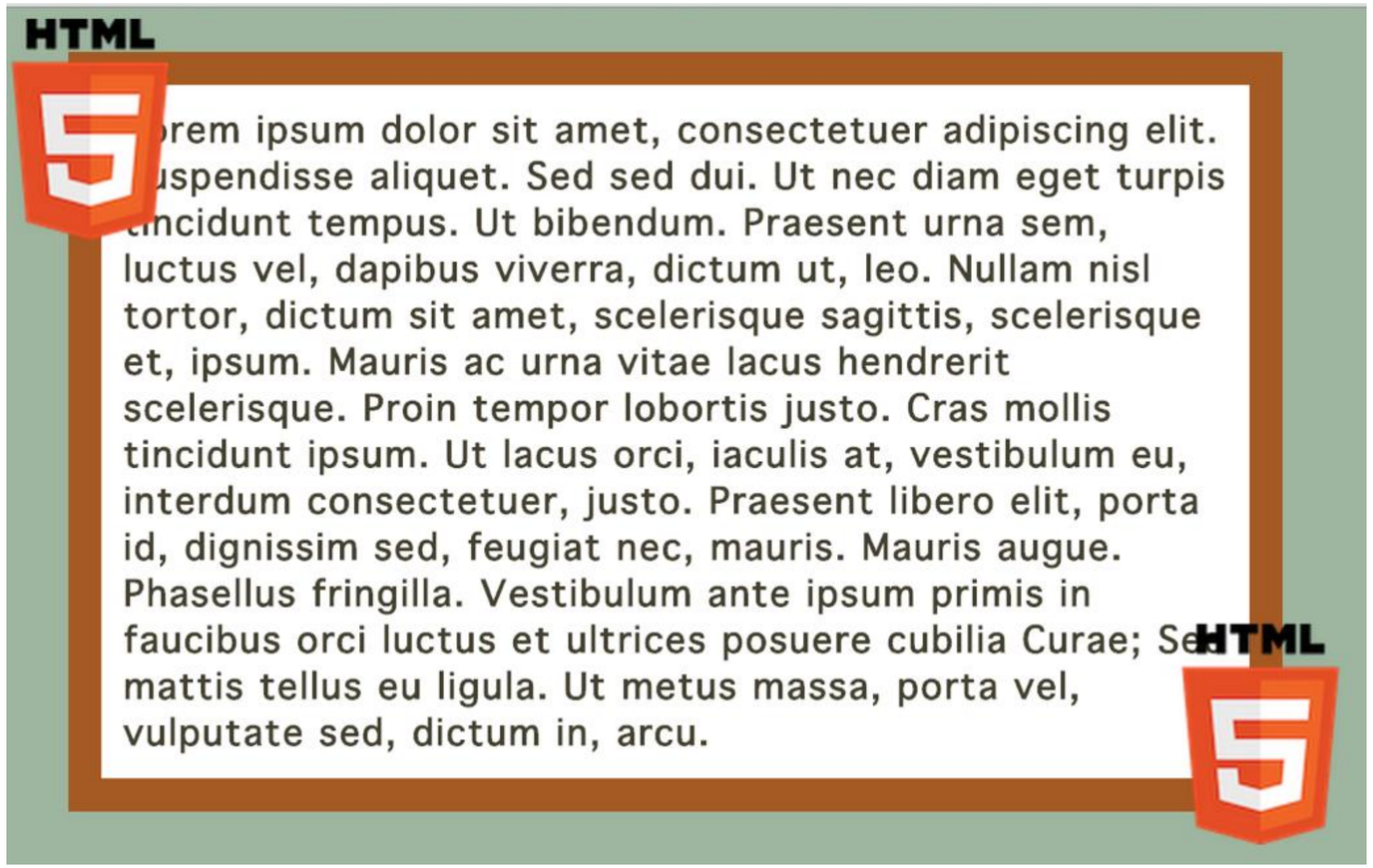


The flex items have room to each be displayed on one line (Large preview)

position: absolute;

Combined with top / left / right / bottom properties

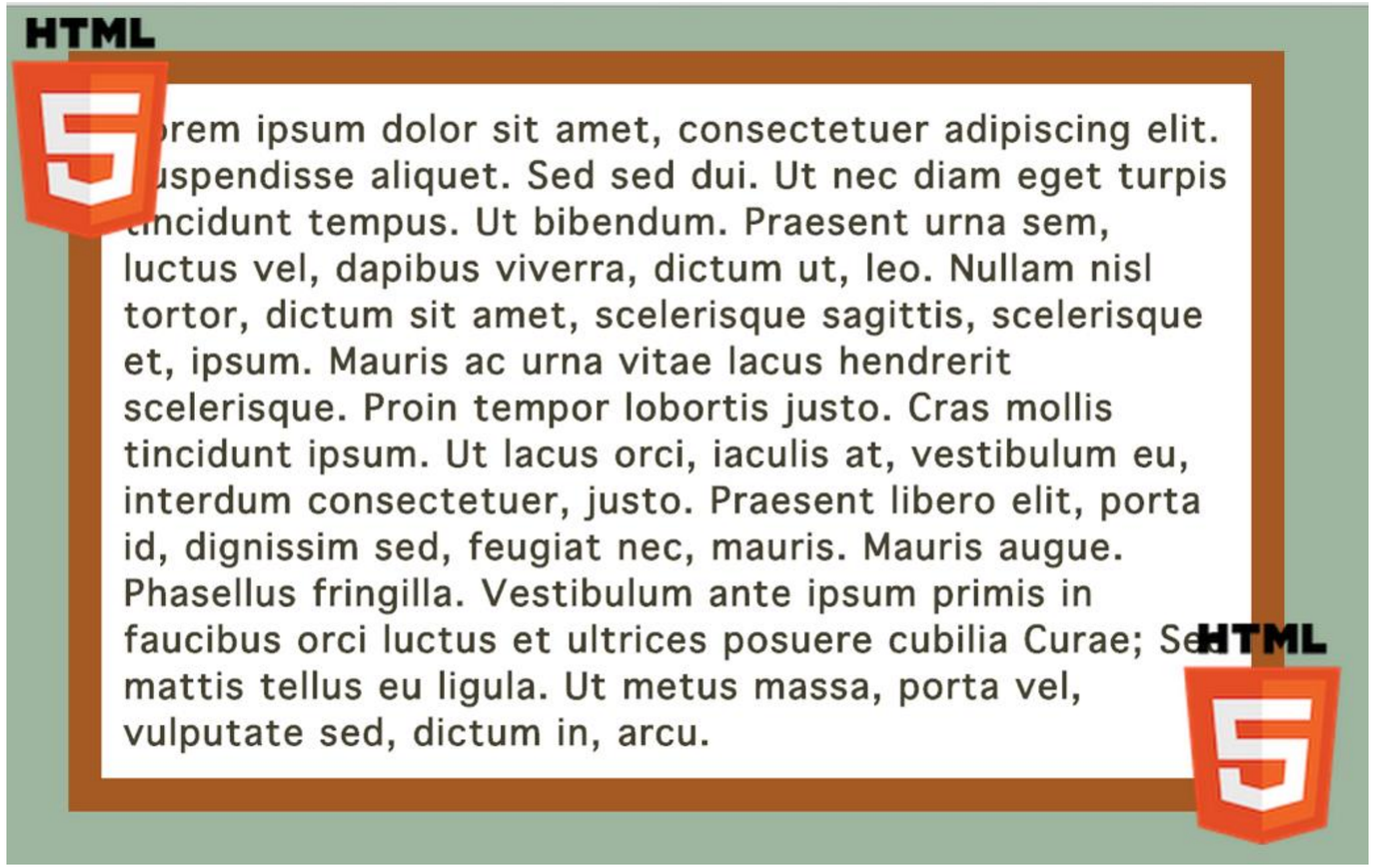
position: absolute;
top: 10px;
left: 20px;



position: absolute;
bottom: 10px;
right: 20px;

Notice the flow of the other elements is unaffected

position: absolute;
top: 10px;
left: 20px;



position: absolute;
bottom: 10px;
right: 20px;

Beware!

Absolute positioned elements need a « parent » declaration, or they will default to the browser window / viewport

Beware!

Absolute positioned elements need a « parent » declaration, or they will default to the browser window / viewport

A common technique is to **declare the parent container** as position:relative; without further properties, preserving its positioning in the document flow

howest
hogeschool

Relative positioning

A **relatively positioned element** is an element whose computed `position` value is `relative`. The `top` and `bottom` properties specify the vertical offset from its normal position; the `left` and `right` properties specify the horizontal offset.

A **relatively positioned element** is an element whose computed `position` value is `relative`. The `top` and `bottom` properties specify the vertical offset from its normal position; the `left` and `right` properties specify the horizontal offset.

Offset means that the original space the element took up is preserved

position: relative;

Combined with top / left / right / bottom properties

position: relative; left: 200px;

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Suspendisse aliquet. Sed sed dui. Ut nec diam eget turpis tincidunt tempus. Ut bibendum. Praesent urna sem, luctus vel, dapibus viverra, dictum ut, leo. Nullam nisl tortor, dictum sit amet, scelerisque sagittis, scelerisque et, ipsum. Mauris ac urna vitae lacus hendrerit scelerisque. Proin tempor lobortis justo. Cras mollis tincidunt ipsum. Ut lacus orci, iaculis at, vestibulum eu,

interdum consectetuer, justo. Praesent libero elit, porta id, dignissim sed, feugiat nec, mauris. Mauris augue. Phasellus fringilla. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Sed mattis tellus eu ligula. Ut metus massa, porta vel, vulputate sed, dictum in, arcu.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Suspendisse aliquet. Sed sed dui. Ut nec diam eget turpis tincidunt tempus. Ut bibendum. Praesent urna sem, luctus vel, dapibus viverra, dictum ut, leo. Nullam nisl tortor, dictum sit amet, scelerisque sagittis, scelerisque et, ipsum. Mauris ac urna vitae lacus hendrerit scelerisque. Proin tempor lobortis justo. Cras mollis tincidunt ipsum. Ut lacus orci, iaculis at, vestibulum eu,

interdum consectetuer, justo. Praesent libero elit, porta id, dignissim sed, feugiat nec, mauris. Mauris augue. Phasellus fringilla. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Sed mattis tellus eu ligula. Ut metus massa, porta vel, vulputate sed, dictum in, arcu.

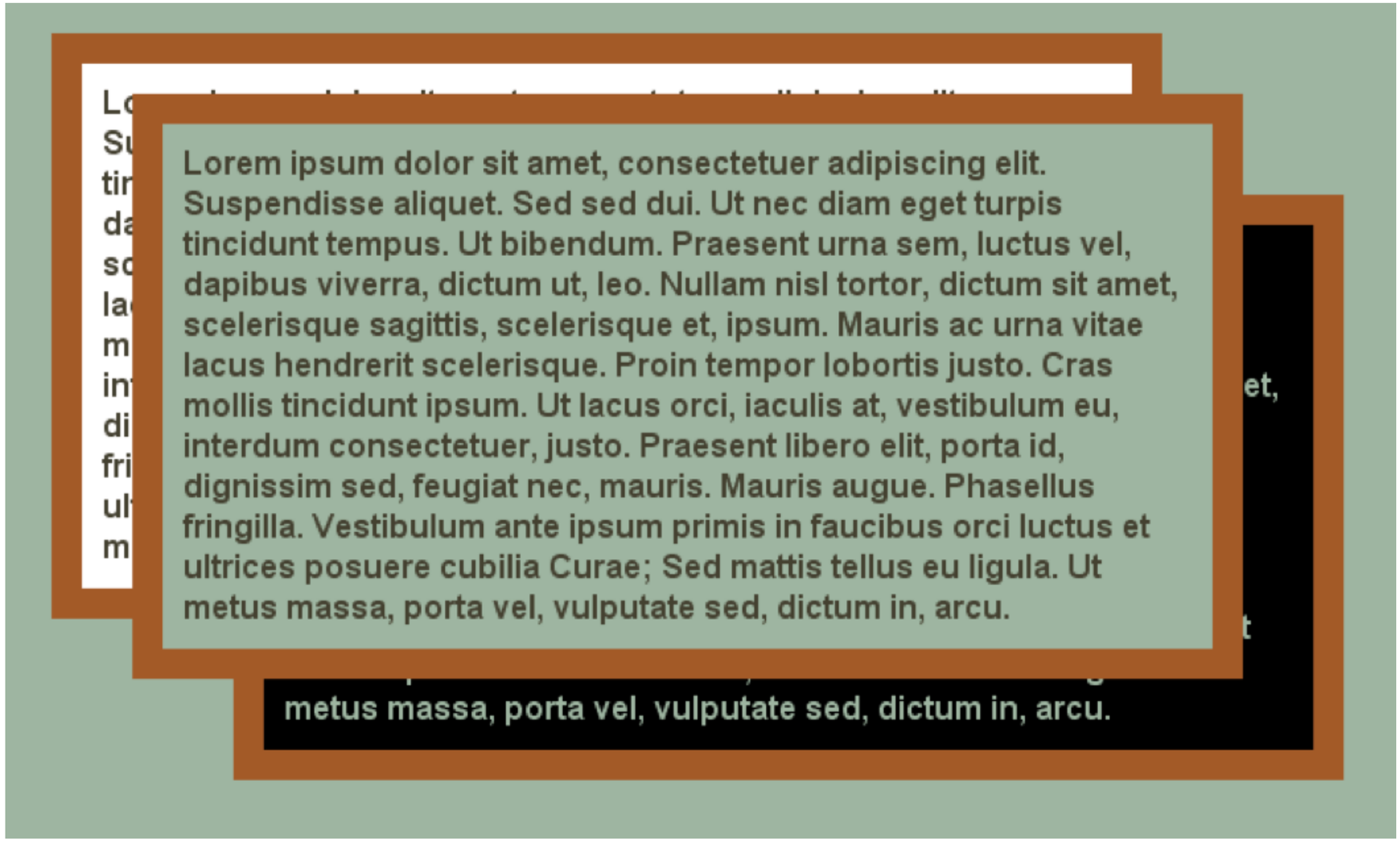
howest
hogeschool

z-index

The **z-index** determines the stacking order of an element.

By default last elements end up on top. Values can be positive or negative

position: absolute;
top: 10px;
left: 20px;
z-index: 1;



howest
hogeschool

demonstration in code

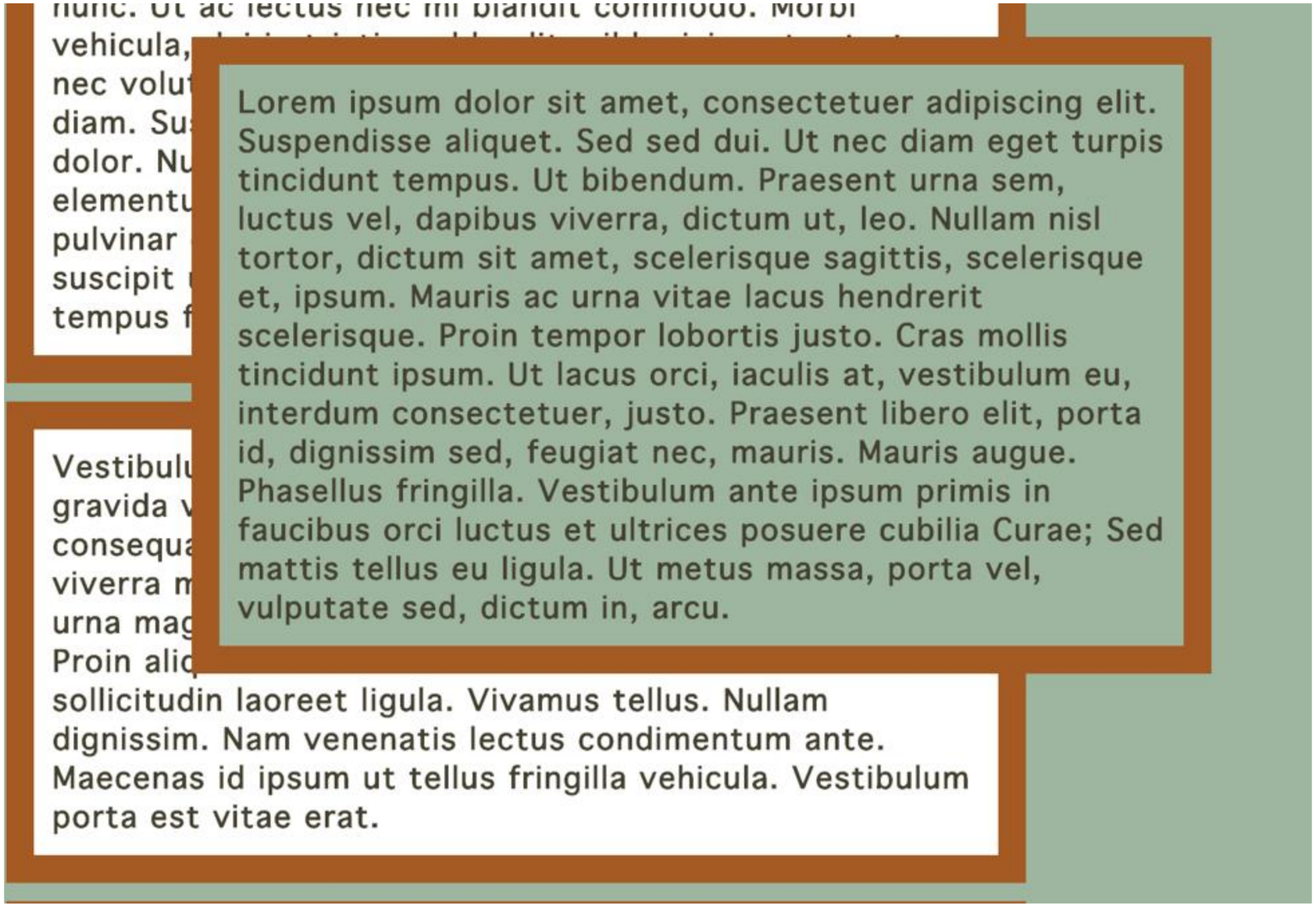
howest
hogeschool

Fixed positioning

An **absolutely positioned element** is an element whose computed `position` value is `absolute` or `fixed`. The `top`, `right`, `bottom`, and `left` properties specify offsets from the edges of the element's containing block. (The containing block is the ancestor relative to which the element is positioned.) If the element has margins, they are added to the offset. The element establishes a new block formatting context (BFC) for its contents.

Behaves in the same way as an absolute positioned element, with the only difference that its position “sticks”
The other elements on the page can scroll while this element stays in place, at all times

position: fixed;
top: 10px;
left: 50px;



howest
hogeschool

demonstration in code

Fixed positioning is similar to absolute positioning, with the exception that the element's containing block is the initial containing block established by the *viewport*, unless any ancestor has `transform`, `perspective`, or `filter` property set to something other than `none` (see CSS Transforms Spec), which then causes that ancestor to take the place of the elements containing block. This can be used to create a "floating" element that stays in the same position regardless of scrolling. In the example below, box "One" is fixed at 80 pixels from the top of the page and 10 pixels from the left. Even after scrolling, it remains in the same place relative to the viewport.

howest
hogeschool

Sticky positioning

A **stickily positioned element** is an element whose computed `position` value is `sticky`. It's treated as relatively positioned until its containing block crosses a specified threshold (such as setting `top` to value other than auto) within its flow root (or the container it scrolls within), at which point it is treated as "stuck" until meeting the opposite edge of its containing block.

A **stickily positioned element** is an element whose computed `position` value is `sticky`. It's treated as relatively positioned until its containing block crosses a specified threshold (such as setting `top` to value other than auto) within its flow root (or the container it scrolls within), at which point it is treated as "stuck" until meeting the opposite edge of its containing block.

i.e. intelligent fixed positioning ☺

howest
hogeschool

demonstration in code

CSS position:sticky - WD

Keeps elements positioned as "fixed" or "relative" depending on how it appears in the viewport. As a result the element is "stuck" when necessary while scrolling.

Usage	% of all users			?
Global	23.82%	+ 70.32%	=	94.14%
unprefixed:	7.22%	+ 70.3%	=	77.51%

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browse
			4-22											
		2-25	2 23-36											
	12-15	1 26-31	37-51	3.1-6	10-38	3.2-5.1								
	6 16-18	3 32-58	2 52-55	5 6.1-7	2 39-41	5 6-7.1							4-5.4	
6-10	4 79	59-73	4 56-79	7.1-12.1	4 42-65	8-13.2		2.1-4.4.4	12-12.1				6.2-10.1	
11	4 80	74	4 80	13	4 66	13.3	all	4 80	4 46	4 80	68	4 12.12	11.1	2 1.2
		75-76	4 81-83	13.1-TP		13.4								