

CSS: the language of Web design

Claudia Hauff

TI1506: Web and Database Technology

ti1506-ewi@tudelft.nl

At the end of this lecture, you should be able to ...

- **Position and style** HTML elements according to a given design of a Web page
- **Employ** pseudo-classes and pseudo-elements
- **Employ** CSS's data access/creation facilities and **reflect** upon them
- **Write** CSS media queries
- **Create** simple CSS-based animations

A bit of context

A brief history of CSS

- **CSS 1**: a W3C recommendation in **1996**
 - Support for fonts, colours, alignment, margins, ids and classes
- **CSS 2**: a W3C recommendation in **1998**
 - Support added for media queries, element positioning
- **CSS 2.1**: a W3C recommendation in 2011
 - Fixed errors and added support for features widely implemented in major browsers
- **CSS 3**: currently under development; specification is split up into modules; progress varies between modules
<http://www.w3.org/Style/CSS/current-work>
- **CSS 4**: some modules have reached “level 4” status

CSS 3+: a tale of many modules

Completed	Current	Upcoming
CSS Snapshot 2017	NOTE	
CSS Snapshot 2015	NOTE	
CSS Snapshot 2010	NOTE	
CSS Snapshot 2007	NOTE	
CSS Color Level 3	REC	REC
CSS Namespaces	REC	REC
Selectors Level 3	REC	REC
CSS Level 2 Revision 1	REC	REC
CSS Level 1	REC	
CSS Print Profile	NOTE	
Media Queries	REC	REC
CSS Style Attributes	REC	REC

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

CSS 3+: a tale of many modules

Stable	Current	Upcoming
CSS Backgrounds and Borders Level 3	CR	PR
CSS Conditional Rules Level 3	CR	CR
CSS Multi-column Layout Level 1	WD	CR
CSS Values and Units Level 3	CR	PR
CSS Cascading and Inheritance Level 3	CR	PR
CSS Fonts Level 3	CR	CR
CSS Writing Modes Level 3	CR	CR
CSS Counter Styles Level 3	CR	PR

Testing	Current	Upcoming
CSS Image Values and Replaced Content Level 3	CR	CR
CSS Speech	CR	CR
CSS Flexible Box Layout Level 1	CR	PR

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

CSS 3+: a tale of many modules

Non-element Selectors	FPWD	WD
CSS Inline Layout Level 3	WD	WD
Motion Path Level 1	WD	WD
CSS Round Display Level 1	WD	WD
CSS Basic User Interface Level 4	FPWD	WD
CSS Text Level 4	FPWD	WD
CSS Painting API Level 1	FPWD	WD
CSS Properties and Values API Level 1	FPWD	WD
CSS Typed OM Level 1	WD	WD
Worklets Level 1	FPWD	WD
CSS Color Level 4	FPWD	WD
CSS Fonts Level 4	WD	WD
CSS Rhythmic Sizing Level 1	FPWD	WD

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

CSS 3

- Impossible to write complex CSS that relies on modern features and works across all browsers
- Implementation of CSS 3 features should be decided based on
 - **intended users** (mostly in the US or China or ... ?)
 - the **mode of usage** (smartphone, touch-screen or ...?)
 - the **type** of Web app (are 3D animations necessary?)
- JavaScript libraries can help front-end developers to build cross-browser apps (e.g. `Modernizr`)

Revision: chapter 3

Chapter 3 of the course book

CSS describes how elements in the DOM should be rendered.

```
1 body {
2   background-color: #ffff00;
3   width: 800px;
4   margin: auto;
5 }
6 h1 {
7   color: maroon;
8 }
9 p span {
10  color: gray;
11  border: 1px solid gray;
12 }
13 p#last {
14  color: green;
15 }
```

selector

property

value

- Three types of style sheets:
(1) **browser's** style sheet
(2) **author's** style sheet
(3) **user's** style sheet



overrides

- Style sheets are processed in order; **later declarations override earlier ones** (if they are on the same level)
- **!important** overrides all other declarations

Pseudo-elements and pseudo-classes

(this means fewer “styling hooks”)

A detour: the rendering engine

- More than 30 pseudo-classes
- Support varies according to the **rendering engine**

A rendering engine (or browser engine, layout engine) is responsible for translating HTML+CSS (among others) to the screen.

Rendering engine	Browser
Gecko	Firefox
from Trident to EdgeHTML	Internet Explorer
WebKit	Safari, older versions of Google Chrome
Blink	Google Chrome (new versions), Opera

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).

```
1 selector:pseudo-class {  
2     property: value;  
3     property: value;  
4 }
```

Popular pseudo-classes

- :nth-child(x)** any element that is the X^{th} child element of its parent
- :nth-of-type(x)** any element that is the X^{th} sibling of its type

```
1 <main>
2   <h2>Todos</h2>
3   <p>Today's todos</p>
4   <p>Tomorrow's todos</p>
5   <p>Saturday's todos</p>
6   <p>Sunday's todos</p>
7 </main>
```

```
1 p:nth-child(2) {
2   color:red;
3 }
4
5 p:nth-of-type(2) {
6   background-color:green;
7 }
```


Popular pseudo-classes

`:nth-child(x)` any element that is the X^{th} child element of its parent

`:nth-of-type(x)` any element that is the X^{th} sibling of its type

2. child

```
1 <main>
2   <h2>Todos</h2>
3   <p>Today's todos</p>
4   <p>Tomorrow's todos</p>
5   <p>Saturday's todos</p>
6   <p>Sunday's todos</p>
7 </main>
```

parent

```
1 p:nth-child(2) {
2   color:red;
3 }
4
5 p:nth-of-type(2) {
6   background-color:green;
7 }
```

Popular pseudo-classes

Todos

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

:nth-child(X) any element that is the X^{th} child element of its parent

:nth-of-type(X) any element that is the X^{th} sibling of its **type** (X can be an int or formula, e.g. " $2n+1$ ")

n represents a number starting at 0 and incrementing

```
1 <main>
2   <h2>Todos</h2>
3   {
4   <p>Today's todos</p>
5   <p>Tomorrow's todos</p>
6   <p>Saturday's todos</p>
7   <p>Sunday's todos</p>
7 </main>
```

siblings

```
1 p:nth-child(2) {
2   color:red;
3 }
4
5 p:nth-of-type(2) {
6   background-color:green;
7 }
```

Popular pseudo-classes

:first-child is equivalent to `:nth-child(1)`
:last-child is equivalent to `:nth-last-child(1)`
:first-of-type is equivalent to `:nth-of-type(1)`
:last-of-type is equivalent to `:nth-last-of-type(1)`

Popular pseudo-classes

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

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

```
1 button {
2   background: white;
3   color: darkgray;
4   width: 100px;
5   padding: 5px;
6   font-weight: bold;
7   text-align: center;
8   border: 1px solid darkgray;
9 }
```

```
1 button:hover {
2   color: white;
3   background: darkgray;
4 }
5
6 button:active {
7   border: 1px dashed;
8   border-color: black;
9 }
```



ADD TODO



ADD TODO




ADD TODO

Popular pseudo-classes

:enabled an element that can be clicked/selected

:disabled an element that cannot be clicked/selected

```
1 button {  
2   ...  
3 }  
4  
5 button:enabled:hover {  
6   ...  
7 }  
8  
9 button:enabled:active {  
10  ...  
11 }
```

- Enabled/disabled buttons look the same
 - Enabled buttons change their look when being activated or at hovering
- 

Popular pseudo-classes

`:not(X)`

matches all elements that are not represented by selector X

```
1 <main>
2   <h2>Todos</h2>
3   <p id="firsttodo">Today's todos</p>
4   <p class="todo">Tomorrow's todos</p>
5   <p class="todo">Saturday's todos</p>
6   <p>Sunday's todos</p>
7 </main>
```

```
1 main *:not(.todo) {
2   color:orange;
3 }
```



e11 e12: Selects all <e12> elements inside <e11>

whitespace in selectors implies the universal selector: *

Popular pseudo-classes

:in-range **:out-of-range**

can be used to style elements with range limitations

```
1 <main>
2   <input type="text" placeholder="add your todo" />
3   <input id="d1" name="d1" type="number" min="1" max="30"
4     placeholder="Days to deadline" required />
5   <label for="deadline1"> </label>
6 </main>
```

```
1 input[type=text] {
2   border: 0px;
3   width: 150px;
4 }
5 input[type=number] {
6   width: 100px;
7 }
```

attribute selector

```
1 input:valid + label::after {
2   content: "\2714";
3   color: rgba(0,100,0,0.7);
4 }
5
6 input:invalid + label::after {
7   content: " (invalid)";
8   color: rgba(255,0,0,0.7);
9 }
```

unicode

rgb & alpha

adjacent selector

Popular pseudo-classes

`:in-range` `:out-of-range`

can be used to style elements with range limitations

add your todo

Days to deadline

(invalid deadline)

Create a lecture

123

(invalid deadline)

Create a lecture

123fdsfds

(invalid deadline)

Create a lecture

12

✓

browser check

Pseudo-elements

Pseudo-element: creates an abstraction about the document tree beyond those specified by the document language; it provides access to an element's sub-part.

:: notation, but ... one-colon notation also acceptable for older pseudo-elements

Pseudo-elements

`::first-letter`

`::first-line`

Canonical example:

enlarge the first letter/line of a paragraph

```
1 p::first-line {
2     color:gray;
3     font-size:125%;
4 }
5
6 p::first-letter {
7     font-size:200%;
8 }
```

```
1 <p>
2     To be, or not to be, that
3     is the question—
4 </p>
5 <p>
6     Whether 'tis Nobler in the
7     mind to suffer
8     The Slings and Arrows of
9     outrageous Fortune,...
10 </p>
```


Pseudo-elements

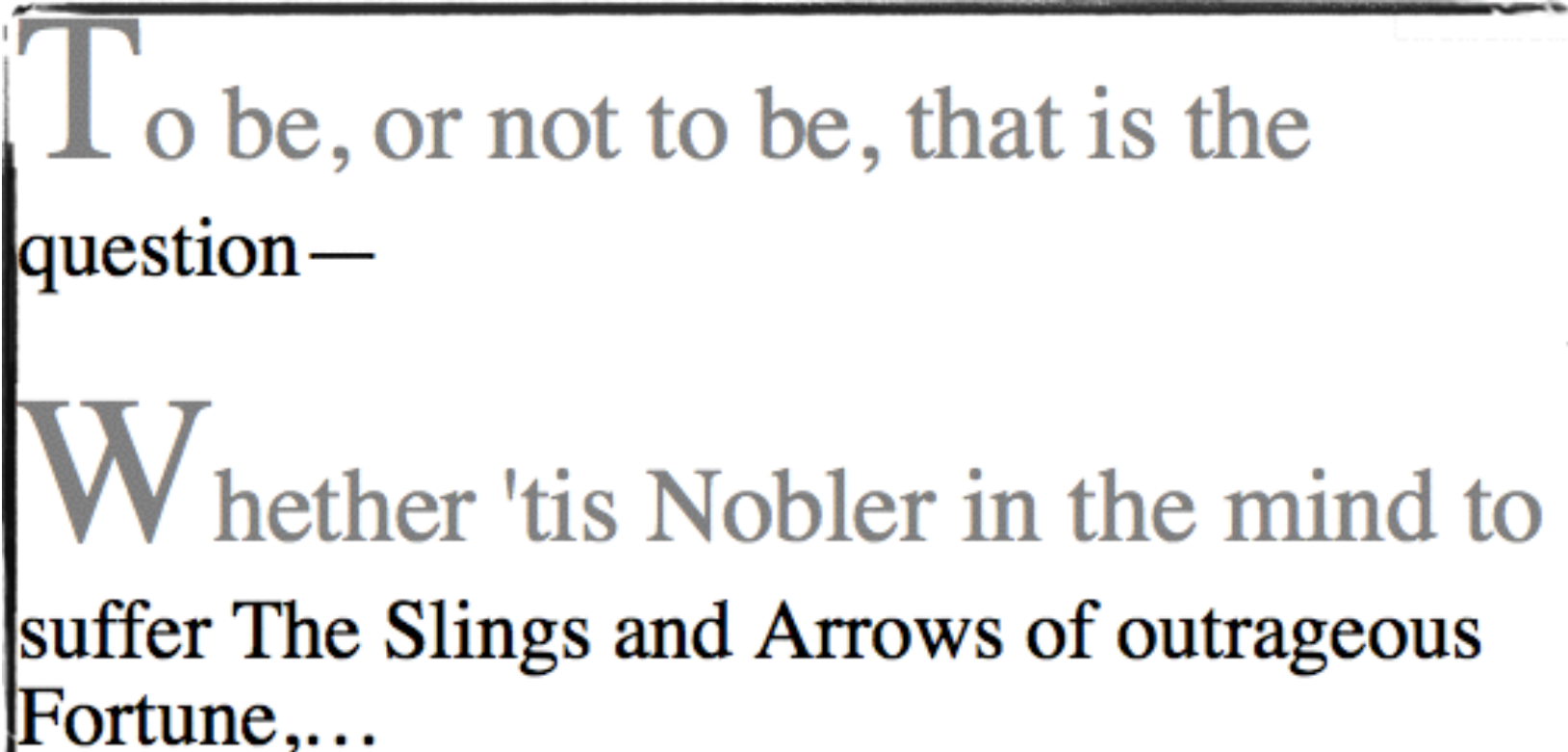
`::first-letter`

`::first-line`

Canonical example:

enlarge the first letter/line of a paragraph

```
1 p::first-line {  
2     color:gray;  
3     font-size:125%;  
4 }  
5  
6 p::first-letter {  
7     font-size:200%;  
8 }
```



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,...

Pseudo-elements

::after used to add (cosmetic) content after an element

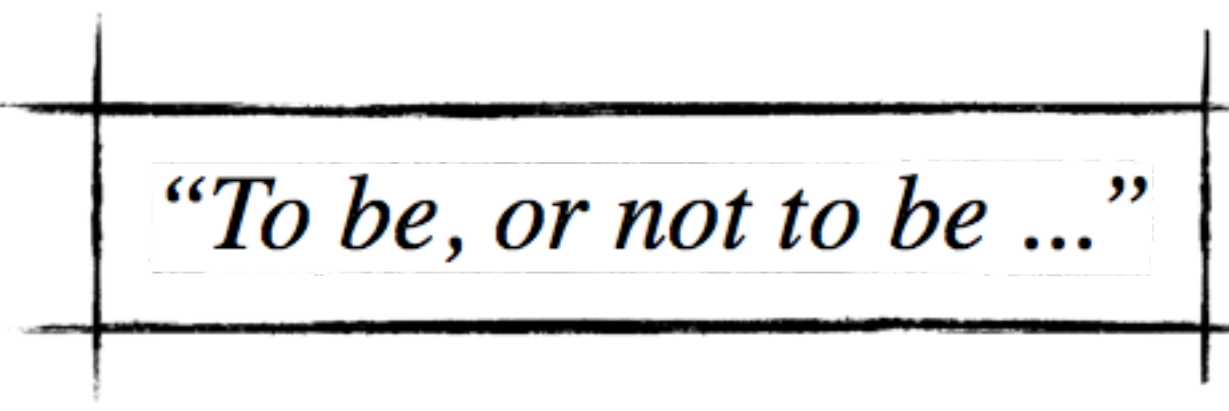
::before used to add (cosmetic) content before an element

```
1 <cite>
2   To be, or not
3   to be ...
4 </cite>
```

```
1 cite::before {
2   content: "\201C";
3 }
4 cite::after {
5   content: "\201D";
6 }
```

Canonical example:

add quotation marks to quotes



“To be, or not to be ...”

Data in CSS

CSS & data (one way)

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

```
1 <main>
2   <h2>Todos</h2>
3   <p id="t1">Walk the dogs</p>
4   <p id="t2">Wash the cups</p>
5   <p id="t3">Clear the pens</p>
6 </main>
```

```
1 p::after {
2   background-color:gold;
3   border: 1px solid;
4   font-size: 70%;
5   padding: 2px;
6   margin-left: 50px;
7 }
8
9 p#t1::after {
10  content: " due 1/1/2015";
11 }
12
13 p#t2::after {
14  content: " due 12/12/2014";
15 }
16
17 p#t3::after {
18  content: " due 1/12/2014";
19 }
```

Todos

Walk the dogs

due 1/1/2015

Wash the cups

due 12/12/2014

Clear the pens

due 1/12/2014

CSS & data (one way)

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

```
1 <main>
2   <h2>Todos</h2>
3   <p id="t1">Walk the dogs</p>
4   <p id="t2">Wash the cups</p>
5   <p id="t3">Clear the pens</p>
6 </main>
```

```
1 p::after {
2   background-color:gold;
3   border: 1px solid;
4   font-size: 70%;
5   padding: 2px;
6   margin-left: 50px;
7 }
8
```

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)

Clear the pens

due 1/12/2014

```
18   content: " due 1/12/2014";
19 }
```


CSS & data-* (the preferred way)

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

Recall: HTML elements can have data-* attributes.

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

```
1 p::after {
2   background-color:gold;
3   border: 1px solid;
4   font-size: 70%;
5   padding: 2px;
6   margin-left: 50px;
7   content: "due " attr(data-due);
8 }
9 p::before {
10  content: url(http://www.abc.de/dot.png);
11 }
```

Todos

- Walk the dogs due 1/1/2015
- Wash the cups due 12/12/2014
- Clear the pens due 1/12/2014

CSS & data-* (the preferred way)

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

Recall: HTML elements can have data-* attributes.

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

```
1 p::after {
2   background-color:gold;
3   border: 1px solid;
4   font-size: 70%;
5   padding: 2px;
6   margin-left: 50px;
7   content: "due " attr(data-due);
8 }
9 p::before {
10  content: url(http://www.abc.de/dot.png);
11 }
```

attr() retrieves the value of an attribute

content attribute can also reference a url

Todos

- Walk the dogs due 1/1/2015
- Wash the cups due 12/12/2014
- Clear the pens due 1/12/2014

CSS & data-* (the preferred way)

Another example: a simple tooltip

```
1 <ul>
2   <li data-name="Cascading Style Sheets">CSS</li>
3   <li data-name="HyperText Markup Language">HTML</li>
4   <li data-name="Hypertext Transfer Protocol">http</li>
5   <li data-name="Hypertext Transfer Protocol Secure">https</li>
6 </ul>
```

```
1 li {
2   cursor: help;
3 }
4 li: hover::after {
5   background-color: rgba(10, 10, 10, 0.7);
6   color: gold;
7   border: 1px dashed;
8   padding: 5px;
9   font-size: 70%;
10  content: attr(data-name);
11  position: relative;
12  bottom: 15px;
13  left: 5px;
14 }
```

we can change the cursor type

- CSS
 - HTML
 - http
 - https
- Hypertext Transfer Protocol

CSS counters

CSS counters can count the number of times a ruleset is called. Counters are set and maintained by CSS.

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

```
1 body {
2   /* initialize counter to 0 */
3   counter-reset: countTodo;
4 }
5 p::before {
6   /* increment at each <p> */
7   counter-increment: countTodo;
8   /* counter written out */
9   content: " Todo " counter(countTodo) ": ";
10 }
```

Todos

Todo 1: Walk the dogs

Todo 2: Wash the cups

Todo 3: Clear the pens

Nested CSS counters

Child elements receive their own counter instance.

Different counter instances are combined via `counters()`.

```
1 <ul>
2   <li>Today's todos
3     <ul>
4       <li>Walk the dogs</li>
5       <li>Wash the cups</li>
6       <li>Clear the pens</li>
7     </ul>
8   </li>
9   <li>Tomorrow's todos
10    <ul>
11      <li>Walk the dogs</li>
12      <li>Wash the dishes</li>
13    </ul>
14  </li>
15 </ul>
```

```
1 ul {
2   counter-reset: cli;
3   list-style-type: none;
4 }
5
6 li::before {
7   counter-increment: cli;
8   content: counters(cli, ".") ". ";
9 }
```

```
1: Today's todos
  1.1: Walk the dogs
  1.2: Wash the cups
  1.3: Clear the pens
2: Tomorrow's todos
  2.1: Walk the dogs
  2.2: Wash the dishes
```

Deciding which CSS
features to use

Can I use `attr()` ?

Is it an established (accepted) part of the CSS specification?

1. W3C CSS specification

- **Candidate Recommendation or Recommendation?**
- **CSS2 or CSS3?**
- **Exhaustive overview of all aspects (by necessity)**

2. Mozilla Developer Network

- **Focuses on the most important aspects of a technology (not exhaustive)**
- **Up-to-date information**
- **Easy to get a quick overview**

Browser-specific prefixes

CSS is under active development, many features are **not stable**, are often used with **browser vendor prefixes**, and, **might change** in the future (as the specification changes).

```
1 main:-webkit-full-screen {
2 } /* Chrome */
3
4 main:-moz-full-screen {
5 } /* Firefox */
6
7 main:-ms-fullscreen {
8 } /* Internet Explorer */
9
10 main:fullscreen {
11 } /* W3C proposal */
```

- **Advantage:** exciting new features can be used early on
- **Disadvantage:** a new browser release might break the implemented CSS

Browser-specific prefixes

CSS is under active development, many features are **not stable**, are often used with **browser vendor prefixes**, and, **might change** in the future (as the specification changes).

Recent move towards disabling experimental features in browsers by default; explicit reset by user required.

But ... **vendor prefixes** will not go away anytime soon (that would break a lot of pages on the Web).

```
8 } /* Internet Explorer */
9
10 main:fullscreen {
11 } /* W3C proposal */
```

browser release might break the implemented CSS

-webkit? Google Chrome is not based on Webkit anymore ...

Will we see a `-chrome-` vendor prefix now?

We've seen how the proliferation of vendor prefixes has caused pain for developers and we don't want to exacerbate this. As of today, Chrome is adopting a policy on vendor prefixes, one that is similar to [Mozilla's recently announced policy](#).

In short: we won't use vendor prefixes for new features. Instead, we'll expose a single setting (in `about:flags`) to enable experimental DOM/CSS features for you to see what's coming, play around, and provide feedback, much as we do today with [the "Experimental WebKit Features"/"Enable experimental Web Platform features" flag](#). Only when we're ready to see these features ship to stable will they be enabled by default in the dev/canary channels.

For legacy vendor-prefixed features, we will continue to use the `-webkit-` prefix because renaming all these prefixes to something else would cause developers unnecessary pain. We've [started looking into](#) real world usage of HTML5 and CSS3 features and hope to use data like this to better inform how we can responsibly deprecate prefixed properties and APIs. As for any non-standard features that we inherited (like `-webkit-box-reflect`), over time we hope to either help standardize or deprecate them on a case-by-case basis.

<http://www.chromium.org/blink/developer-faq>

Element positioning
with `float`, `position`
and `display`

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.

e.g. `<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.

e.g. `` or `<a>`

```
1 <main>
2 <p>
3   This is a paragraph containing <a href="#">a link</a>
4 </p>
5 <p>
6   This is another paragraph
7   <span>
8     with a span and <a href="#">a link in the span</a>
9   </span>
10 </p>
11 </main>
```

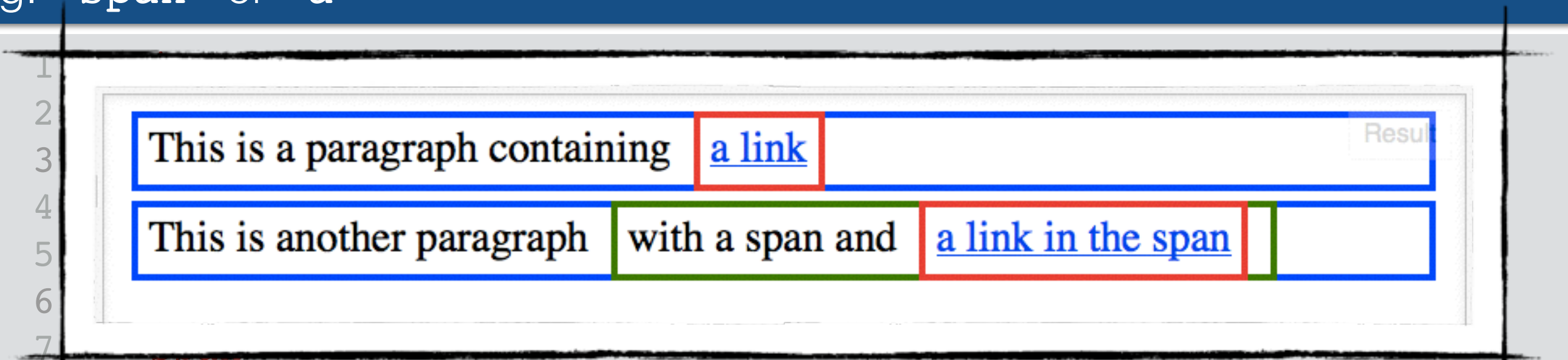

Elements “flow” by default

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

e.g. `<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.

e.g. `` or `<a>`



```
1  
2  
3 This is a paragraph containing a link   
4  
5 This is another paragraph with a span and a link in the span   
6  
7  
8 with a span and <a href="#">a link 1 main {width: auto;}  
9 </span>  
10 </p>  
11 </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.

e.g. `<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.

e.g. `` or `<a>`

```
1  
2  
3 This is a paragraph containing a link  
4  
5 This is another paragraph with a span and  
6 a link in the span  
7  
8 with a span and <a href="#">a link  
9 </span>  
10 </p>  
11 </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.

Result

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.

a link This is a paragraph containing

a link in the span This is another paragraph **with a span and**

```
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

Resur
a link

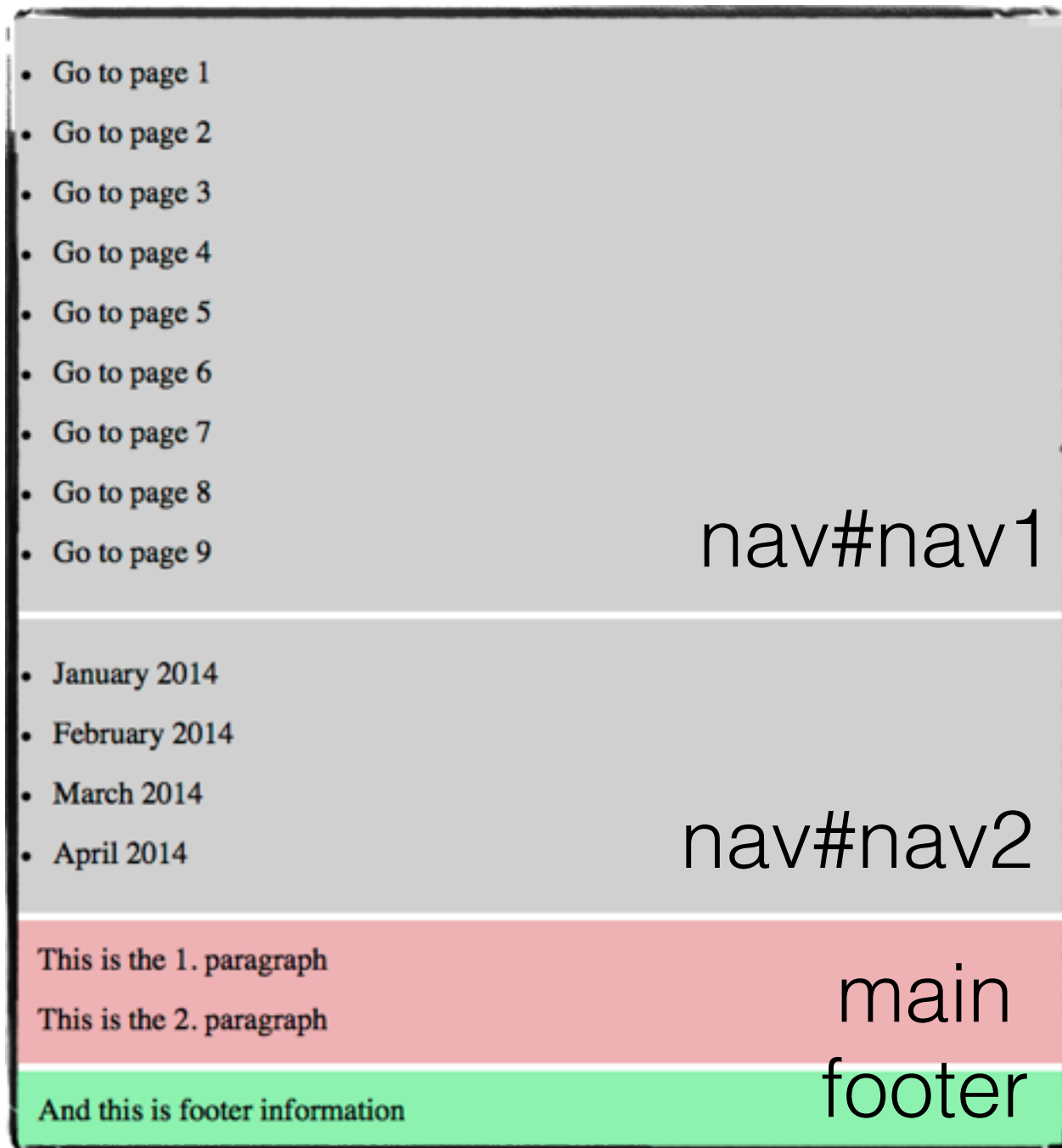
This is another paragraph with a span and

a link in the span

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

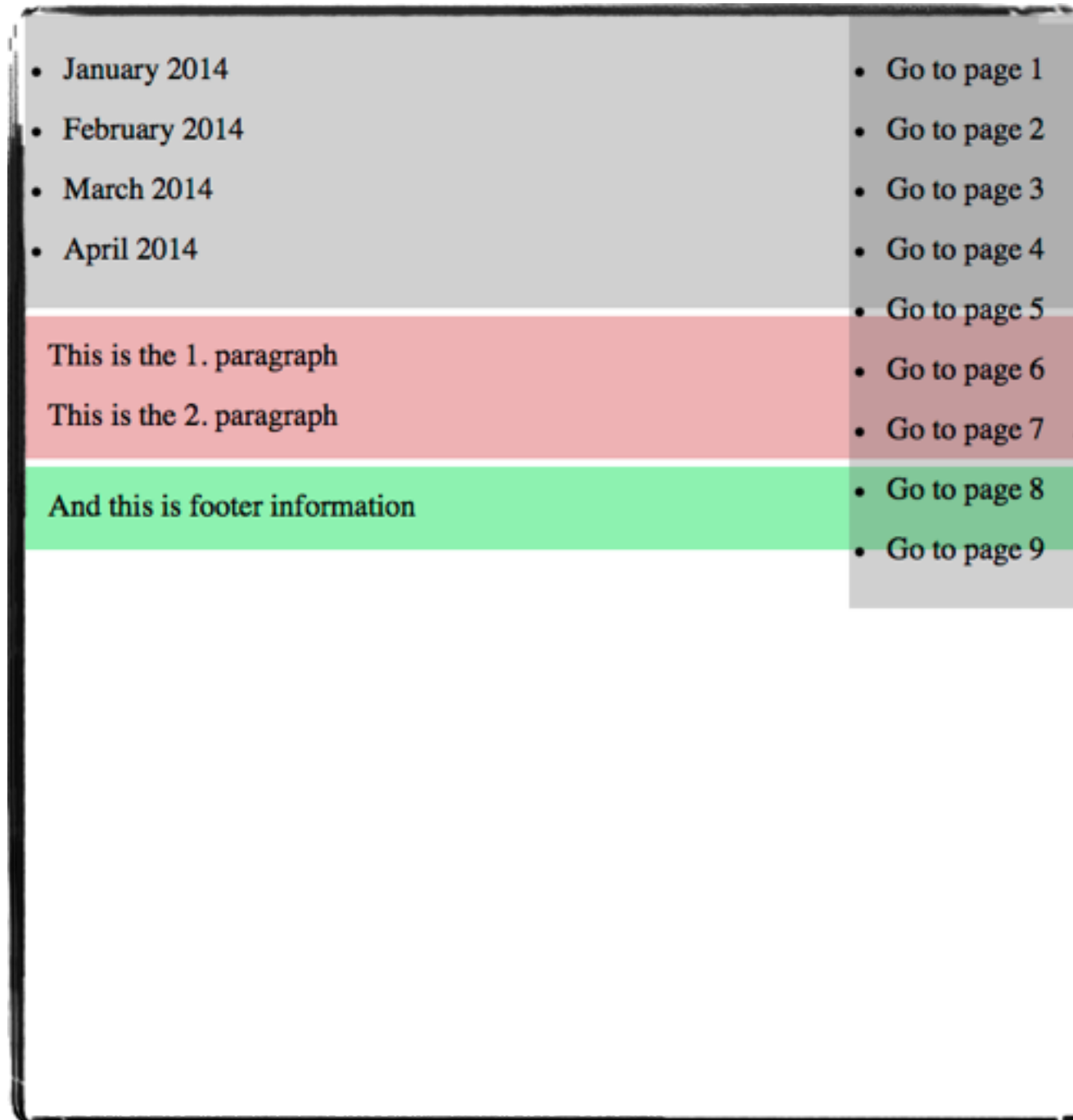
Resetting the flow with clear

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



Resetting the flow with `clear`

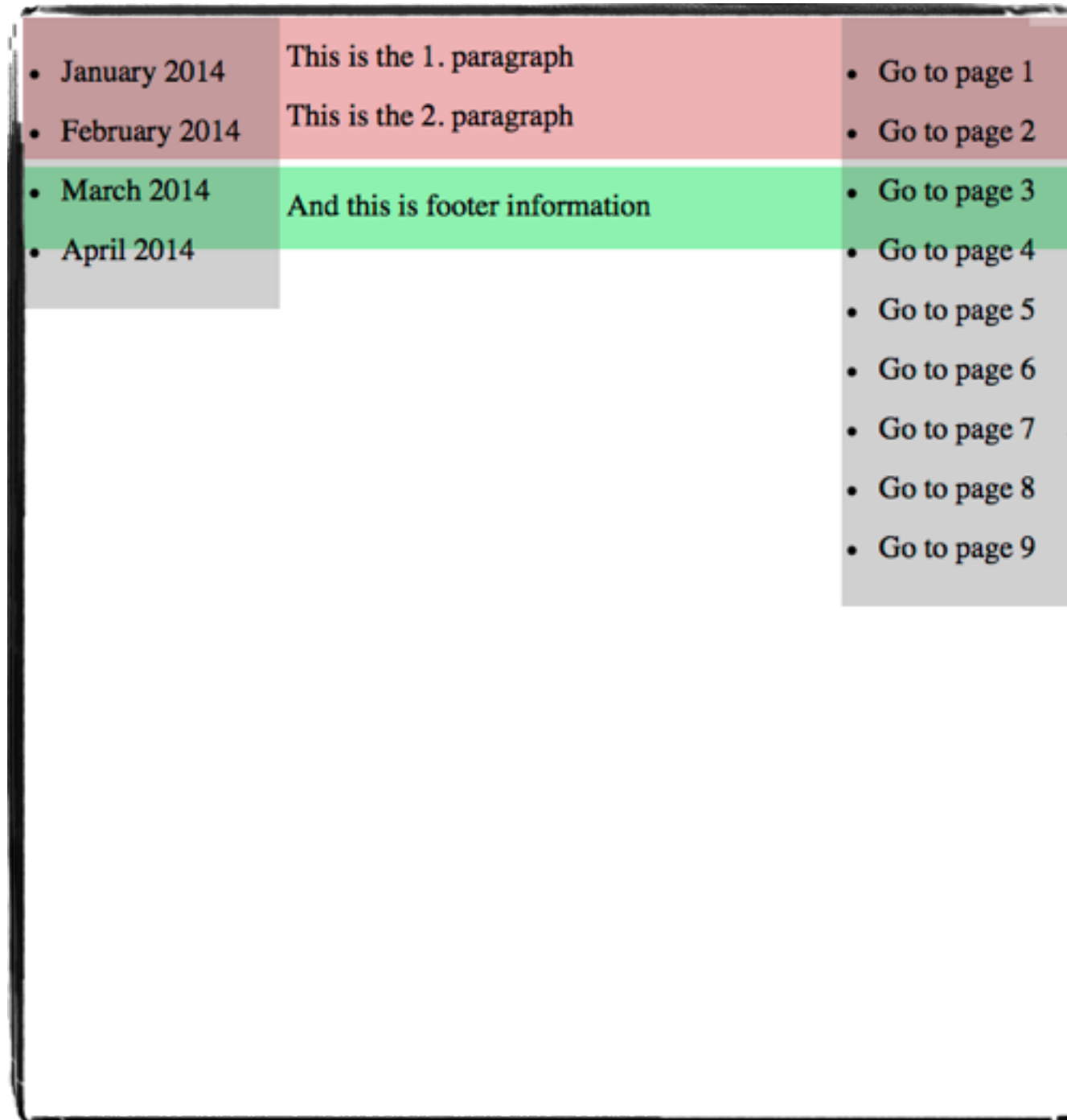
Canonical example: **adding sidebars** to a layout



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

Resetting the flow with `clear`

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

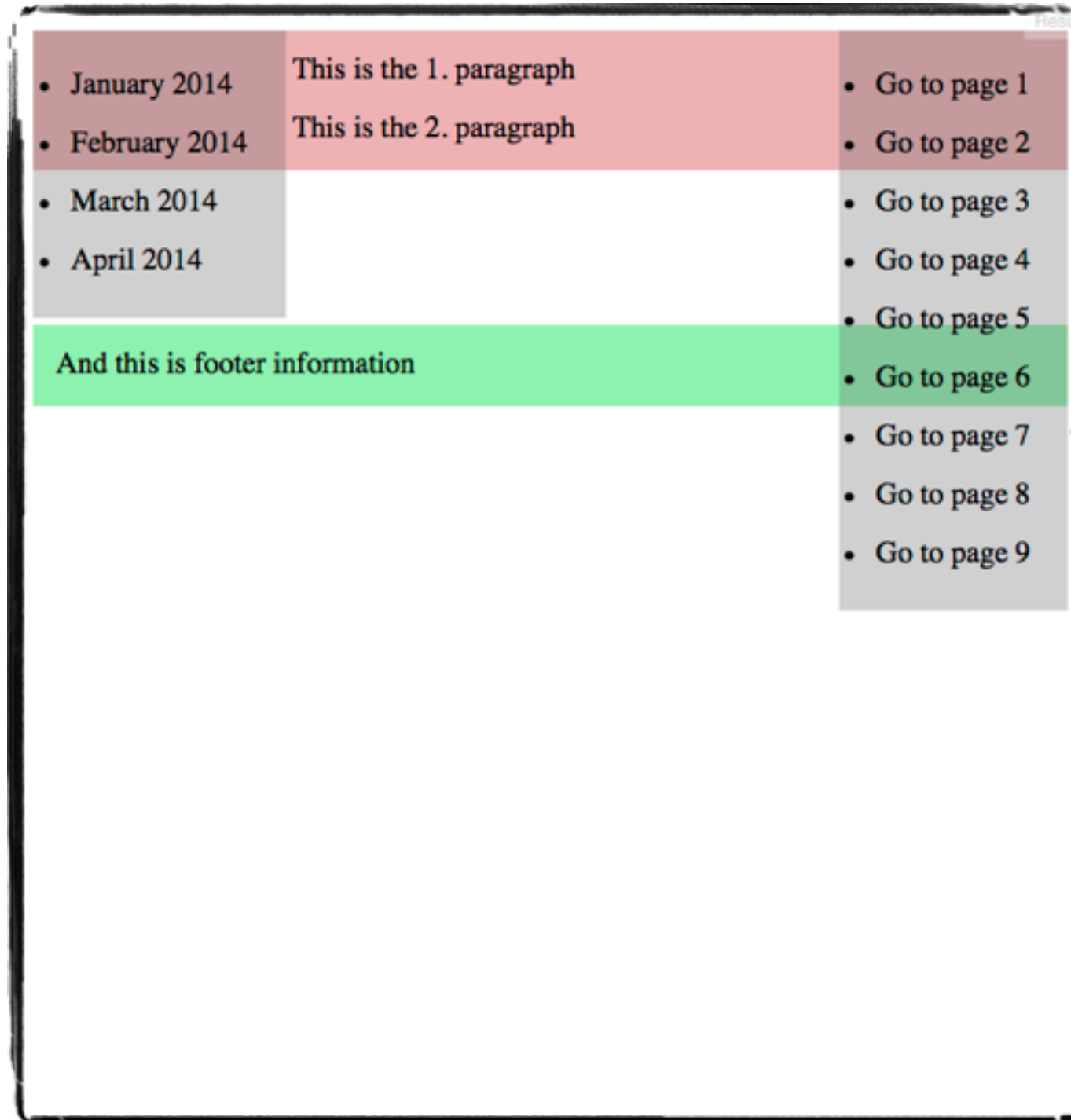


```
1 #nav1 {float: right;}
```

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

Resetting the flow with clear

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



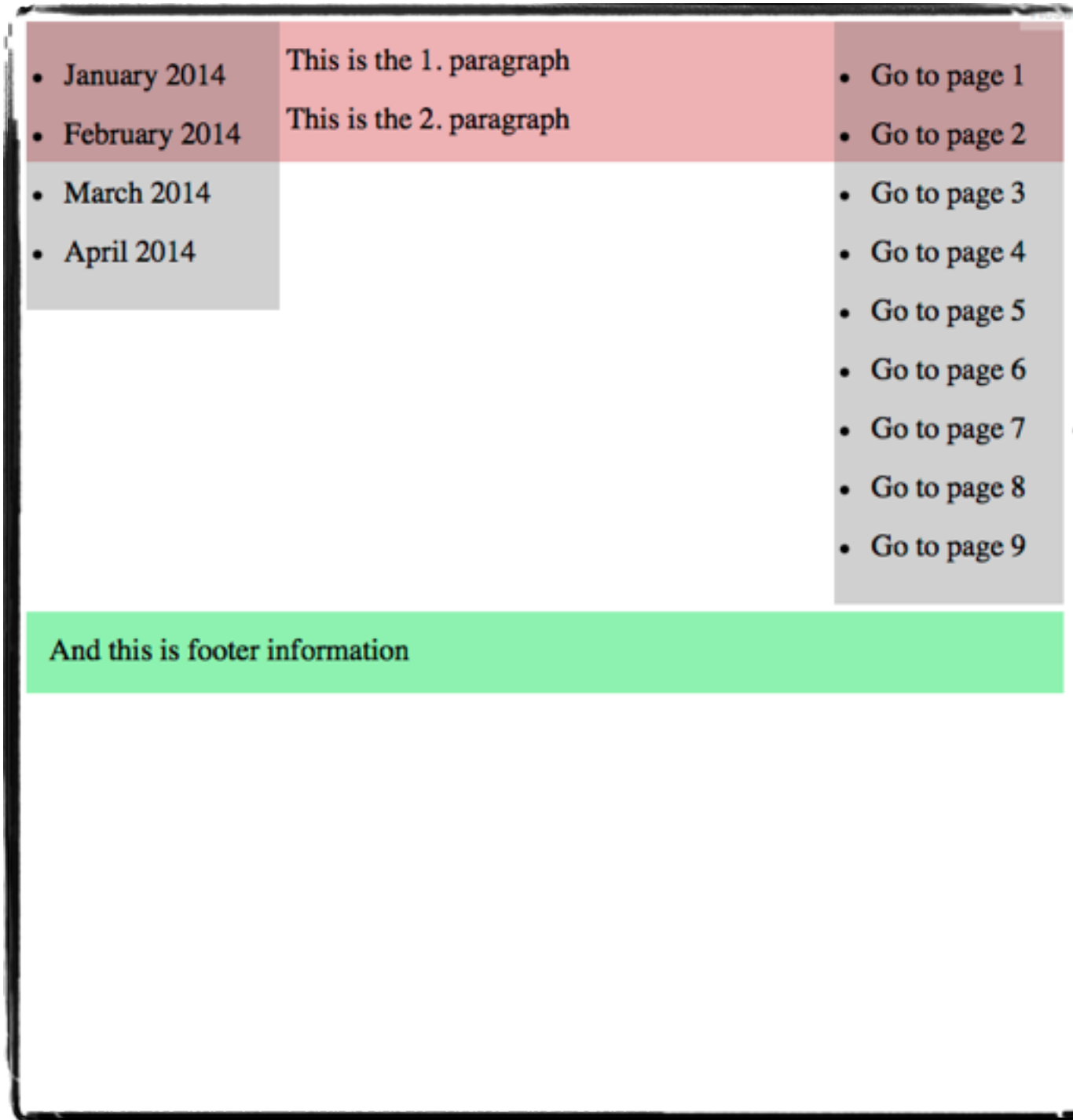
```
1 #nav1 {float: right;}
```

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

```
3 footer{clear: 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;}
```

```
4 footer{clear: right;}
```

```
3 footer{clear: both;}
```

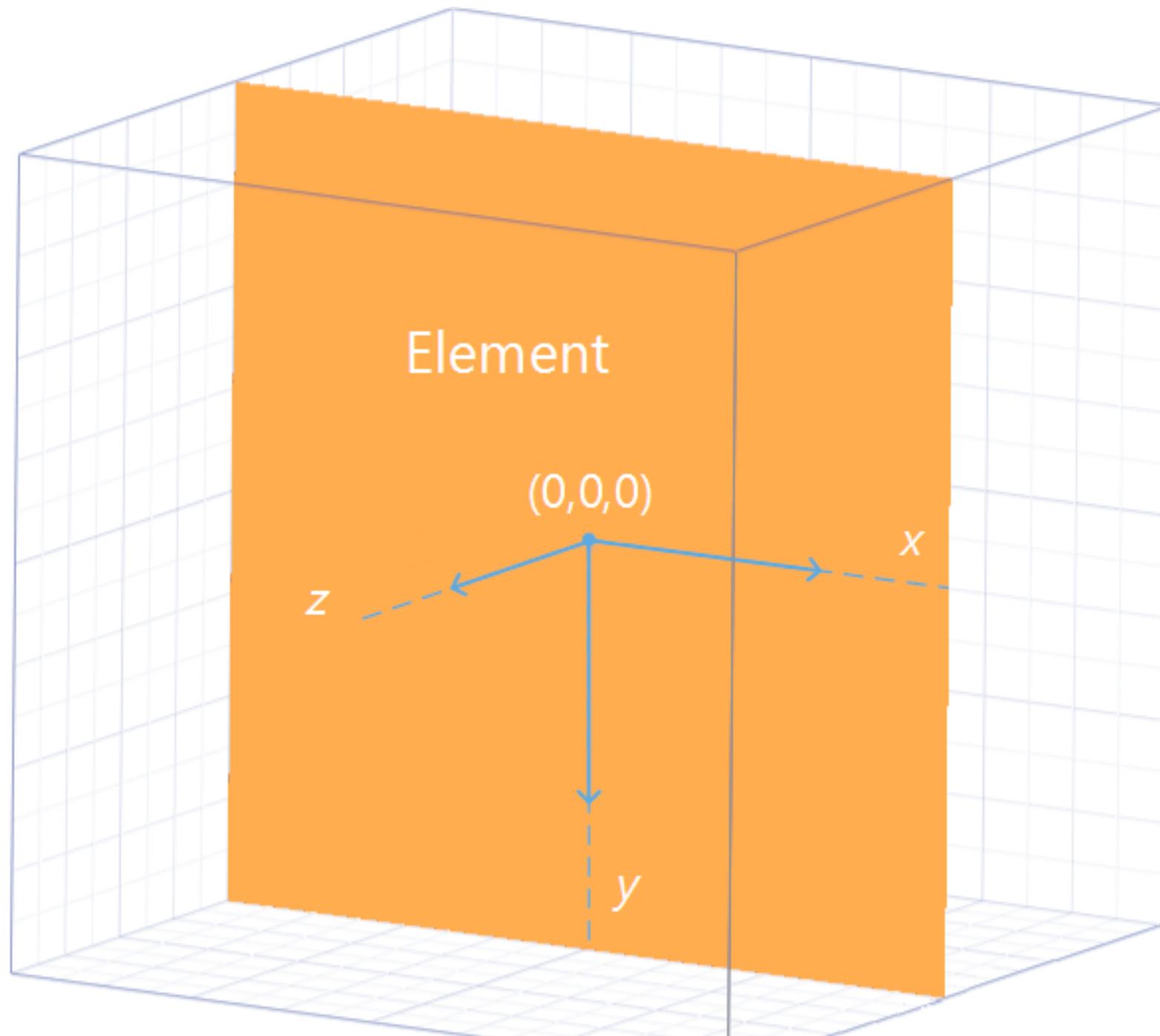
can be used
instead

Fine-grained movement of elements: `position`

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

<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>	the area currently being viewed the viewport
<code>position:sticky</code>	in-between relative and fixed

CSS coordinate system

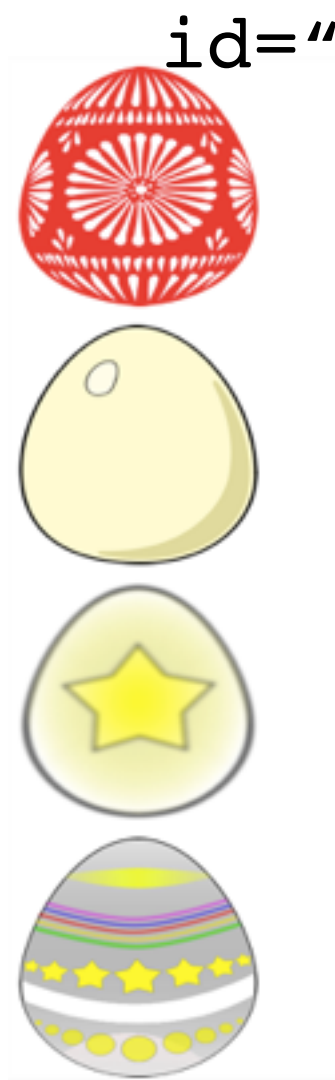


y extends downward. **x** extends to the right.

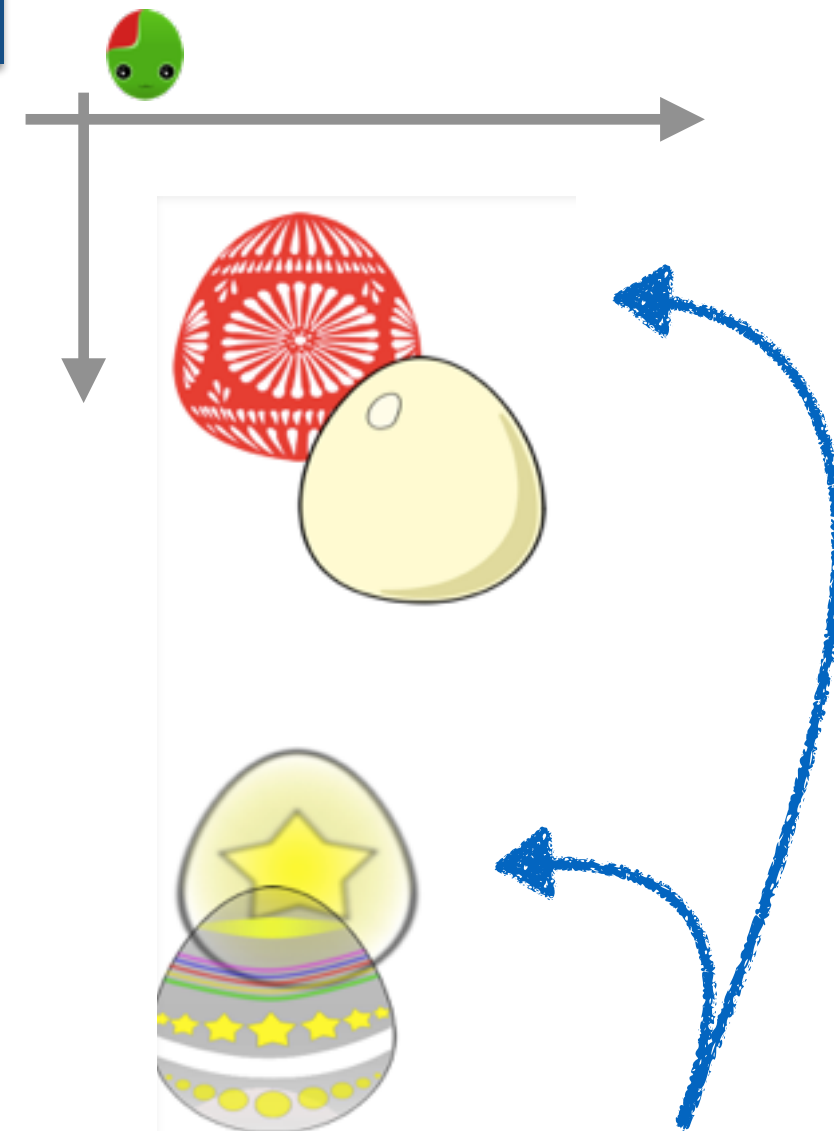
position: relative

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

movement is **relative** to its original position



```
1 #egg2 {  
2   position: relative;  
3   bottom: 20px;  
4   left: 20px;  
5 }  
6  
7 #egg4 {  
8   position: relative;  
9   bottom: 50px;  
10  right: 10px;  
11 }
```



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

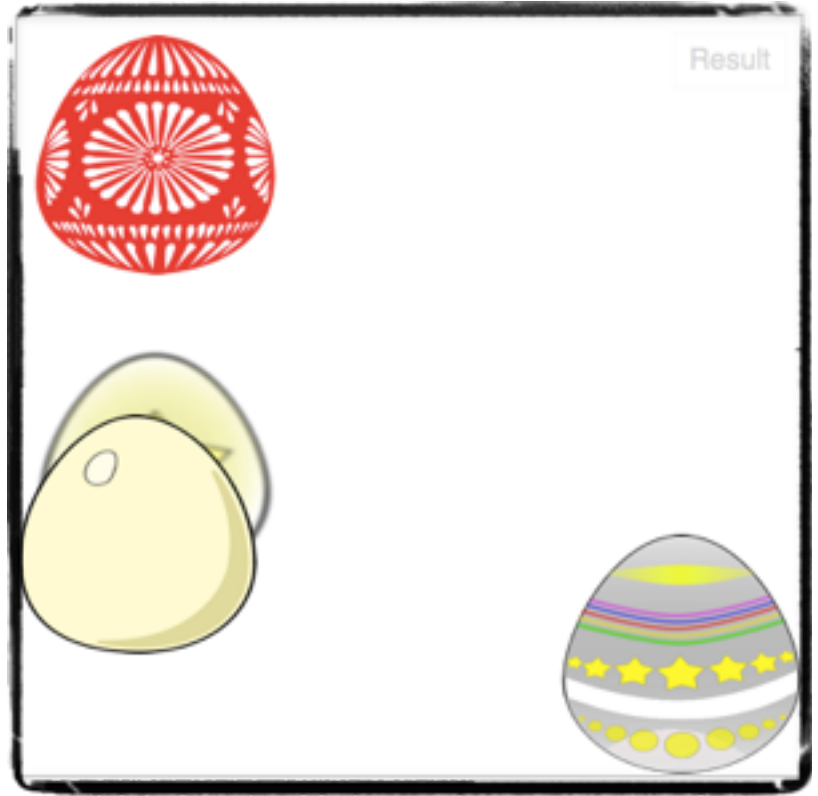
the element is taken out of the normal flow (no space is reserved)

positioning is relative to nearest ancestor or the window

`id="egg1"`



```
1 #egg2 {
2   position: absolute;
3   bottom: 50px;
4   left: 0px;
5 }
6
7 #egg4 {
8   position: absolute;
9   bottom: 0px;
10  right: 0px;
11 }
```



`id="egg4"`

Distance between the element's bottom edge (BE) and that of its containing block.
Distance between the element's left edge and that of its containing block.

Result

window

position: fixed

similar to absolute, but the containing “element” is the viewport

area of the document visible in the browser

elements with position: fixed are always visible

The screenshot shows a web browser window with the URL `https://thimble.mozilla.org/user/claudiahauff/64405`. The page title is "CSS pseudo-classes". The browser interface includes a navigation bar with "Help", "Hi, claudiahauff", and "Publish" buttons. Below the browser, a code editor displays the HTML code for the page. The code includes a `<div id="todaysdate">` block containing the text "Ads should always be visible." and a `<div>` block containing a span with "Today's todos" and a date "27/11/2015", followed by a button with the text "ADD TODO".

```
1 <html>
2 <head>
3   <title>My todos</title>
4   <!-- This links to the stylesheet for this page -->
5   <link rel="stylesheet" type="text/css" href="style.css">
6 </head>
7 <body>
8 <main>
9 <div id="todaysdate">
10   Ads should always be visible.
11 </div>
12 <div>
13   <span>Today's todos</span>
14   <span>27/11/2015</span>
15   <button id="addtoday">ADD TODO</button>
```

The preview window on the right shows the rendered page. It features a "Today's todos" section with a date of "27/11/2015" and an "ADD TODO" button. The list of todos includes: "Prepare TH1506 lecture 4", "Correct MSc report", "Buy milk", and "Write email to John". Below this is a "Tomorrow's todos" section with a date of "28/11/2015" and another "ADD TODO" button. The list of todos includes: "Prepare TH1506 lecture 5", "Prepare workshop presentation", "Work on todo list project (issue#12)", and "Make dinner". At the bottom, there is a "Looking ahead" section with an "ADD TODO" button. A yellow sticky note on the right side of the preview window reads "Ads should always be visible.", demonstrating the effect of the `position: fixed` CSS property.

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

most useful to us

This is paragraph one.

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

This is paragraph two.

display

display:inline

display:block

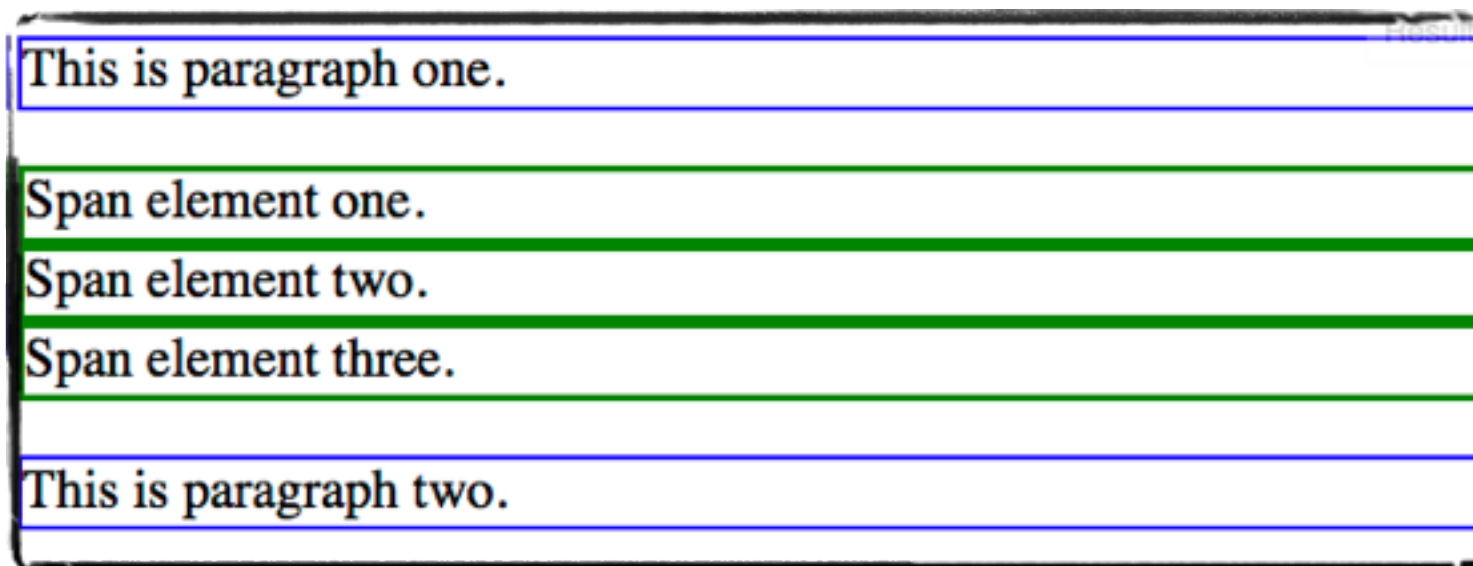
display:none

most useful to us

element rendered with an inline element box

element rendered with a block element box

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



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

display

display:inline

display:block

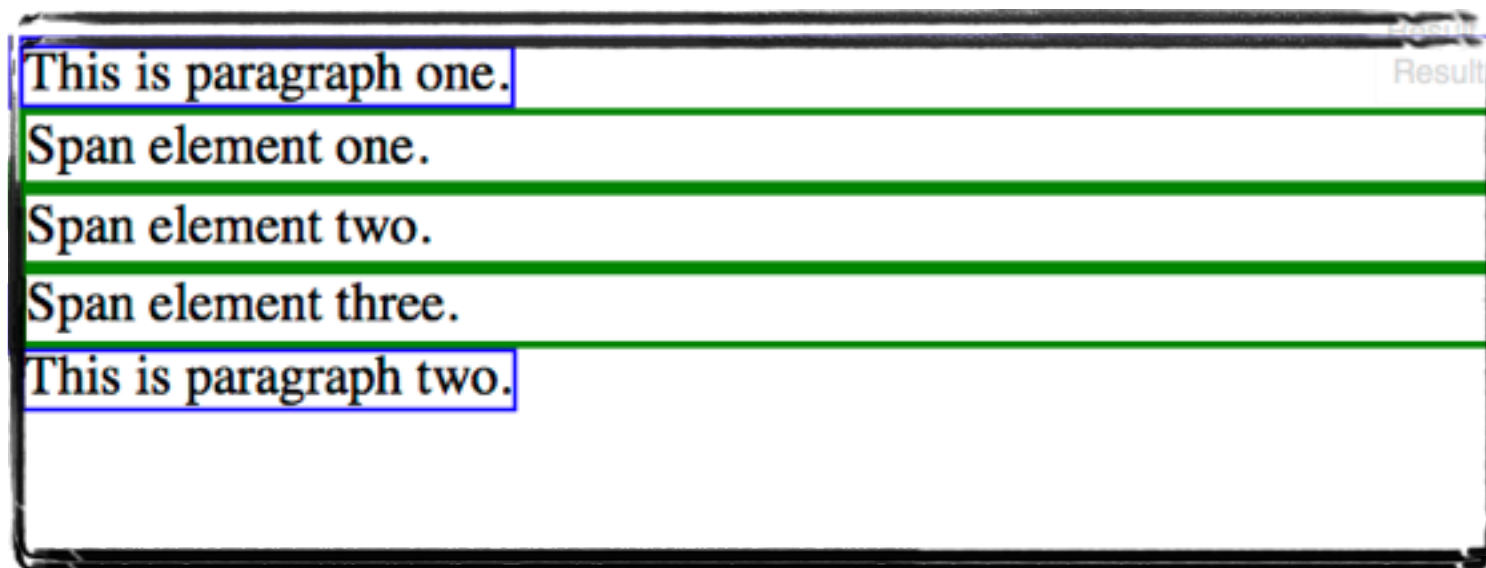
display:none

most useful to us

element rendered with an inline element box

element rendered with a block element box

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



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


display

display:inline

element rendered with an inline element box

display:block

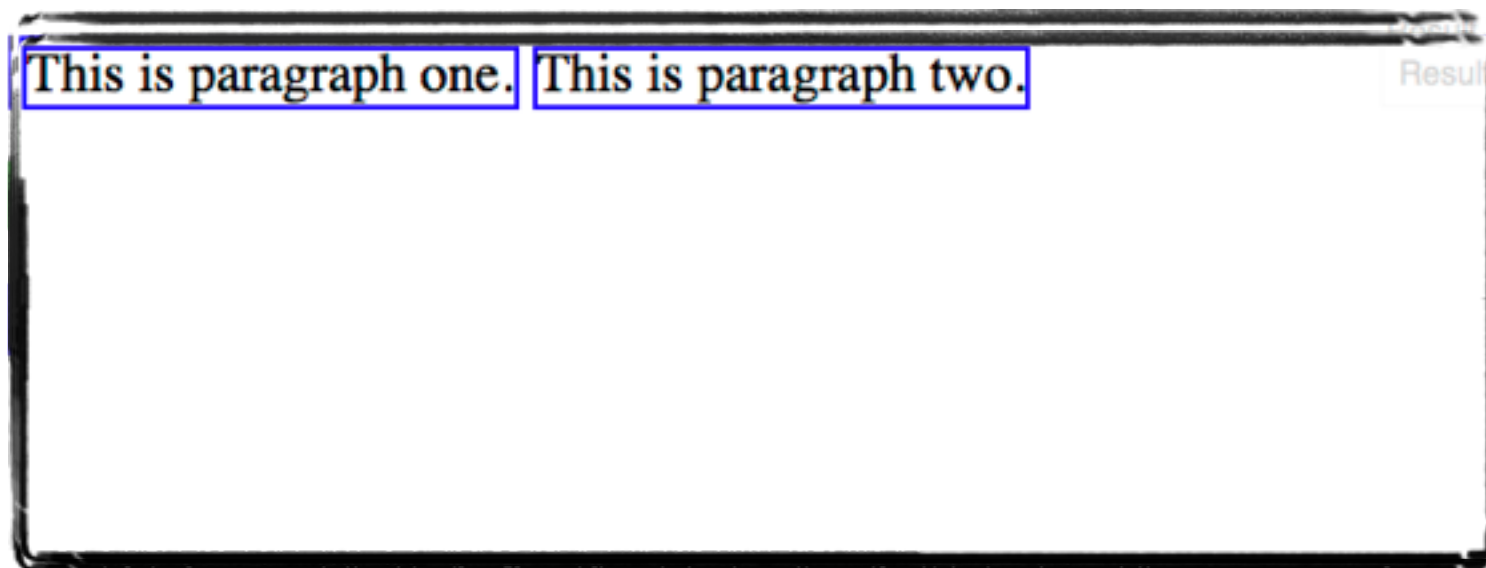
element rendered with a block element box

display:none

element (and its descendants) are hidden

from view; no space is reserved in the layout

most useful to us



```
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, e.g.
 - **Printing** a todo list: ideally only b/w, no color blocks
 - **Viewing** a todo list on a **small screen**: remove non-essential information (footer, etc.)
 - **Viewing** a todo list on a **large screen**: present all available information
 - **Text-to-speech** devices: remove non-essential information (e.g. <http://responsivevoice.org/>)
- CSS media queries enable the use of **device-dependent** (i.e. media-type dependent) stylesheets

HTML5 technology

HTML: write once

CSS: write once per device

Media queries can be complex

Media types: **all, print, screen, speech**

```
1 <link rel="stylesheet"
2   media="screen and (min-width: 800px),
3     (min-width: 3000px)"
4   href="large-device.css">
5 ...
6 <style>
7   @media print {
8     body {
9       color: black !important;
10      width: 100%;
11    }
12  }
13  @media screen and (max-width: 300px) {
14    #sidebar {
15      display: none;
16    }
17  }
18 </style>
```

“,”: logical or

dedicated CSS files

rules for different devices in one file

when printing, use black and white

“and”: logical and

hide the sidebar for small devices

Animations and transitions

CodePen: front-end developer playground.

SCSS: “Sassy CSS”, a CSS preprocessor.

CSS animated Xmas Tree
A PEN BY **dodozhang21** PRO

Fork

Settings

Change View

Log In

Sign Up

HTML (Slim)

```
1 div.xmasTree
```

CSS (SCSS)

```
1 @import "compass/css3";  
2  
3 $gray : #4b4b4b;  
4 $yellow : #f5cd2d;
```

JS

```
1
```



<https://codepen.io/dodozhang21/pen/imlvq>

In general ...

- CSS styles (states) are defined by the user, the rendering engine takes care of the transition between styles
- **Animations** consist of:
 - an animation style (linear, etc.)
 - a number of “keyframes” that act as transition waypoints
- **Transitions** are animations (with a simpler syntax):
 - that consist of exactly 2 states: start and end state

CSS vs. JavaScript animations

- **Easy to use** (standard CSS) — no need to learn JavaScript
- Rendering engines are **optimised** for CSS-based animations
- CSS animations can do much more than animating buttons

CSS animation example (Firefox)

```
1 #p1 {
2   animation-duration: 5s;
3   animation-name: pToRight;
4   top: 5px; left: 5px;
5 }
6
7 @keyframes pToRight {
8   from {
9     top: 5px; left: 5px;
10    background-color: lightgreen;
11  }
12  50% {
13    background-color: red;
14  }
15  to {
16    top: 5px; left: 250px;
17    background-color: lightblue;
18  }
19 }
```

duration of animation (seconds)

animation name (@keyframes)

start state

intermediate state

end state

CSS animation example (-webkit-)

```
1 #p1 {
2     -webkit-animation-duration: 5s;
3     -webkit-animation-name: pToRight;
4     top: 5px; left: 5px;
5 }
6
7 @-webkit-keyframes pToRight {
8     from {
9         top:5px; left:5px;
10        background-color: lightgreen;
11    }
12    50% {
13        background-color: red;
14    }
15    to {
16        top:5px; left:250px;
17        background-color: lightblue;
18    }
19 }
```

to support different browsers, the code needs to be **repeated** for **every browser prefix**

CSS animation control

animation-iteration-count

number of times an animation is executed (default: 1); value either a positive number or **infinite**

animation-direction

by default the animation restarts at the starting keyframe; if set to **alternate** the animation direction change every iteration

animation-delay

number of seconds until the animation starts (default 0s)

CSS transitions

```
1 .box {
2     border-style: solid;
3     border-width: 1px;
4     display: block;
5     width: 100px;
6     height: 100px;
7     background-color: red;
8
9
10 }
11 .box:hover {
12     background-color: green;
13     width: 200px;
14     height: 200px;
15     -webkit-transform: rotate(180deg);
16     transform: rotate(180deg);
17 }
```

start state

declare transition

end state

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

Today we covered

- the basics of CSS positioning
- the CSS box model
- CSS pseudo-classes and pseudo-elements
- CSS media queries
- the basics of CSS animations