

A photograph of the TU Delft campus under a blue sky with white clouds. In the foreground, there's a modern building with a grey facade and a red brick walkway. In the middle ground, a large green lawn with several trees and a paved path. In the background, a tall, modern glass building with a red vertical stripe and a clock tower is visible.

CSS: the language of Web design

Claudia Hauff
cse1500-ewi@tudelft.nl



Learning goals

- **Position and style** HTML elements according to a given design
- **Employ** pseudo-classes and pseudo-elements
- **Employ** CSS' data access/creation facilities
- **Write** CSS media queries
- **Create** CSS-based animations



<https://codepen.io/charlottehase/pen/abzYoQp>



A word of warning ...



I Am Devloper

@iamdevloper

Following

when I think I'm a fairly decent programmer,
and then try and align two buttons next to
each other in CSS



12:45 PM - 30 Nov 2017

2,794 Retweets 7,692 Likes



92



2.8K



7.7K





A bit of context (again).

Cascading Style Sheets (CSS)

- **CSS1**: W3C recommendation in **1996**
 - Fonts, colours, alignment, margins, ids and classes
- **CSS2**: W3C recommendation in **1998**
 - Media queries, element positioning, etc.
- **CSS2.1**: W3C recommendation in **2011**
 - Fixed errors and added already implemented browser features
- **CSS3**: **specification split up** into modules; progress varies
 - <https://www.w3.org/Style/CSS/current-work>
 - **Polyfills, fallbacks**

Geometry Interfaces Level 1

CSS Cascading and Inheritance Level 4

CSS Scroll Snap Level 1

CSS Painting API Level 1

CSS Writing Modes Level 4

CR	CR
CR	PR

Refining

CSS Animations Level 1

Web Animations

CSS Text Level 3

CSS Transforms Level 1

CSS Transitions

CSS Box Alignment Level 3

Selectors Level 4

CSS Lists Level 3

Motion Path Level 1

Preview of CSS Level 2

CSS Fonts Level 4

CSS Easing Functions Level 1

Current Upcoming Notes

WD	WD
WD	WD
WD	CR
CR	PR
WD	CR

Abbreviation	Full name
FPWD	First Public Working Draft
WD	Working Draft
CR	Candidate Recommendation
PR	Proposed Recommendation
REC	Recommendation

CSS3

- Impossible to write complex CSS that relies on modern features and works across all browsers
- Implementation of CSS3 features should be decided based on:
 - **intended users**
 - **mode of usage**
 - **type** of web app



Revision chapter 3

Chapter 3 of the course book

CSS describes how elements in the DOM should be rendered.

px: "the magic unit of CSS"

```
body {  
    background-color: #ffff00;  
    width: 800px;  
    margin: auto;  
}  
h1 {  
    color: maroon;  
}  
p span {  
    color: gray;  
    border: 1px solid gray;  
}  
p#last {  
    color: green;  
}
```

selector

property

value

140+ color names in CSS3

Three types of style sheets:

- (1) **browser's** style sheet
- (2) **author's** style sheet
- (3) **user's** style sheet overriding

Style sheets are processed in order; **later declarations**

override earlier ones (if they are on the same or a higher *specificity* level)

! important overrides all other declarations



Pseudo-elements and pseudo-classes

Pseudo-class

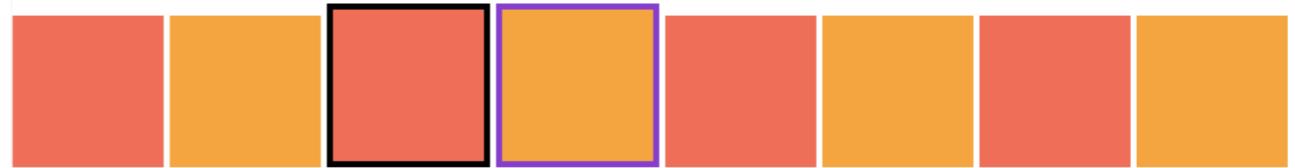
Pseudo-class: a keyword added to a **selector** which indicates a **particular state or type** of the corresponding element.

Pseudo-classes allow styling according to (among others) **document external** factors (e.g. mouse movements, user browsing history).

```
selector:pseudo-class {  
    property: value;  
    property: value;  
}
```

Children & types

Weather



:nth-child(x) any element that is the Xth child element of its parent

n represents a number starting at 0 and (X can be an int or formula, e.g. "2n+1")

:nth-of-type(x) any element that is the Xth sibling of its type

```
<main>
  <h1>Weather</h1>
  <span id="c1"></span>
  <span id="c2"></span>
  <span id="c3"></span>    4th child
  <span id="c4"></span>
  <span id="c5"></span>
  <span id="c6"></span>
  <span id="c7"></span>
  <span id="c8"></span>
</main>
```

parent

```
span {
  width: 100px;
  height: 100px;
  display: inline-block;
}

span:nth-child(2n){
  background-color: tomato;
}

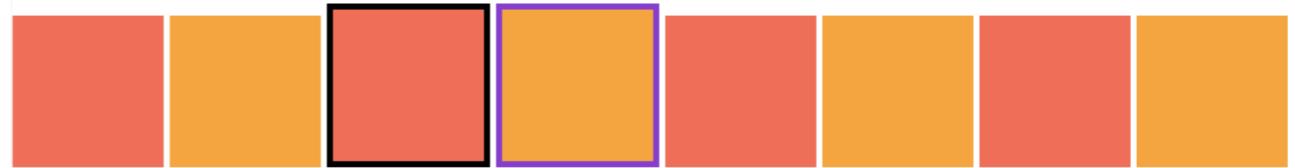
span:nth-child(2n+1){
  background-color: orange;
}

span:nth-child(4){
  border: 4px solid black;
}

span:nth-of-type(4){
  border: 4px solid darkviolet;
}
```

Children & types

Weather



:nth-child(x) any element that is the Xth child element of its parent

n represents a number starting at 0 and (X can be an int or formula, e.g. "2n+1")

:nth-of-type(x) any element that is the Xth sibling of its type

```
<main>
  <h1>Weather</h1>
  <span id="c1"></span>
  <span id="c2"></span>
  <span id="c3"></span>
  <span id="c4"></span>
  <span id="c5"></span>
  <span id="c6"></span>
  <span id="c7"></span>
  <span id="c8"></span>
</main>
```

siblings

```
span {
  width: 100px;
  height: 100px;
  display: inline-block;
}

span:nth-child(2n){
  background-color: tomato;
}

span:nth-child(2n+1){
  background-color: orange;
}

span:nth-child(4){
  border: 4px solid black;
}

span:nth-of-type(4){
  border: 4px solid darkviolet;
}
```

Popular pseudo-classes

Pseudo-class	Equivalent to
:first-child	:nth-child(1)
:last-child	:nth-last-child(1)
:first-of-type	:nth-of-type(1)
:last-of-type	:nth-last-of-type(1)

CSS variables

included in 2015-16 in major browsers

:root represents the <html> element; **global scope CSS variables** can be added:

1. custom prefix --
2. variable access via var()

Local scope CSS variables can be added everywhere (but are only accessible within their {...})

```
:root {  
/* 100+ color names exist, check e.g.  
 * https://en.wikipedia.org/wiki/Web_colors  
 */  
--boardColor1: steelblue;  
--boardColor2: deeppink;  
--hoverTextColor: mediumvioletred;  
--borderBoardColor: pink;  
--nailColor: darkblue;  
--boardRotationStart: 5deg;  
--boardRotationEnd: -5deg;  
--switchOnColor: dodgerblue;  
--switchOffColor: grey;  
--switchColor: white;  
}
```

```
.letter {  
width: 130px;  
height: 150px;  
/* border-radius rounds the corners of an element */  
border-radius: 0 0 20px 20px;  
/*linear-gradient creates an img based on 2+ colors */  
background: linear-gradient(to top, var(--boardColor2), var(--boardColor1));  
border: 5px solid var(--borderBoardColor);  
opacity: 0.7; /*determines transparency: 0 (invisible), 1 (opaque) */  
/* creates shadow around element: x-offset, y-offset, blur amount, color */  
box-shadow: 5px 5px 10px var(--boardColor2);  
...  
}
```



Hover & active

Often employed in:

- image galleries
- dropdown menus

:hover a pointing device (mouse) hovers over the element

:active the element is currently being active (e.g. clicked)

```
button {  
background: white;  
color: darkgray;  
width: 100px;  
padding: 5px;  
font-weight: bold;  
text-align: center;  
border: 1px solid darkgray;  
}
```

```
button:hover {  
color: white;  
background: darkgray;  
}
```

```
button:active {  
border: 1px dashed;  
border-color: black;  
}
```



Enabled & disabled

Add Todo

:enabled

an element that can be clicked/
selected

:disabled

an element that cannot be clicked/
selected

game project:
not all tiles/buttons are
available at all times
(e.g. connect4: disable
columns that are already
full)

```
button {  
    background: white;  
    color: darkgray;  
    width: 100px;  
    padding: 5px;  
    ...  
}  
  
button:enabled:hover {  
    color: white;  
    background: darkgray;  
}  
  
button:enabled:active {  
    border: 1px dashed;  
    border-color: black;  
}  
  
button:disabled {  
    background: #ddd;  
    color: #aaa;  
}
```

pseudo-classes
can be combined
(logical and)

Not

`:not(X)`

matches all elements that are not represented by selector X

```
<main>
  <h2>Todos</h2>
  <p id="firssttodo">Today's todos</p>
  <p class="todo">Tomorrow's todos</p>
  <p class="todo">Saturday's todos</p>
  <p>Sunday's todos</p>
</main>
```

```
main :not(.todo) {
  color: orange;
}
```

el1 el2: selects all `<el2>` elements inside `<el1>`



Todos

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

Not

`:not(X)`

matches all elements that are not represented by selector X

```
<main>
  <h2>Todos</h2>
  <p id="firssttodo">Today's todos</p>
  <p class="todo">Tomorrow's todos</p>
  <p class="todo">Saturday's todos</p>
  <p>Sunday's todos</p>
</main>
```

```
main *:not(.todo) {
  color: orange;
}
```

Todos

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

el1 el2: selects all `<el2>` elements inside `<el1>`

whitespace in selectors implies the universal selector: *

Not

`:not(X)`

matches all elements that are not represented by selector X

```
<main>
  <h2>Todos</h2>
  <p id="firstattodo">Today's todos</p>
  <p class="todo">Tomorrow's todos</p>
  <p class="todo">Saturday's todos</p>
  <p>Sunday's todos</p>
</main>
```

```
main *:not(.todo) {
  color: orange;
}
```

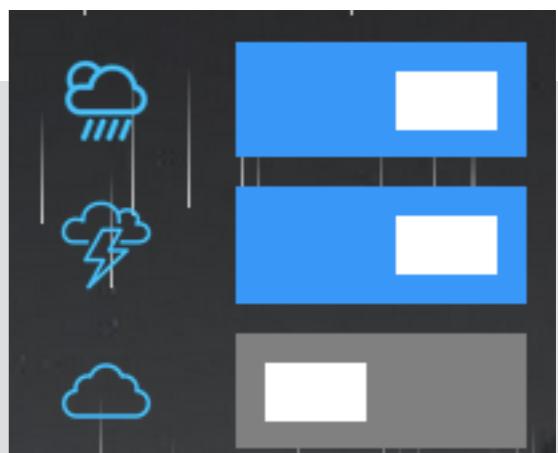
```
Todos
Today's todos
Tomorrow's todos
Saturday's todos
Sunday's todos
```

el1 el2: selects all `<el2>` elements inside `<el1>`

whitespace in selectors implies the universal selector: *

Selector combinations

Selector	Description
e1	Selects all <e1> elements
e1 e2	Selects all <e2> elements within <e1>
e1,e2	Selects all <e1> elements and all <e2> elements
e1>e2	Selects all <e2> elements that have <e1> as parent (i.e. direct children)
e1+e2	Selects all <e2> elements that follow <e1> immediately



Pseudo-elements

- Create **abstractions** about the **document tree** beyond those specified by the document language
- Provide access to an element's **sub-part**
- :: notation, however, one-colon notation is also acceptable for older pseudo-elements

Two firsts

::first-letter

::first-line

Canonical example: enlarge the first letter/line of a paragraph

```
p::first-line {  
    color: gray;  
    font-size: 125%; ←  
}  
  
p::first-letter {  
    font-size: 200%; ←  
}
```

relative font size changes

```
<p>  
To be, or not to be, that is the question-  
</p>  
<p>  
Whether 'tis Nobler in the mind to suffer  
The Slings and Arrows of outrageous  
Fortune,...  
</p>
```

Two firsts

```
::first-letter
```

```
::first-line
```

Canonical example: enlarge the first letter/line of a paragraph

```
p::first-line {  
  color: gray;  
  font-size: 125%;  
}  
  
p::first-letter {  
  font-size: 200%;  
}
```

relative font size changes

To be, or not to be, that is the question—
Whether 'tis Nobler in the mind to suffer The Slings and Arrows of outrageous Fortune,...

Before and after

::before add (cosmetic) content before an element

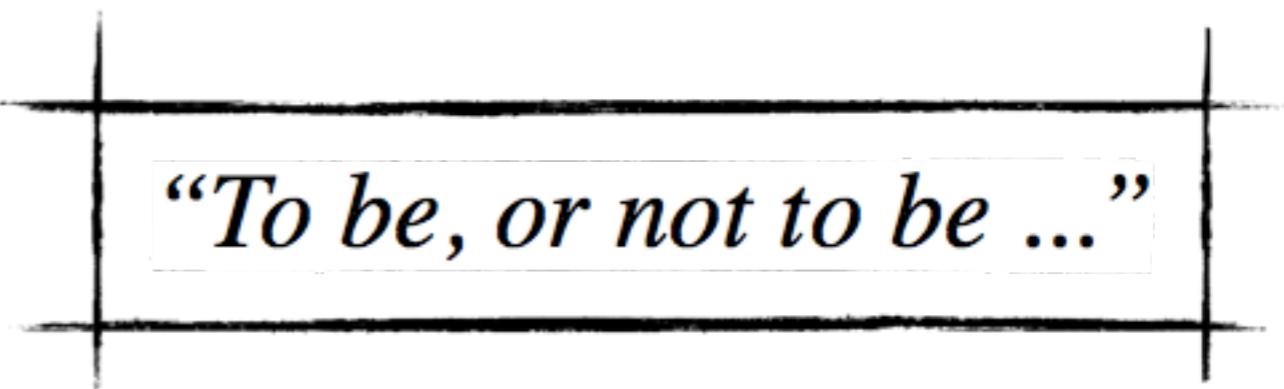
::after add (cosmetic) content after an element

!! pseudo-element is the first child of the selected element

```
<cite>  
  To be, or not  
  to be ...  
</cite>
```

Canonical example:
add quotation marks to quotes

```
cite::before {  
  content: "\201C";  
}  
cite::after {  
  content: "\201D";  
}
```



"To be, or not to be ..."

Before and after

::before add (cosmetic) content before an element

::after add (cosmetic) content after an element

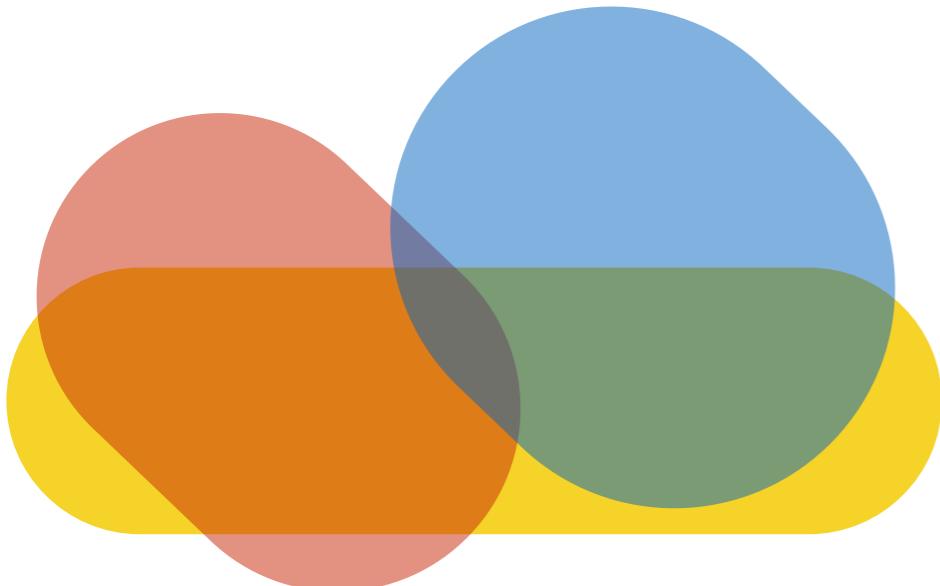
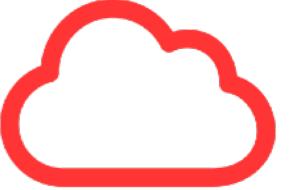
!! pseudo-element is the first child of the selected element

```
<cite>  
  To be, or not  
  to be ...  
</cite>
```

```
cite::before {  
  content: "\201C";  
}  
cite::after {  
  content: "\201D";  
}
```

Code [hide] ↗	Glyph ↗	Description	↗	# ↗
U+2013	–	En dash		0903
U+2014	—	Em dash	unicode	0904
U+2015	—	Horizontal bar		0905
U+2017	=	Double low line		0906
U+2018	‘	Left single quotation mark		0907
U+2019	’	Right single quotation mark		0908
U+201A	,	Single low-9 quotation mark		0909
U+201B	‘	Single high-reversed-9 quotation mark		0910
U+201C	“	Left double quotation mark		0911
U+201D	”	Right double quotation mark		0912

Before and after



`.cloud { ... }`

A rectangle with rounded edges.

`.cloud::before { ... }`

A rectangle with rounded edges that is rotated 33 degrees and moved slightly up and to the right of its parent.

`.cloud::after { ... }`

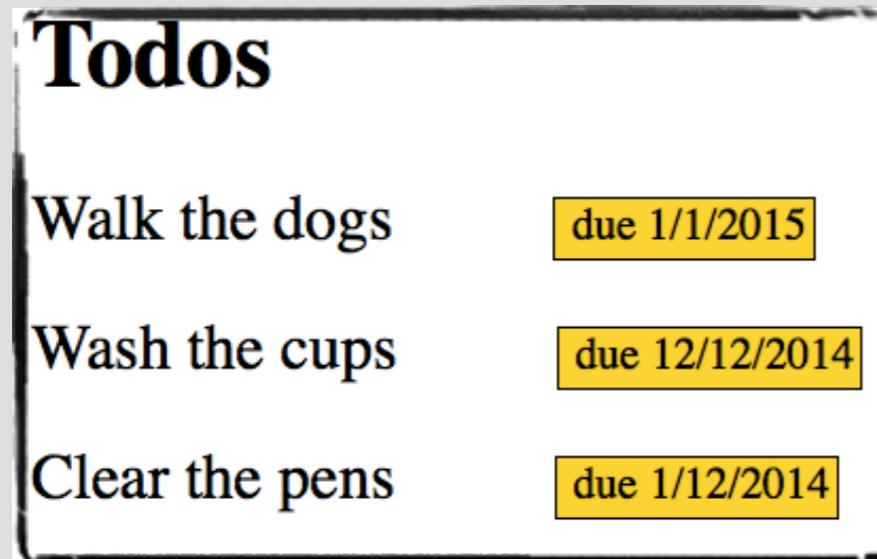
A rectangle with rounded edges that is rotated 33 degrees and moved up and to the right of its parent.

Data in CSS

CSS & data (one way)

CSS does not only describe the style, it can carry data too.

```
<main>
  <h2>Todos</h2>
  <p id="t1">Walk the dogs</p>
  <p id="t2">Wash the cups</p>
  <p id="t3">Clear the pens</p>
</main>
```



```
p::after {
  background-color: gold;
  border: 1px solid;
  font-size: 70%;
  padding: 2px;
  margin-left: 50px;
}

p#t1::after {
  content: " due 1/1/2015";
}

p#t2::after {
  content: " due 12/12/2014";
}

p#t3::after {
  content: " due 1/12/2014";
}
```

CSS & data (one way)

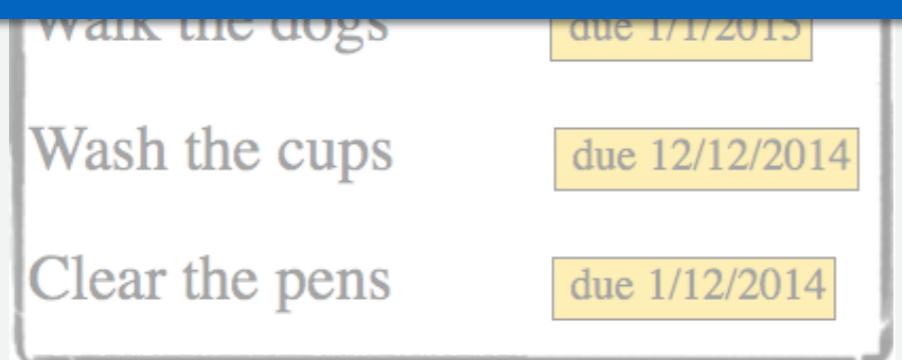
CSS does not only describe the style, it can carry data too.

```
<main>  
<h2>Todos</h2>
```

```
p::after {  
background-color: gold;  
border: 1px solid;
```

Issues:

1. Data is **distributed** across HTML and CSS files.
2. CSS is conventionally not used to store data.
3. **Content is not part of the DOM** (accessibility problem)



```
content: " due 12/12/2014 ",  
}  
  
p#t3::after {  
content: " due 1/12/2014";  
}
```

CSS & data-* (the preferred way)

CSS can make use of **data stored in HTML elements**.

Valid HTML elements can have data-* attributes.

```
<main>
  <h2>Todos</h2>
  <p id="t1" data-due="1/1/2015">Walk the dogs</p>
  <p id="t2" data-due="12/12/2014">Wash the cups</p>
  <p id="t3" data-due="1/12/2014">Clear the pens</p>
</main>
```

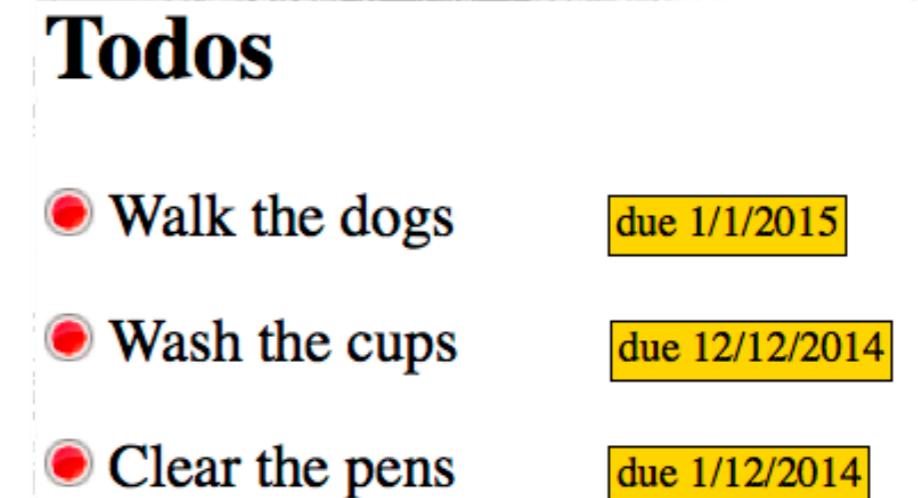
```
p::after {
  background-color: gold;
  border: 1px solid;
  font-size: 70%;
  padding: 2px;
  margin-left: 50px;
  content: "due " attr(data-due);
}

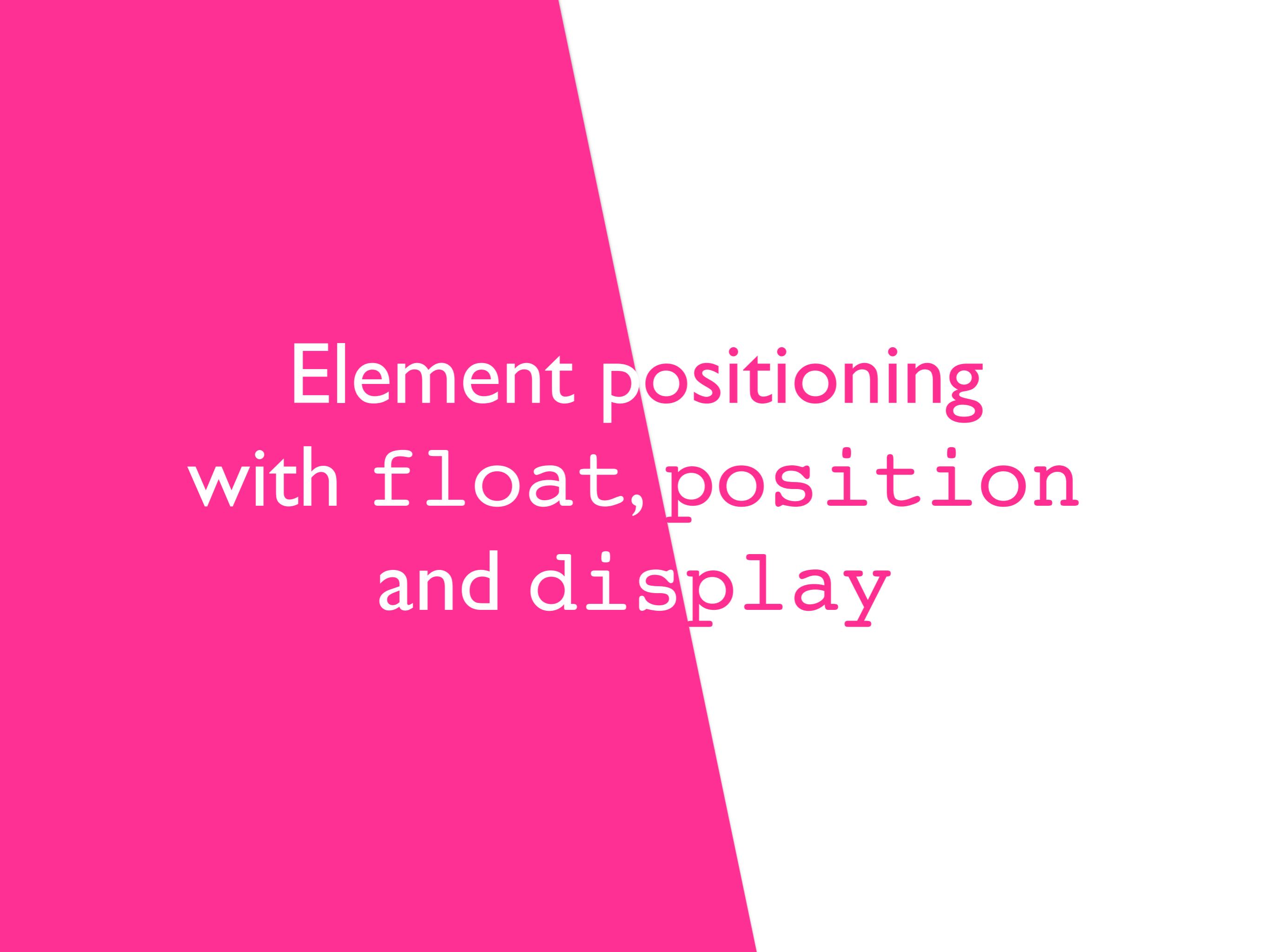
p::before {
  content: attr(data-due);
}
```

attr() retrieves the value of an attribute

content attribute can reference a url

```
content: url(http://www.abc.de/dot.png);
```





Element positioning with float, position and display

What does this mean?

Property	Description
float	defines how an element floats in the containing element ; this determines how other elements flow around it
position	defines how an element is positioned in a document
display	defines the display type of an element

Elements “flow” by default

Block-level are surrounded by line-breaks. They can contain block-level and inline elements. The **width is determined by their containing element.**

Examples: <main> or <p>

Inline elements can be placed within block/inline elements. They can contain other inline elements. The **width is determined by their content.**

Examples: or <a>

```
<main>
  <p>This is a paragraph containing <a href="#">a link</a></p>
  <p>
    This is another paragraph
    <span> with a span and <a href="#">a link in the span</a> </span>
  </p>
</main>
```

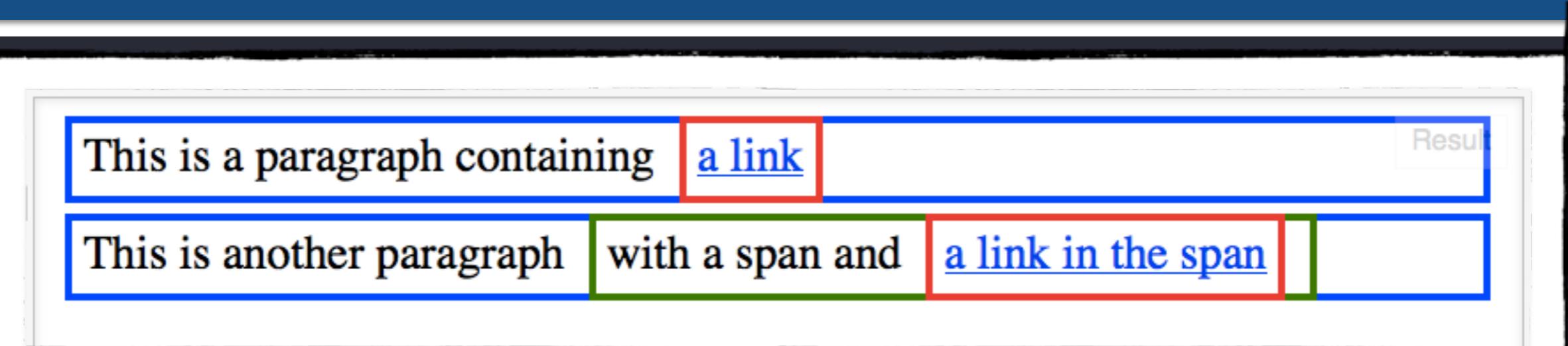
Elements “flow” by default

Block-level are surrounded by line-breaks. They can contain block-level and inline elements. The **width is determined by their containing element.**

Examples: <main> or <p>

Inline elements can be placed within block/inline elements. They can contain other inline elements. The **width is determined by their content.**

Examples: or <a>



```
<main>
  This is a paragraph containing a link
  This is another paragraph with a span and a link in the span
</main>
```

1 main {width: auto;}

Elements “flow” by default

Block-level are surrounded by line-breaks. They can contain block-level and inline elements. The **width is determined by their containing element.**

Examples: <main> or <p>

Inline elements can be placed within block/inline elements. They can contain other inline elements. The **width is determined by their content.**

Examples: or <a>

The screenshot shows the browser's developer tools with the element inspector open. The main element (<main>) has a blue border and contains two paragraphs. The first paragraph has a blue border and contains a link with a red border. The second paragraph has a green border and contains a span with a red border containing a link. The 'Result' tab on the right shows the rendered HTML code.

```
<ma  
<  
<  
This is a paragraph containing a link  
This is another paragraph with a span and  
a link in the span  
</p>  
</main>
```

```
1 main {width: 400px;}
```

Taking elements out of the flow

float:left (or **:right**) takes an element out of the flow; it is moved to the leftmost (or rightmost) possible position **in the containing element** —either the element edge or another float.

Taking elements out of the flow

float:left (or **:right**) takes an element out of the flow; it is moved to the leftmost (or rightmost) possible position **in the containing element** —either the element edge or another float.

This is a paragraph containing [a link](#)

This is another paragraph with a span and [a link in the span](#)

```
1 a {float: none;}
```

Taking elements out of the flow

float:left (or **:right**) takes an element out of the flow; it is moved to the leftmost (or rightmost) possible position **in the containing element** —either the element edge or another float.

The screenshot shows a browser's developer tools interface with a 'Result' panel. Inside the panel, there are two paragraphs. The first paragraph contains the text 'a link' followed by 'This is a paragraph containing'. The second paragraph contains the text 'a link in the span' followed by 'This is another paragraph with a span and'. The word 'link' in both instances is highlighted with a red border. The word 'span' in the second paragraph is highlighted with a green border. A blue border surrounds the entire content area of the Result panel. At the bottom right of the panel, there is a code snippet: '1 a {float: left;}'.

Taking elements out of the flow

float:left (or **:right**) takes an element out of the flow; it is moved to the leftmost (or rightmost) possible position **in the containing element** —either the element edge or another float.

This is a paragraph containing [a link](#)

This is another paragraph with a span and [a link in the span](#)

```
1 a {float: right;}
```

Resetting the flow with clear

Canonical example: **adding sidebars** to a layout

The image shows a screenshot of a web page with a dark grey header bar at the top. Below it is a light grey sidebar containing two columns of lists. The left column lists items from 'Go to page 1' to 'Go to page 9'. The right column lists months from 'January 2014' to 'April 2014'. Below the sidebar is a pink main content area containing two paragraphs: 'This is the 1. paragraph' and 'This is the 2. paragraph'. At the bottom is a green footer bar with the text 'And this is footer information'. Labels are overlaid on the sidebar: 'nav#nav1' is positioned above the first list, and 'nav#nav2' is positioned above the second list. A label 'main footer' is positioned below the pink area, aligned with the green footer.

- Go to page 1
- Go to page 2
- Go to page 3
- Go to page 4
- Go to page 5
- Go to page 6
- Go to page 7
- Go to page 8
- Go to page 9

nav#nav1

- January 2014
- February 2014
- March 2014
- April 2014

nav#nav2

This is the 1. paragraph

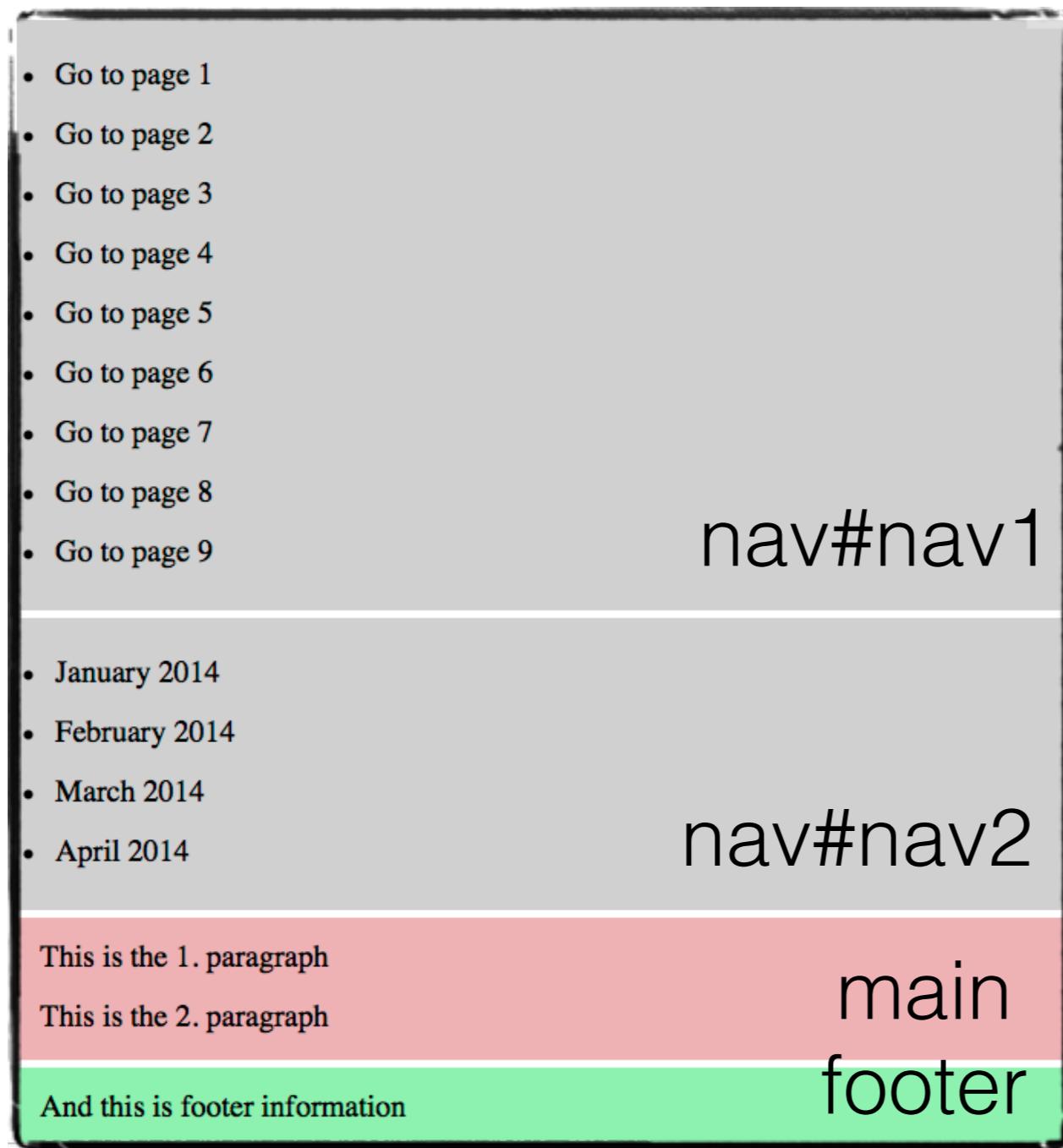
This is the 2. paragraph

And this is footer information

main
footer

Resetting the flow with clear

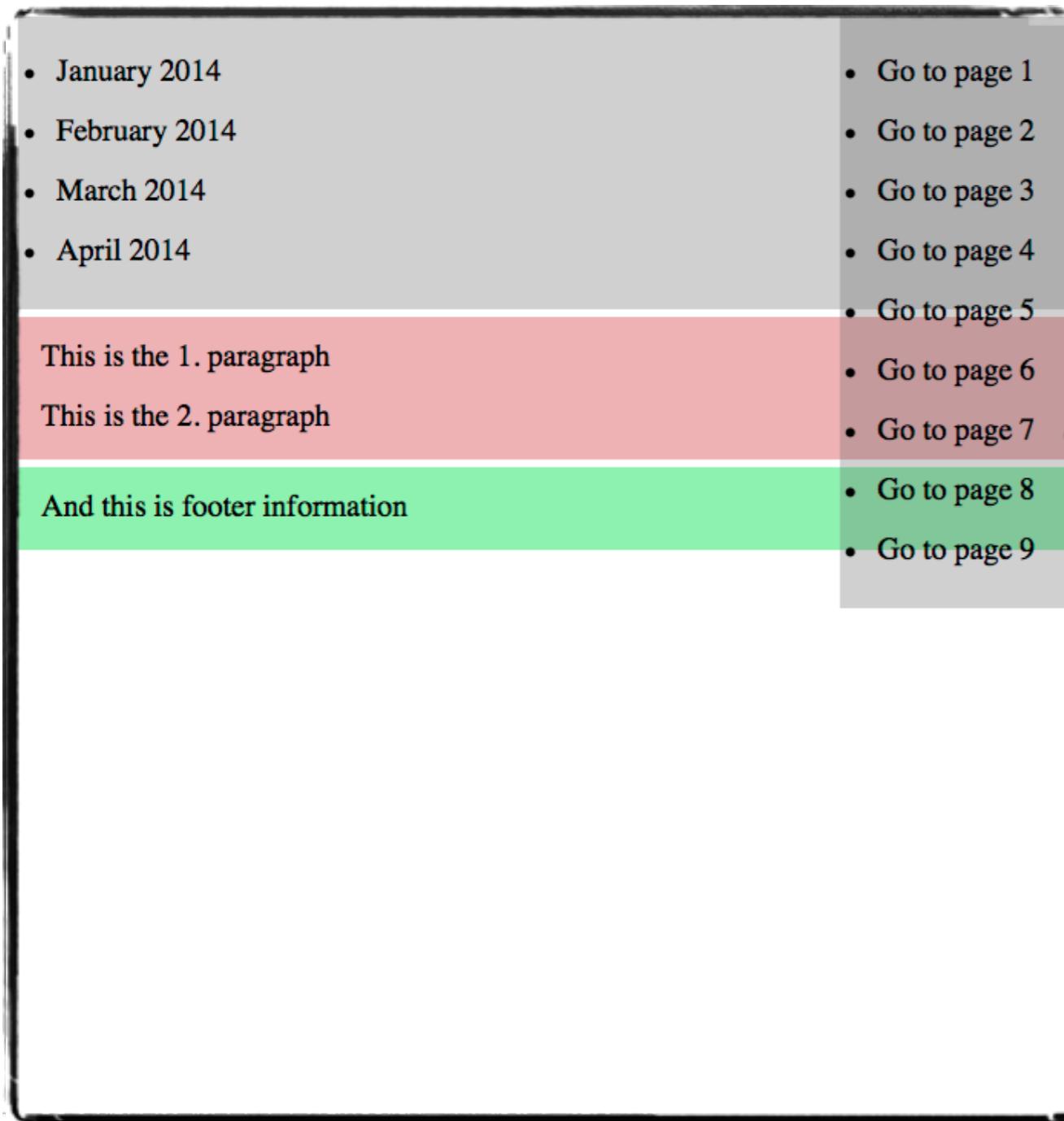
Canonical example: **adding sidebars** to a layout



```
1 #nav1 {float: right;}
```

Resetting the flow with clear

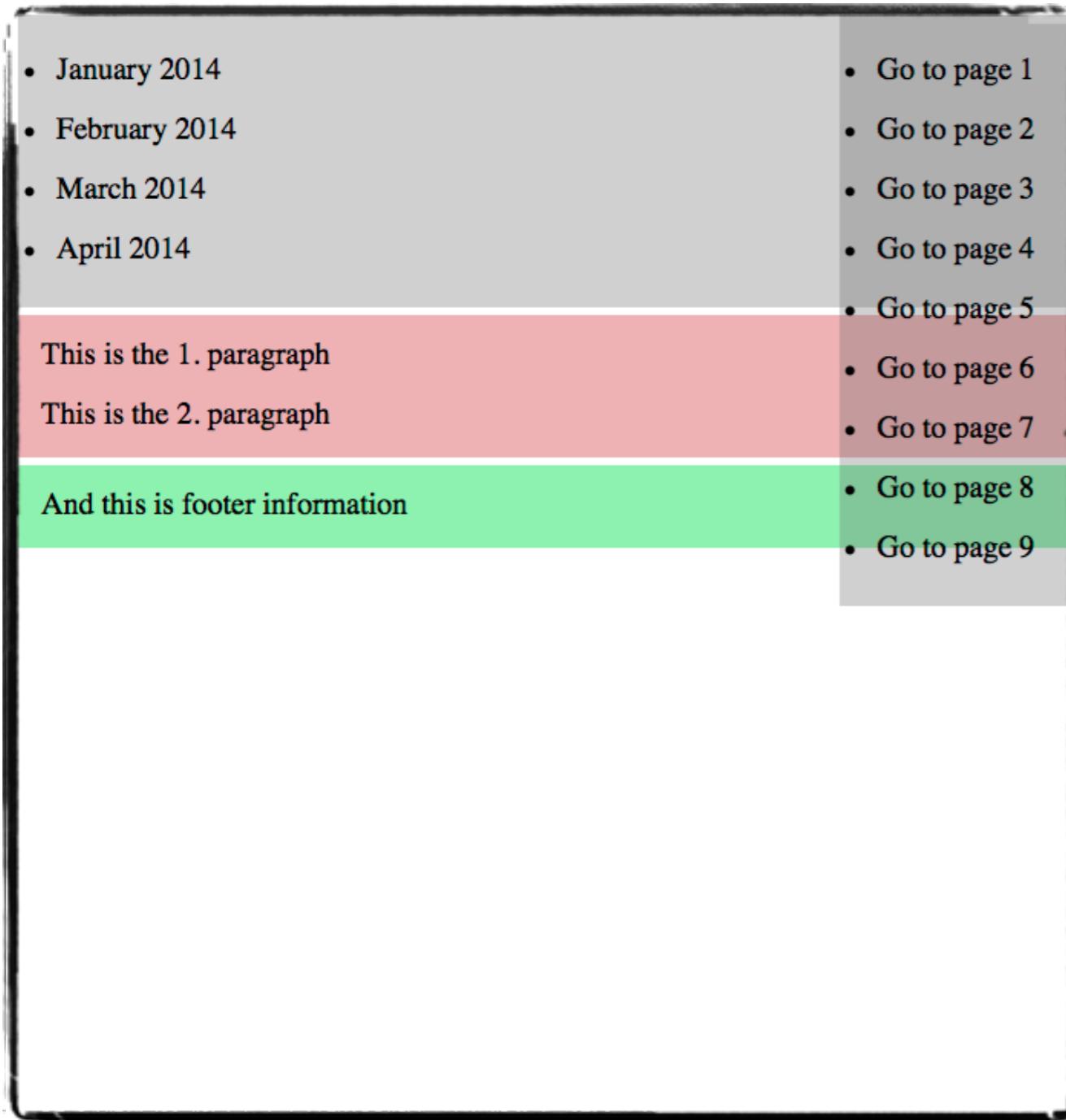
Canonical example: **adding sidebars** to a layout



```
1 #nav1 {float: right;}
```

Resetting the flow with clear

Canonical example: adding sidebars to a layout

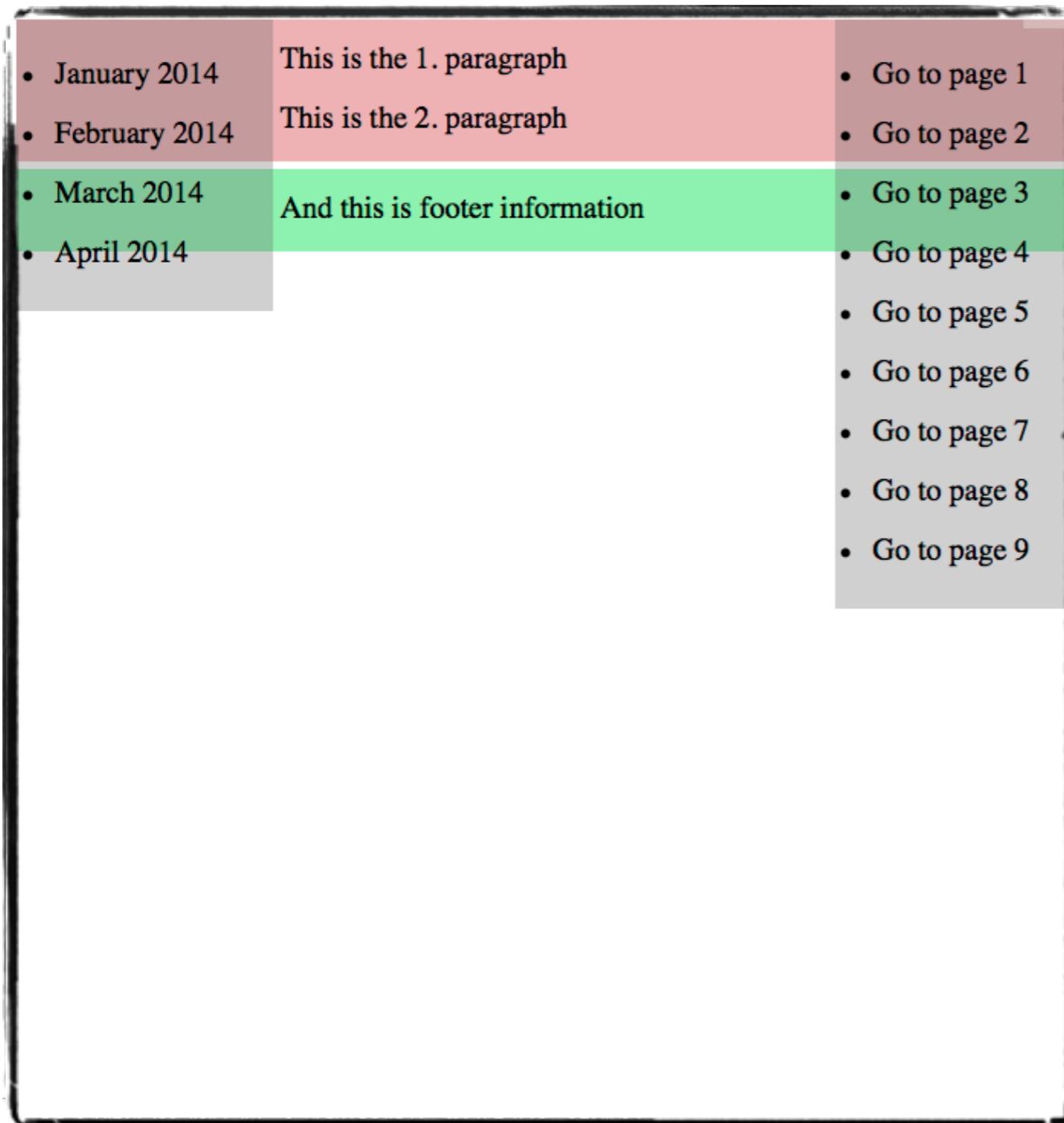


```
1 #nav1 {float: right;}
```

```
2 #nav2 {float: left;}
```

Resetting the flow with clear

Canonical example: **adding sidebars** to a layout

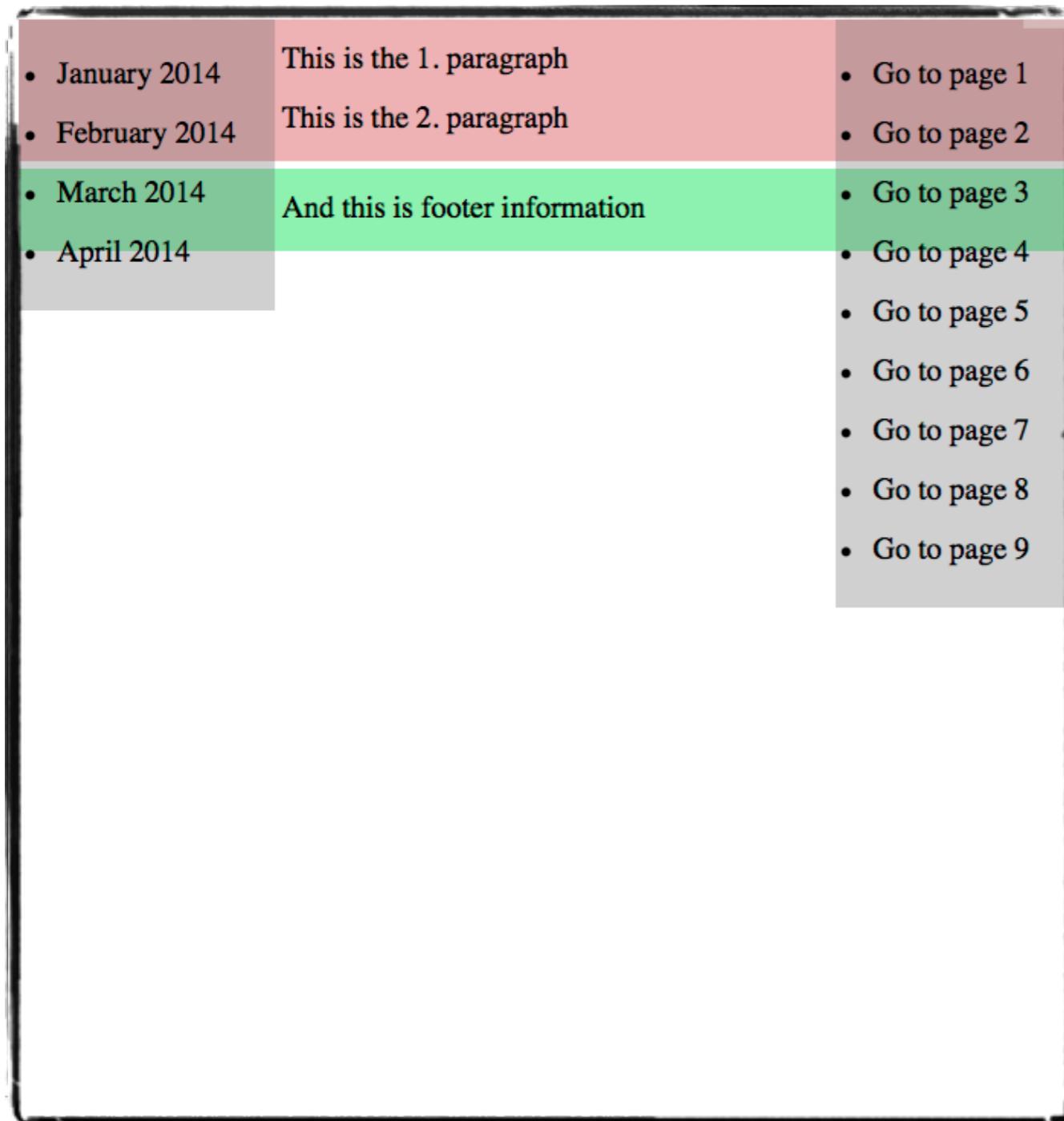


```
1 #nav1 {float: right;}
```

```
2 #nav2 {float: left;}
```

Resetting the flow with clear

Canonical example: adding sidebars to a layout



```
1 #nav1 {float: right;}  
2 #nav2 {float: left;}  
3 footer{clear: left;}
```

Resetting the flow with clear

Canonical example: adding sidebars to a layout

Result

• January 2014	This is the 1. paragraph	• Go to page 1
• February 2014	This is the 2. paragraph	• Go to page 2
• March 2014		• Go to page 3
• April 2014		• Go to page 4
		• Go to page 5
		• Go to page 6
		• Go to page 7
		• Go to page 8
		• Go to page 9

And this is footer information

```
1 #nav1 {float: right;}  
2 #nav2 {float: left;}  
3 footer{clear: left;}
```

Resetting the flow with clear

Canonical example: adding sidebars to a layout

The screenshot shows a layout with three main sections: a sidebar on the left, a main content area in the center, and a footer at the bottom. The sidebar contains a list of months from January 2014 to April 2014. The main content area contains two paragraphs: "This is the 1. paragraph" and "This is the 2. paragraph". To the right of the main content, there is a vertical list of links labeled "Go to page 1" through "Go to page 9". A green horizontal bar at the bottom of the main content area contains the text "And this is footer information". The word "Result" is visible in the top right corner of the browser window.

```
1 #nav1 {float: right;}  
2 #nav2 {float: left;}  
3 footer{clear: left;}  
4 footer{clear: right;}
```

Resetting the flow with clear

Canonical example: adding sidebars to a layout

The diagram illustrates a layout with floating elements. On the left, there is a sidebar with a list of months from January to April. The main content area contains two paragraphs: "This is the 1. paragraph" and "This is the 2. paragraph". To the right of the content is another sidebar with a list of links from "Go to page 1" to "Go to page 9". At the bottom is a green footer bar with the text "And this is footer information".

```
1 #nav1 {float: right;}  
2 #nav2 {float: left;}  
3 footer{clear: left;}  
4 footer{clear: right;}
```

Resetting the flow with clear

Canonical example: adding sidebars to a layout

This is the 1. paragraph

This is the 2. paragraph

- January 2014
- February 2014
- March 2014
- April 2014

- Go to page 1
- Go to page 2
- Go to page 3
- Go to page 4
- Go to page 5
- Go to page 6
- Go to page 7
- Go to page 8
- Go to page 9

And this is footer information

```
1 #nav1 {float: right;}  
2 #nav2 {float: left;}  
3 footer{clear: left;}  
4 footer{clear: right;}  
3 footer{clear: both;}
```

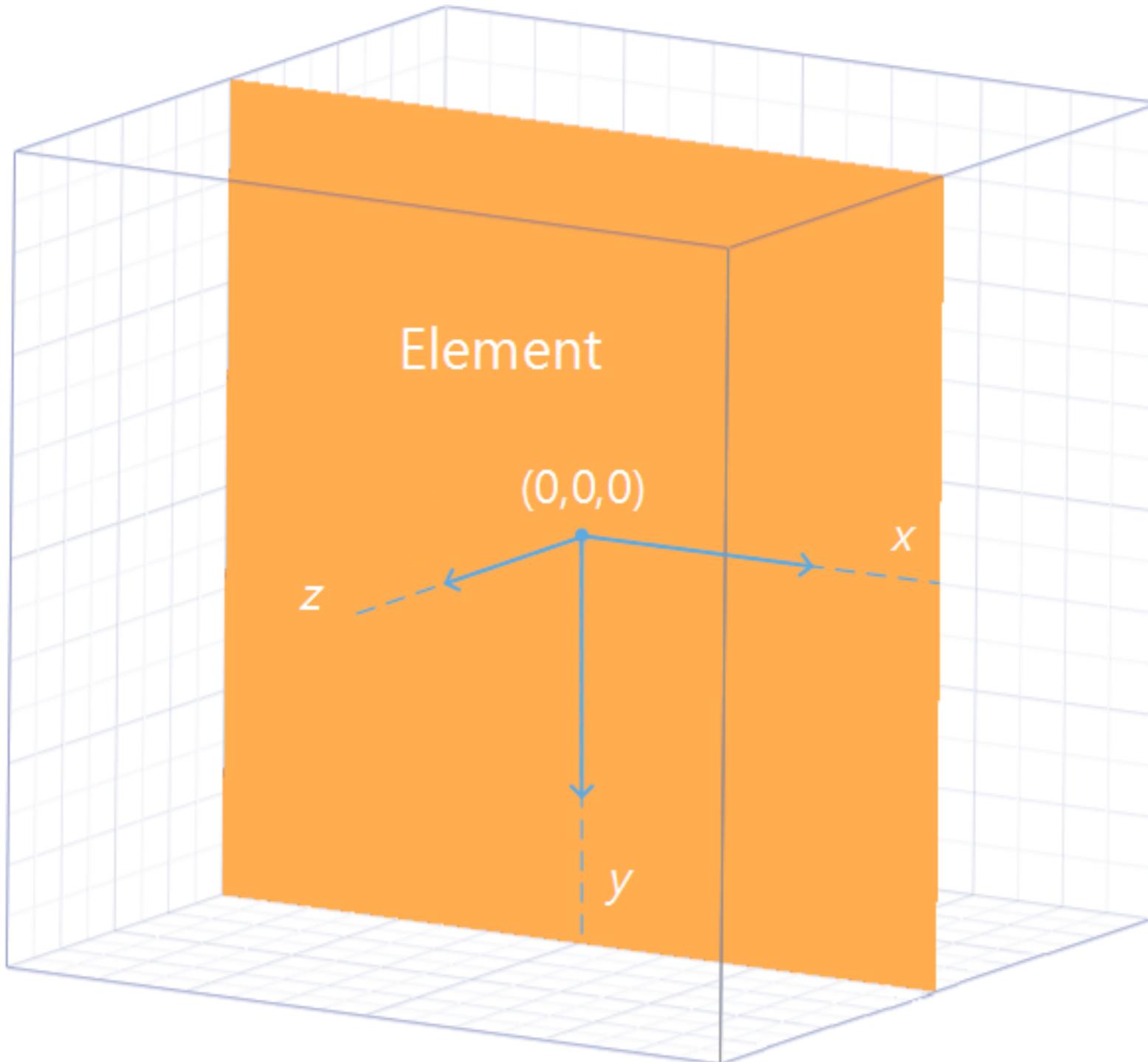
can be used instead

Fine-grained movement: position

position enables elements to be moved around in any direction (up/down/left/right) by absolute or relative units.

Property	Description
<code>position:static</code>	the default
<code>position:relative</code>	the element is adjusted on the fly, other elements are not affected
<code>position:absolute</code>	element is taken out of the normal flow (no space is reserved for it)
<code>position:fixed</code>	similar to absolute, but fixed to the viewport (area currently being viewed)
<code>position:sticky</code>	inbetween relative and fixed (we do not consider it further)

CSS coordinate system



y extends downward. **x** extends to the right.

position: relative

The element is adjusted on the fly, other elements are **not** affected.

The movement is **relative** to its original position.

The diagram shows four decorative eggs on the left and their corresponding CSS styles on the right. A blue arrow points from each egg to its respective style block. The first egg, labeled `id="egg1"`, has no styling. The second egg, labeled `id="egg2"`, has `position: relative;`, `bottom: 20px;`, and `left: 20px;`. The third egg, labeled `id="egg4"`, has `position: relative;`, `bottom: 50px;`, and `right: 10px;`. To the right, the eggs are shown in two states: their original positions (top row) and their new relative positions (bottom row). The second egg has moved down and to the right, while the third egg has moved down and to the left. The first and fourth eggs remain in their original positions, labeled "unchanged".

```
#egg2 {  
    position: relative;  
    bottom: 20px;  
    left: 20px;  
}  
  
#egg4 {  
    position: relative;  
    bottom: 50px;  
    right: 10px;  
}
```

`id="egg1"`

`id="egg4"`

Distance the element's bottom edge moves *above* its position.

Distance the element's left edge is moved to the *right* from its position.

unchanged

position: absolute

The element is **taken out of the normal flow** (no space is reserved).

The positioning is relative to nearest ancestor or the window.



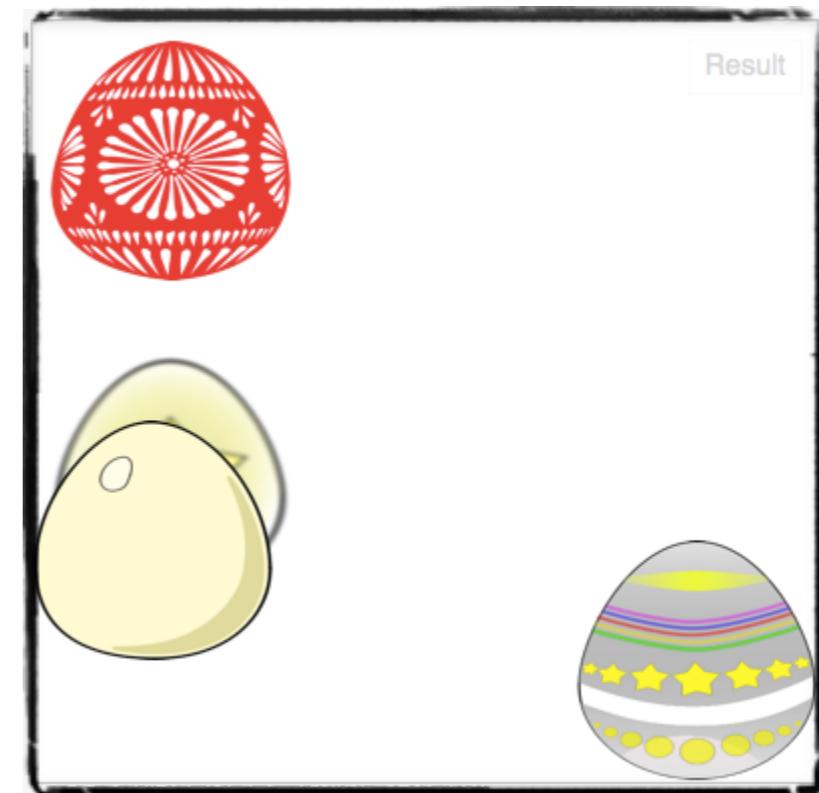
`id="egg1"`

```
#egg2 {  
    position: absolute;  
    bottom: 50px;  
    left: 0px;  
}
```

```
#egg4 {  
    position: absolute;  
    bottom: 0px;  
    right: 0px;  
}
```

`id="egg4"`

window



Distance between the element's bottom edge and that of its containing block.

Distance between the element's left edge and that of its containing block.

position:fixed

Similar to `absolute`, but the containing “element” is the viewport.
Elements with `position:fixed` are always visible.

area of the document visible in the browser

```
#swingingBoards {  
    text-align: center;  
    position: fixed;  
    /*top and left determine where fixed position is*/  
    top: 50%;  
    left: 40%;  
}
```

display

display:inline

element rendered with an **inline** element box

display:block

element rendered with a **block** element box

display:none

element (and its descendants) are **hidden** from view; no space is reserved in the layout

display:inline-block block element box that flows as inline el.

display

display:inline

element rendered with an **inline** element box

display:block

element rendered with a **block** element box

display:none

element (and its descendants) are **hidden** from view; no space is reserved in the layout

display:inline-block block element box that flows as inline el.

This is paragraph one.

Span element one. Span element two. Span element three.

This is paragraph two.

display

display:inline

element rendered with an **inline** element box

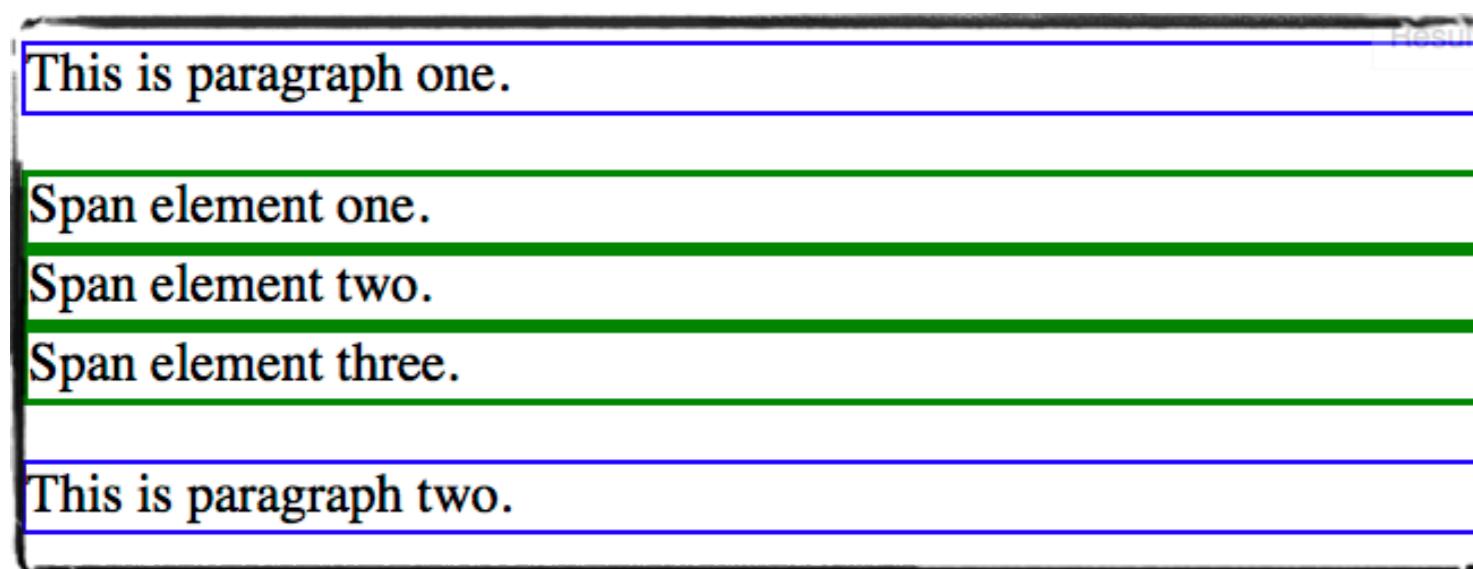
display:block

element rendered with a **block** element box

display:none

element (and its descendants) are **hidden** from view; no space is reserved in the layout

display:inline-block block element box that flows as inline el.



```
1 span {display: block; }
```

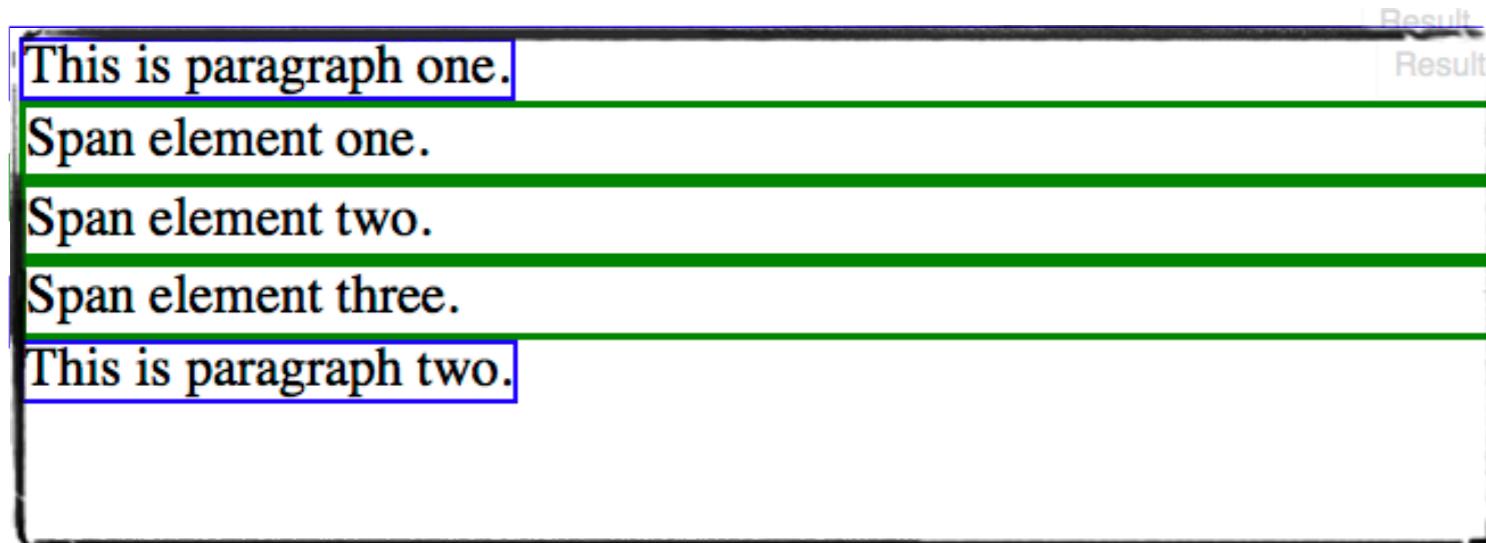
display

display:inline element rendered with an **inline** element box

display:block element rendered with a **block** element box

display:none element (and its descendants) are **hidden** from view; no space is reserved in the layout

display:inline-block block element box that flows as inline el.



```
1 span {display: block; }  
2 p {display: inline; }
```

display

display:inline

element rendered with an **inline** element box

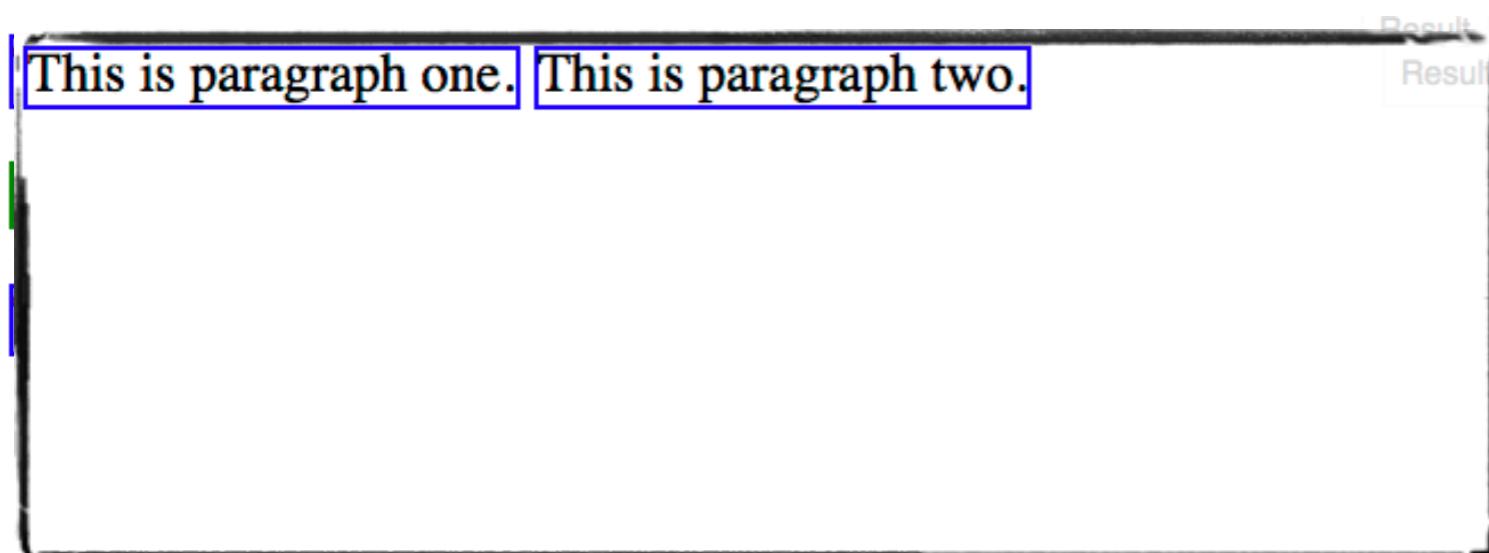
display:block

element rendered with a **block** element box

display:none

element (and its descendants) are **hidden** from view; no space is reserved in the layout

display:inline-block block element box that flows as inline el.



```
1 span {display: block; }
2 p     {display: inline; }
3 span {display: none; }
```

CSS media queries



Not just one device but many...

- Different devices should be served different styles
 - **Printing**: print the essentials only (no ads, sidebars, in black and white)
 - **Viewing** on a **small screen**: remove non-essential information
 - **Viewing** on a **large screen**: present all information
 - **Text-to-speech** devices: remove non-essential information
- CSS media queries enable the use of **device-dependent stylesheets**

HTML: write once

CSS: write once per device

Four device types

Value	Description
media all	Suitable for all device types.
media print	Suitable for documents in print preview.
media screen	Suitable for screens.
media speech	Suitable for speech synthesizers.

at-rules starting with @

responsive design mode

Media queries can be complex

```
<link  
  rel="stylesheet"  
  media="screen and (min-width: 800px),  
        (min-width: 3000px)"  
  href="large-device.css"  
/>  
...  
<style>  
  @media print {  
    body {  
      color: black !important;  
      width: 100%;  
    }  
  }  
  @media screen and (max-width: 300px) {  
    #sidebar {  
      display: none;  
    }  
  }  
</style>
```

“ , ”: logical or
dedicated CSS files
rules for different devices in one file
when printing, use black and white
hide the sidebar for small devices

Animations and transitions

Rendering engines

(layout engine, browser engine)

Engine	Browsers
Gecko	Firefox
Trident	Internet Explorer
EdgeHTML	Microsoft Edge
WebKit	Safari, older versions of Google Chrome
Blink	Google Chrome, Opera

Rendering engines

(layout engine, browser engine)

Engine	Browsers
Gecko	Firefox
Trident	Internet Explorer
EdgeHTML	Microsoft Edge
WebKit	Safari, older versions
Blink	Google Chrome,

 **Sabri Haddouche**
@pwnsdx

How to force restart any iOS device with just CSS? 💣

Source: gist.github.com/pwnsdx/ce64de2...

IF YOU WANT TO TRY (DON'T BLAME ME IF YOU CLICK) : cdn.rawgit.com/pwnsdx/ce64de2...

 **GitHub Gist** · · · · ·

Safari Dos 💀 (Original tweet: <https://twitter.com/pwnsdx/status/1040944750...>)
Safari Dos 💀 (Original tweet: <https://twitter.com/pwnsdx/status/1040944750973595649>, try it: <https://reaperbugs.com/>, mirror by ...
gist.github.com

2:45 PM · Sep 15, 2018 · Tweetbot for iOS

1.7K Retweets 2.6K Likes

In general ...

- CSS styles (states) are defined by the developer, the rendering engine takes care of the transition
- **Animations** consist of:
 - An animation **style** (linear, 3 seconds duration, 10 times, etc.)
 - **Keyframes** acting as transition waypoints
- **Transitions** are animations that consist of exactly two states: start and end state

CSS vs. JavaScript animations

- Rendering engines are **optimised** for CSS-based animations
- CSS animations can do much more than animating buttons



Lightning animation

```
.lightning {  
    /* the background-image should be filled with lightning;  
     */  
    position: absolute;  
    width: 100%;  
    height: 100%;  
  
    animation: flash ease-out 5s infinite;  
}  
  
animation name
```

animation properties

```
@keyframes flash {  
    /* basic idea: the white div covering the entire screen appears  
     * very briefly with varying levels of opacity;  
     */  
  
    from { opacity: 0; }  
    74% { opacity: 0; }  
    75% { background-color: white; opacity: 0.6; }  
    76% { background-color: white; opacity: 0.2; }  
    80% { background-color: white; opacity: 0.95; }  
    82% { opacity: 0; }  
    92% { opacity: 0; }  
    93% { background-color: white; opacity: 0.5; }  
    94% { background-color: white; opacity: 0.2; }  
    96% { background-color: white; opacity: 0.9; }  
    to { opacity: 0; }  
}
```

Define the states of the animation, the rendering engine does the rest. 0%-100% is one animation cycle.

from = 0%

to = 100%

Moon animation

```
#moon {  
  position: absolute;  
  left: 60px;  
  top: 40px;  
  border-radius: 50%; /* making it round */  
  height: 100px;  
  width: 100px;  
  border: 1px solid black;  
  
  background: radial-gradient(circle at 20px 10px, #333, #fff);  
  box-shadow: 0 0 25px #fff;  
  animation: flicker 3s infinite;  
}  
  
@keyframes flicker {  
  0% {  
    box-shadow: 0 0 25px #fff;  
  }  
  25% {  
    box-shadow: 0 0 60px #fff;  
  }  
  50% {  
    box-shadow: 0 0 50px #fff;  
  }  
  90% {  
    box-shadow: 0 0 27px #fff;  
  }  
}
```

animation properties

animation name

Define the states of
the animation, the
rendering engine does
the rest.

Cloud animation

```
/* simple animation: clouds go from left to right */
@keyframes moveclouds {
  0% {margin-left: -20%;}
  100% {margin-left: 120%;}
}

function toggleClouds(e) {
  let clouds = document.getElementById("clouds");

  if (document.getElementById("cloudsCheckbox").checked == false) {
    while (clouds.firstChild) {
      clouds.removeChild(clouds.firstChild);
    }
  } else {
    for (let i = 1; i <= totalNumClouds; i++) {
      let c = document.createElement("div");
      c.classList.add("cloud");
      c.style.top = 50 + getRandomInt(120) + "px";
      c.style.opacity = Math.random();
      c.style.animationDuration = 15 + getRandomInt(20) + "s";
      c.style.animationDelay = -1 * getRandomInt(40) + "s";
      c.style.transform = "scale(" + (0.1 + Math.random()) + ")";
      clouds.appendChild(c);
    }
  }
}
```

Cloud animation

```
/* simple animation: clouds go from left to right */
@keyframes moveclouds {
  0% {margin-left: -20%;}
  100% {margin-left: 120%;}
}

function toggleClouds(e) {
  let clouds = document.getElementById("clouds");

  if (document.getElementById("cloudsCheckbox").checked == false) {
    while (clouds.firstChild) {
      clouds.removeChild(clouds.firstChild);
    }
  } else {
    for (let i = 1; i <= totalNumClouds; i++) {
      let c = document.createElement("div");
      c.classList.add("cloud");
      c.style.top = 50 + getRandomInt(120) + "px";
      c.style.opacity = Math.random();
      c.style.animationDuration = 15 + getRandomInt(20) + "s";
      c.style.animationDelay = -1 * getRandomInt(40) + "s";
      c.style.transform = "scale(" + (0.1 + Math.random()) + ")";
      clouds.appendChild(c);
    }
  }
}
```

1 cloud = 1 div

Cloud animation

```
/* simple animation: clouds go from left to right */  
@keyframes moveclouds {  
  0% {margin-left: -20%;}  
  100% {margin-left: 120%;}  
}
```

```
function toggleClouds(e) {  
  let clouds = document.getElementById("clouds");  
  
  if (document.getElementById("cloudsCheckbox").checked == false) {  
    while (clouds.firstChild) {  
      clouds.removeChild(clouds.firstChild);  
    }  
  } else {  
    for (let i = 1; i <= totalNumClouds; i++) {  
      let c = document.createElement("div");  
      c.classList.add("cloud");  
      c.style.top = 50 + getRandomInt(120) + "px";  
      c.style.opacity = Math.random();  
      c.style.animationDuration = 15 + getRandomInt(20) + "s";  
      c.style.animationDelay = -1 * getRandomInt(40) + "s";  
      c.style.transform = "scale(" + (0.1 + Math.random()) + ")";  
      clouds.appendChild(c);  
    }  
  }  
}
```

1 cloud = 1 div

same partial css for every cloud

Cloud animation

```
/* simple animation: clouds go from left to right */  
@keyframes moveclouds {  
  0% {margin-left: -20%;}  
  100% {margin-left: 120%;}  
}
```

```
function toggleClouds(e) {  
  let clouds = document.getElementById("clouds");  
  
  if (document.getElementById("cloudsCheckbox").checked == false) {  
    while (clouds.firstChild) {  
      clouds.removeChild(clouds.firstChild);  
    }  
  } else {  
    for (let i = 1; i <= totalNumClouds; i++) {  
      let c = document.createElement("div");  
      c.classList.add("cloud");  
      c.style.top = 50 + getRandomInt(120) + "px";  
      c.style.opacity = Math.random();  
      c.style.animationDuration = 15 + getRandomInt(20) + "s";  
      c.style.animationDelay = -1 * getRandomInt(40) + "s";  
      c.style.transform = "scale(" + (0.1 + Math.random()) + ")";  
      clouds.appendChild(c)  
    }  
  }  
}
```

1 cloud = 1 div

same partial css for every cloud

transform property: rotate, skew, scale, translate

Cloud animation

```
/* simple animation: clouds go from left to right */  
@keyframes moveclouds {  
  0% {margin-left: -20%;}  
  100% {margin-left: 120%;}  
}
```

```
function toggleClouds(e) {  
  let clouds = document.getElementById("clouds");  
  
  if (document.getElementById("cloudsCheckbox").checked == false) {  
    while (clouds.firstChild) {  
      clouds.removeChild(clouds.firstChild);  
    }  
  } else {  
    for (let i = 1; i <= totalNumClouds; i++) {  
      let c = document.createElement("div");  
      c.classList.add("cloud");  
      c.style.top = 50 + getRandomInt(120) + "px";  
      c.style.opacity = Math.random();  
      c.style.animationDuration = 15 + getRandomInt(20) + "s";  
      c.style.animationDelay = -1 * getRandomInt(40) + "s";  
      c.style.transform = "scale(" + (0.1 + Math.random()) + ")";  
      clouds.appendChild(c)  
    }  
  }  
}
```

1 cloud = 1 div

same partial css for every cloud

Every cloud gets animation specific property values (speed, opacity, size, starting location)

transform property: rotate, skew, scale, translate

Cloud animation

```
/* simple animation: clouds go from left to right */  
@keyframes moveclouds {  
  0% {margin-left: -20%;}  
  100% {margin-left: 120%;}  
}
```

```
function toggleClouds(e) {  
  let clouds = document.getElementById("clouds");  
  
  if (document.getElementById("cloudsCheckbox").checked == false) {  
    while (clouds.firstChild) {  
      clouds.removeChild(clouds.firstChild);  
    }  
  } else {  
    for (let i = 1; i <= totalNumClouds; i++) {  
      let c = document.createElement("div");  
      c.classList.add("cloud");  
      c.style.top = 50 + getRandomInt(120) + "px";  
      c.style.opacity = Math.random();  
      c.style.animationDuration = 15 + getRandomInt(20) + "s";  
      c.style.animationDelay = -1 * getRandomInt(40) + "s";  
      c.style.transform = "scale(" + (0.1 + Math.random()) + ")";  
      clouds.appendChild(c)  
    }  
  }  
}  
Beware: animation-delay (CSS) vs. animationDelay (JavaScript)!
```

1 cloud = 1 div

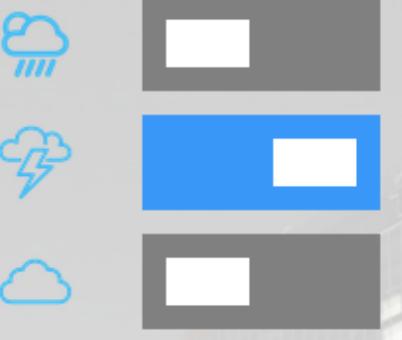
same partial css for every cloud

Every cloud gets animation specific property values (speed, opacity, size, starting location)

transform property: rotate, skew, scale, translate

CSS animation control

Property	Description
animation-iteration-count	Number of times an animation is executed (default: 1); the value is either a positive number or <code>infinite</code> .
animation-direction	By default the animation restarts at the starting keyframe; if set to <code>alternate</code> the animation direction changes every iteration.
animation-delay	Number of seconds (s) or milliseconds (ms) until the animation starts (default 0s).
animation-name	Identifies the <code>@keyframes</code> .
animation-duration	The duration of a single animation cycle in seconds (s) or milliseconds (ms).
animation-timing-function	Specifies how a CSS animation progresses between waypoints (common choices are <code>linear</code> , <code>ease-in</code> , <code>ease-out</code> , <code>steps(N)</code>).

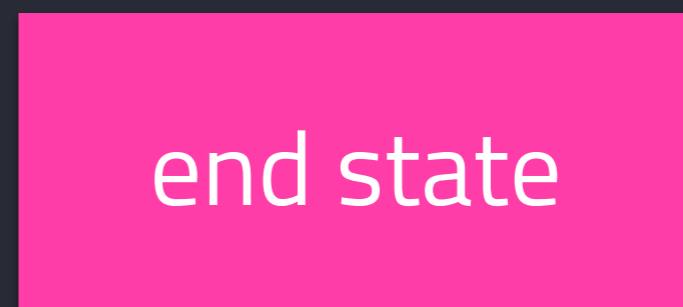


CSS transitions

```
.box {  
    border-style: solid;  
    border-width: 1px;  
    display: block;  
    width: 100px;  
    height: 100px;  
    background-color: red;  
    -webkit-transition: width 2s, height 2s, -webkit-transform 2s;  
    transition: width 2s, height 2s, transform 2s; }  
  
.box:hover {  
    background-color: green;  
    width: 200px;  
    height: 200px;  
    -webkit-transform: rotate(180deg);  
    transform: rotate(180deg); }
```



declare transition



We have been using (default) transitions all the time.

- Work on **Assignment 5**.
- Start working on **Assignment 6**.
- Tutorial topic this week: Node.js