



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

A brief history of CSS

- **CSS 1**: a W3C recommendation in **1996**
 - Support for fonts, colors, alignment, margins, ids and classes

A brief history of CSS

- **CSS 1:** a W3C recommendation in **1996**
 - Support for fonts, colors, alignment, margins, ids and classes
- **CSS 2:** a W3C recommendation in **1998**
 - Support added for media queries, element positioning

A brief history of CSS

- **CSS 1:** a W3C recommendation in **1996**
 - Support for fonts, colors, 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

A brief history of CSS

- **CSS 1**: a W3C recommendation in **1996**
 - Support for fonts, colors, 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>

A brief history of CSS

- **CSS 1**: a W3C recommendation in **1996**
 - Support for fonts, colors, 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

CSS Snapshot 2010

CSS Snapshot 2007

CSS Color Level 3

CSS Namespaces

Selectors Level 3

CSS Level 2 Revision 1

CSS Level 1

CSS Print Profile

Media Queries

CSS Style Attributes

Current Upcoming

NOTE

NOTE

REC

REC

REC

REC

REC

REC

REC

NOTE

REC

REC

REC

REC

Abbreviation Full name

WD Working Draft

LC Last Call

CR Candidate Recommendation

PR Proposed Recommendation

REC Recommendation

CSS 3+: a tale of many modules

Testing	Current	Upcoming
CSS Backgrounds and Borders Level 3	CR	PR
CSS Conditional Rules Level 3	CR	CR
CSS Image Values and Replaced Content Level 3	CR	PR
CSS Marquee	CR	PR
CSS Multi-column Layout	CR	CR
CSS Speech	CR	PR
CSS Values and Units Level 3	CR	PR
CSS Flexible Box Layout	LC	CR
CSS Text Decoration Module Level 3	CR	PR
CSS Cascading and Inheritance Level 3	CR	PR
CSS Fonts Level 3	CR	PR
CSS Writing Modes Level 3	CR	PR
CSS Shapes	CR	PR

Abbreviation	Full name
WD	Working Draft
LC	Last Call
CR	Candidate Recommendation
PR	Proposed Recommendation
REC	Recommendation

CSS 3+: a tale of many modules

Refining

CSS Animations

CSS Counter Styles Level 3

CSS Text Level 3

CSS Fragmentation Level 3

CSS Transforms

CSS Transitions

Cascading Variables

Compositing and Blending

CSS Syntax Level 3

Current Upcoming

WD	WD
LC	CR
LC	CR
WD	WD
WD	WD
WD	LC
WD	LC
LC	CR
CR	PR

Abbreviation Full name

WD	Working Draft
LC	Last Call
CR	Candidate Recommendation
PR	Proposed Recommendation
REC	Recommendation

CSS 3

- Impossible to write 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: #ffee22;  
3   width: 800px;  
4   margin: auto;  
5 }  
6 h1 {  
7   color: maroon;  
8 }  
9 p li {  
10  color: gray;  
11  border: 1px solid gray;  
12 }  
13 p.last {  
14  color: green;  
15 }
```

selector

property

value

Chapter 3 of the course book

CSS describes how elements in the DOM should be rendered.

```
1 body {  
2   background-color: #ffee22;  
3   width: 800px;  
4   margin: auto;  
5 }  
6 h1 {  
7   color: maroon;  
8 }  
9 p li {  
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

Chapter 3 of the course book

CSS describes how elements in the DOM should be rendered.

```
1 body {  
2   background-color: #ffee22;  
3   width: 800px;  
4   margin: auto;  
5 }  
6 h1 {  
7   color: maroon;  
8 }  
9 p li {  
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

Chapter 3 of the course book

CSS describes how elements in the DOM should be rendered.

```
1 body {  
2   background-color: #ffee22;  
3   width: 800px;  
4   margin: auto;  
5 }  
6 h1 {  
7   color: maroon;  
8 }  
9 p li {  
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 trump earlier ones** (if they are on the same level)

Chapter 3 of the course book

CSS describes how elements in the DOM should be rendered.

```
1 body {  
2   background-color: #ffee22;  
3   width: 800px;  
4   margin: auto;  
5 }  
6 h1 {  
7   color: maroon;  
8 }  
9 p li {  
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 trump earlier ones** (if they are on the same level)
- **!important** overrides all other declarations

Pseudo-elements and pseudo-classes

A detour: the rendering engine

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

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.

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 version of Google Chrome
Blink	Google Chrome (new versions), Opera

Pseudo-class

Pseudo-class

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

Pseudo-class

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

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

Pseudo-class

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

Pseudo-classes allow styling according to **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 Xth child element of its parent

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

Popular pseudo-classes

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

:nth-of-type(x) any element that is the Xth 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>
```

Popular pseudo-classes

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

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

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

2. child

```
1 <main>
2   <h2>Todos</h2>
3   parent<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 Xth child element of its parent

:nth-of-type(x) any element that is the Xth 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>
```

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

:nth-child(x)

any element that is the Xth child
of its parent

:nth-of-type(x)

any element that is the Xth sibling of its type

Todos

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

```
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>
```

siblings

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

What is the result of applying the CSS?

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

Todos

A

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

Todos

B

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

Todos

C

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

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)

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 }
```

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 }
```



Popular pseudo-classes

- :enabled** an element that can be clicked/selected
- :disabled** an element that cannot be clicked/selected

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 }
```

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 }
```

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 }
```

Todos

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

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 }
```

Todos
Today's todos
Tomorrow's todos
Saturyday's todos
Sunday's todos

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

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 }
```

Todos
Today's todos
Tomorrow's todos
Saturyday's todos
Sunday's todos

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

whitespace in selectors implies the universal selector: *

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 }
```

Todos
Today's todos
Tomorrow's todos
Saturyday's todos
Sunday's todos

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

whitespace in selectors implies the universal selector: *

Which lines appear in orange?

```
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 }
```

Which lines appear in orange?

```
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>
```

Todos

Today's todos

Tomorrow's todos

Saturday's todos

Sunday's todos

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

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="dl" name="dl" 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 }
```

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="dl" name="dl" 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 }
```



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="dl" name="dl" 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



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="dl" name="dl" 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 }
```

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="dl" name="dl" 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 }
```

adjacent 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="dl" name="dl" 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

adjacent 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 }
```

rgb & alpha

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="dl" name="dl" 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

adjacent selector

```
1 input:valid + label::after {
2   content: "\2714";
3   color: rgba(0,100,0,0.7);
4 }
```

unicode

```
6 input:invalid + label::after {
7   content: "(invalid)";
8   color: rgba(255,0,0,0.7);
9 }
```

rgb & alpha

Popular pseudo-classes

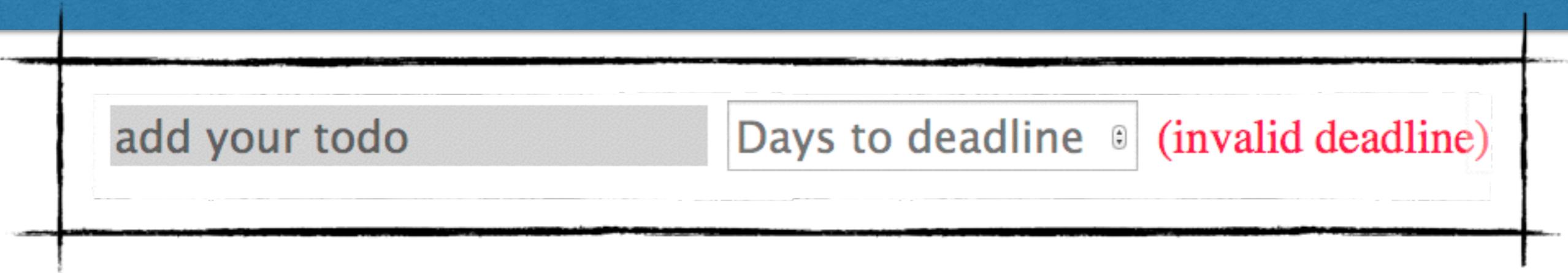
:in-range :out-of-range

can be used to style elements with range limitations

Popular pseudo-classes

:in-range :out-of-range

can be used to style elements with range limitations



Popular pseudo-classes

:in-range :out-of-range

can be used to style elements with range limitations

The image displays two horizontal rows from a user interface, likely a todo application. Both rows consist of a grey input field on the left and a white input field on the right.

Top Row: The grey input field contains the text "add your todo". To its right is a white input field with a light blue border, labeled "Days to deadline" above it. To the right of the input field is the text "(invalid deadline)" in red. A small circular icon with a question mark is positioned between the input field and the error message.

Bottom Row: The grey input field contains the text "Create a lecture". To its right is a white input field with a light blue border, containing the text "123" in black. To the right of the input field is the text "(invalid deadline)" in red. A small circular icon with a question mark is positioned between the input field and the error message.

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)

Popular pseudo-classes

:in-range :out-of-range

can be used to style elements with range limitations

The image displays three horizontal rows from a web application interface. Each row consists of a grey input field on the left and a white input field on the right.

- Row 1:** The grey field contains "add your todo". The white field is labeled "Days to deadline" and contains the value "123". To the right of the input is the text "(invalid deadline)" in red. A small blue icon is visible next to the input field.
- Row 2:** The grey field contains "Create a lecture". The white field is labeled "Days to deadline" and contains the value "123". To the right of the input is the text "(invalid deadline)" in red. A small blue icon is visible next to the input field.
- Row 3:** The grey field contains "Create a lecture". The white field is labeled "Days to deadline" and contains the value "123fdsfds". To the right of the input is the text "(invalid deadline)" in red. A small blue icon is visible next to the input field.

A purple arrow points from the text "(invalid deadline)" in the third row to a purple button at the bottom right labeled "browser check".

Popular pseudo-classes

:in-range :out-of-range

can be used to style elements with range limitations

The image displays a user interface for managing todo items or lectures. It features a header with a search bar and a 'Create a lecture' button. Below this, there are four rows of input fields for 'Days to deadline'. Each row includes a 'Create a lecture' button, an input field, and a status message in red text. A purple arrow points from the status message in the third row to a purple box labeled 'browser check'.

Create a lecture	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-elements

Pseudo-element: creates an abstractions about the document tree beyond those specified by the document language.

Pseudo-elements

Pseudo-element: creates abstractions about the document tree beyond those specified by the document language.

:: 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

Canonical example:
add quotation marks to quotes

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>
```

Canonical example:
add quotation marks to quotes

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>
```

Canonical example:
add quotation marks to quotes

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

Pseudo-elements

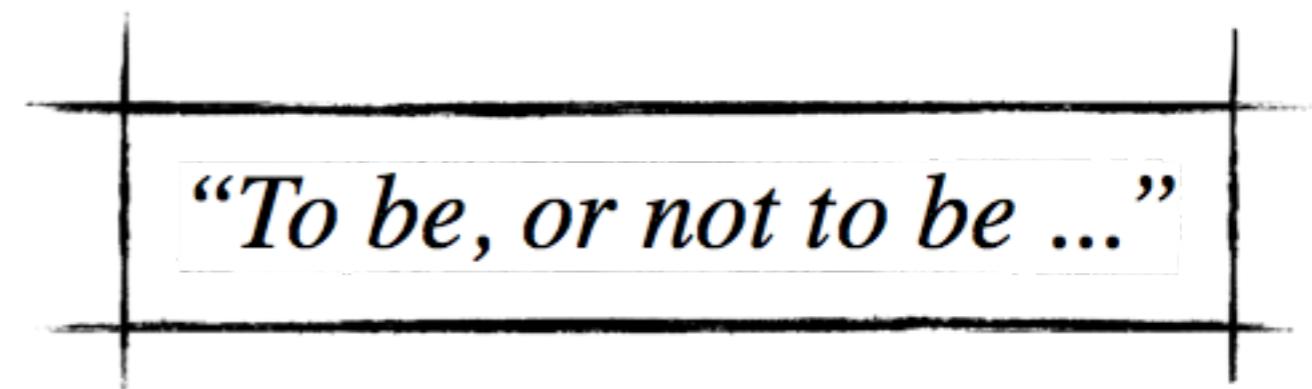
::after used to add (cosmetic) content after an element

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

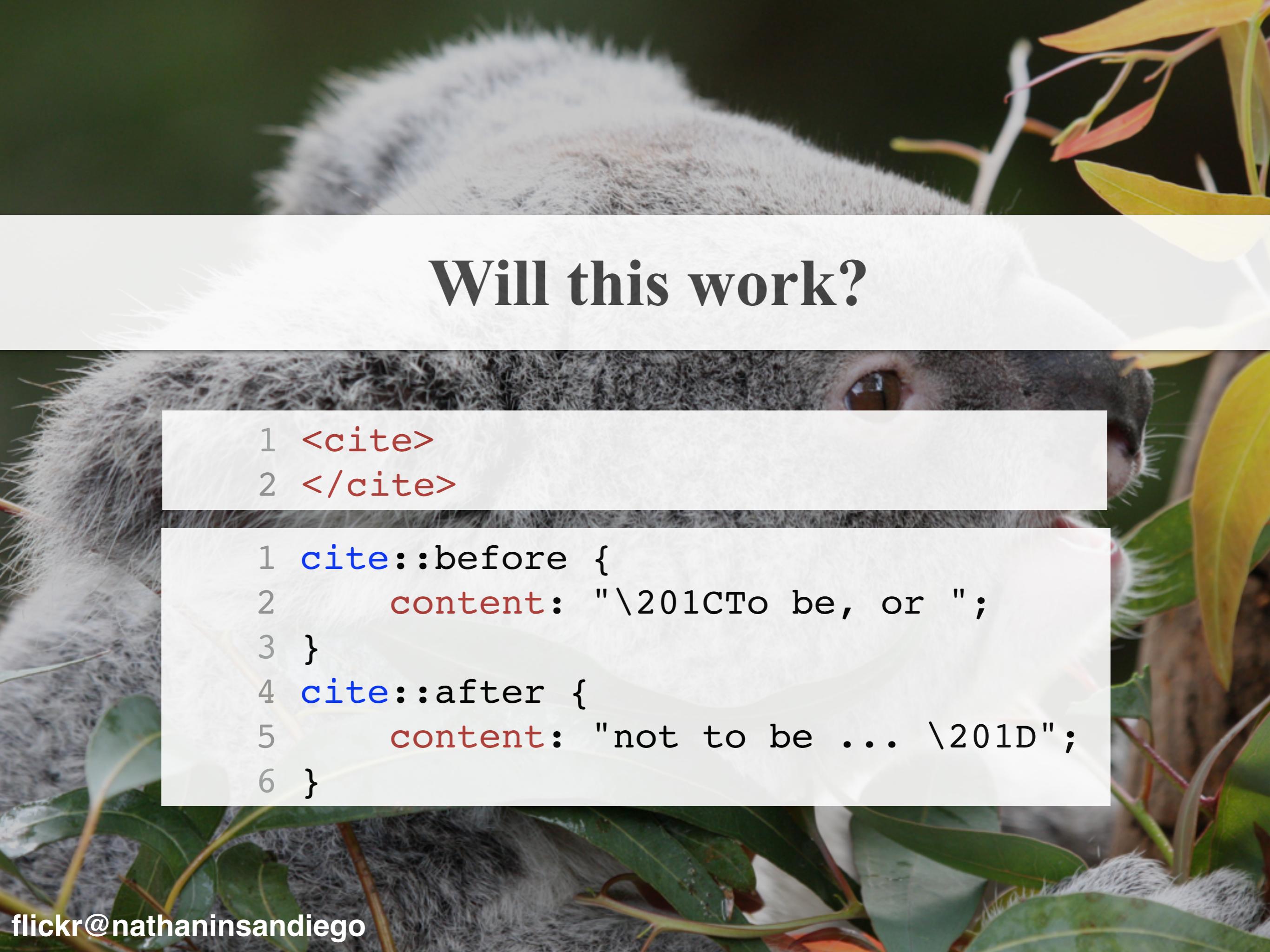
```
1 <cite>  
2 To be, or not  
3 to be ...  
4 </cite>
```

Canonical example:
add quotation marks to quotes

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



“To be, or not to be ...”



Will this work?

```
1 <cite>  
2 </cite>
```

```
1 cite::before {  
2   content: "\201CTo be, or ";  
3 }  
4 cite::after {  
5   content: "not to be ... \201D";  
6 }
```

Data in CSS

CSS & data (one way)

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

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>
```

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>
```

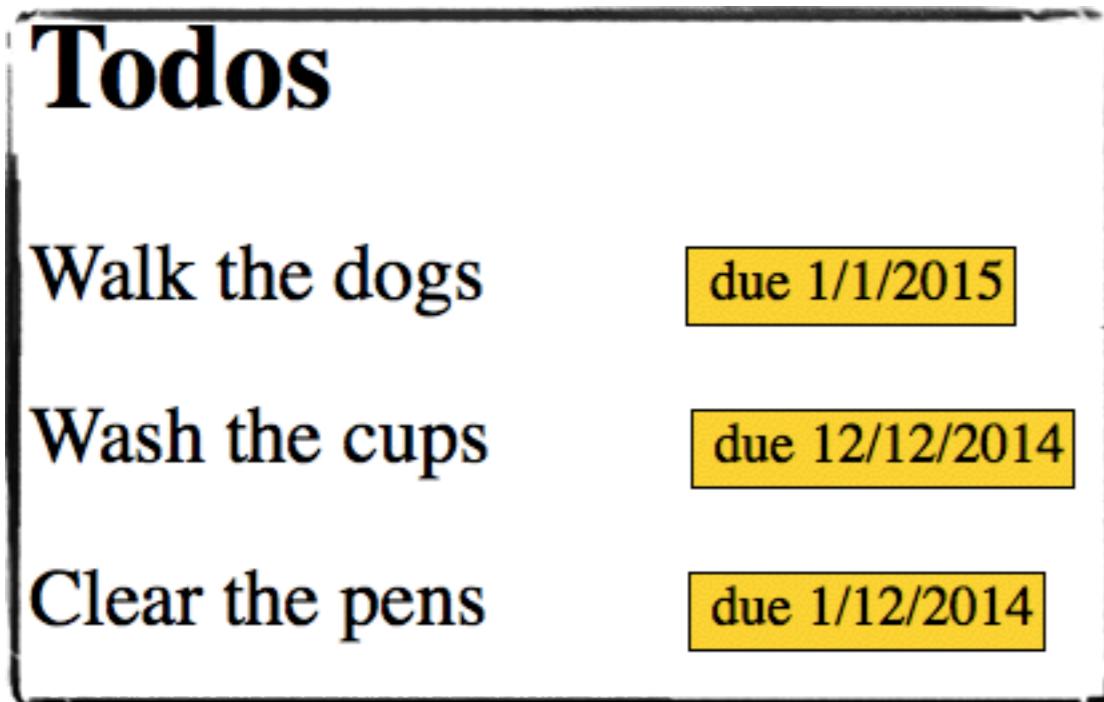
The image shows a user interface for a 'Todos' application. At the top, there is a header with the word 'Todos'. Below the header, there is a list of three items, each consisting of a task name and a due date in a yellow box. The tasks are: 'Walk the dogs' (due 1/1/2015), 'Wash the cups' (due 12/12/2014), and 'Clear the pens' (due 1/12/2014). The entire list is enclosed in a black-bordered box.

Todo	Due Date
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
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 }
```

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**.

```
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>
```

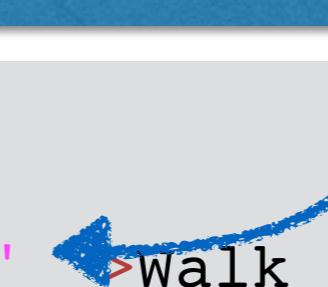
```
1 p::after {
2   background-color:gold;
3   border: 1px solid;
4   font-size: 70%;
5   padding: 2px;
6   margin-left: 50px;
7   content: "due " attr(data-due);
8 }
9 p::before {
10  content: url(http://www.abc.de/dot.png);
11 }
```

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 }
```

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

CSS & data-* (the preferred way)

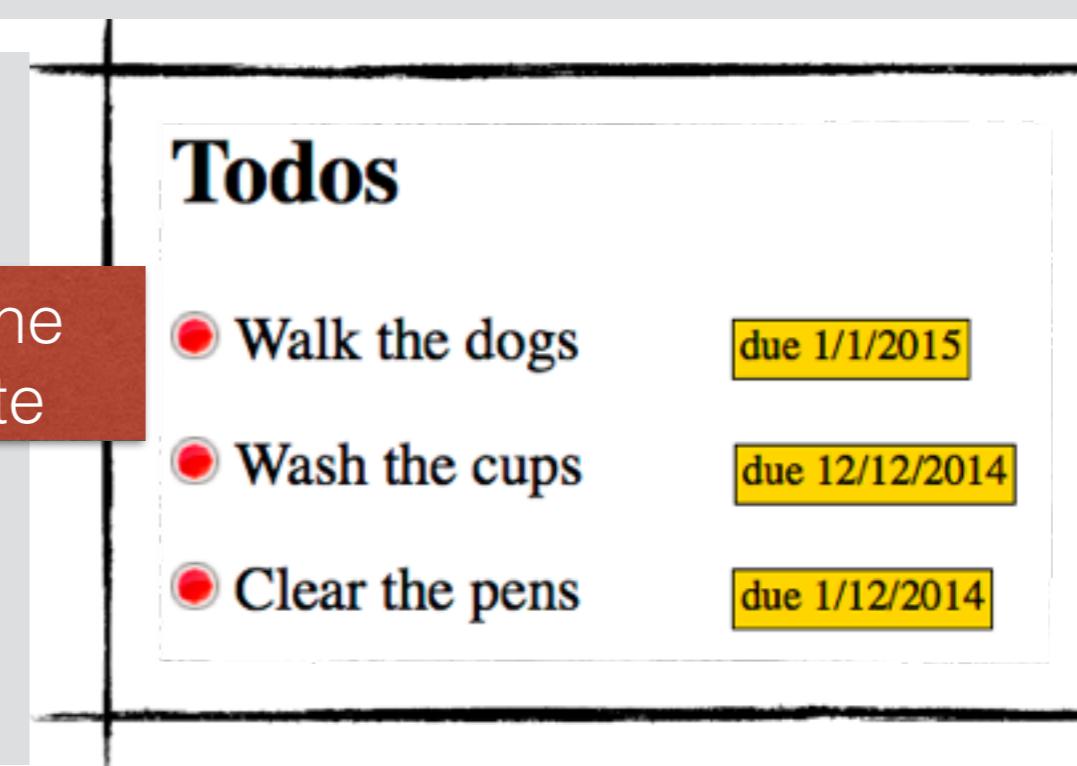
CSS can make use of **data stored in HTML elements**.

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



CSS & data-* (the preferred way)

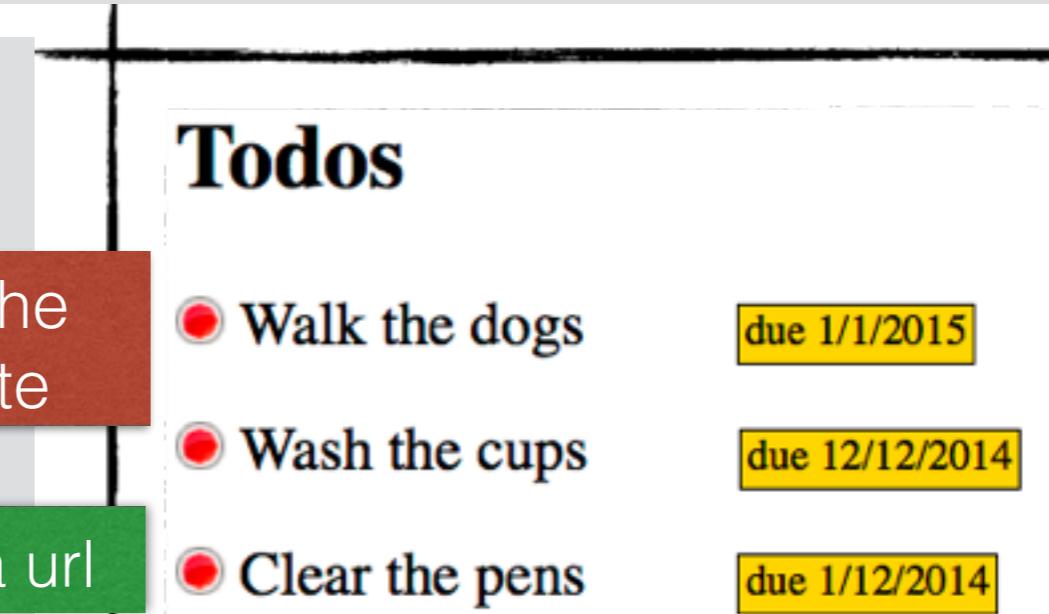
CSS can make use of **data stored in HTML elements**.

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::bef content attribute can also reference a url
10 content: url(http://www.abc.de/dot.png);
11 }
```

attr() retrieves the value of an attribute



CSS & data-* (the preferred way)

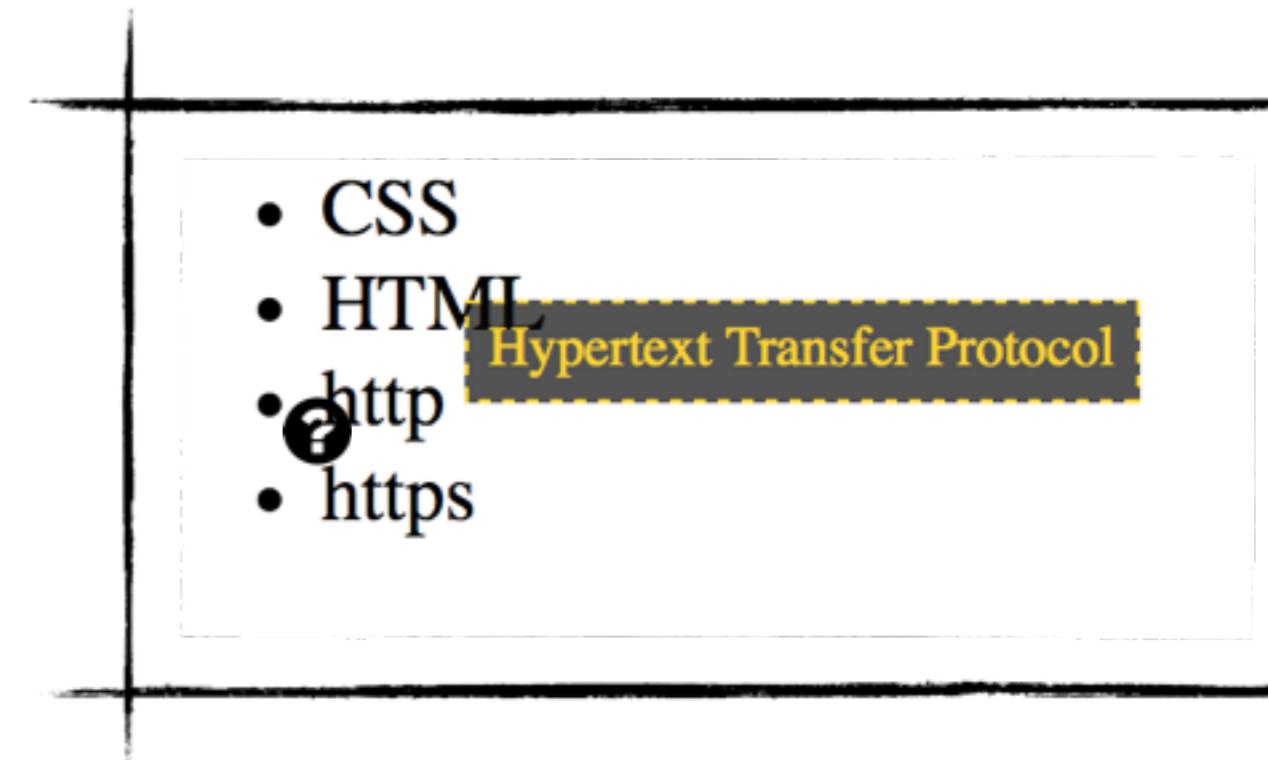
Another example: a simple tooltip

```
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>
```

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>
```



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 }
```

- CSS
- HTML
- http
- https

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 & 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.**

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>
```

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 }
```

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 }
```

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

How are the list elements enumerated?

```
<p>Today's todos</p>
<ul>
  <li>Walk the dogs</li>
  <li>Wash the cups</li>
  <li>Clear the pens</li>
</ul>

<p>Tomorrow's todos</p>
<ul>
  <li>Walk the dogs</li>
  <li>Wash the dishes</li>
</ul>
```

```
body {
  counter-reset: cli;
}

ul {
  list-style: none;
}

li::before {
  counter-increment: cli;
  content: counter(cli)": ";
```

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).

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 */
```

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 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.

Element positioning
with float, position
and display

Elements “flow” by default

```
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>

```
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 <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 or 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 elements are surrounded by line-breaks. They can contain block-level or 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>

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

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

```
1 main {width: auto;}
```

Elements “flow” by default

Block-level elements are surrounded by line-breaks. They can contain block-level or 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>

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

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

```
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 window with two paragraphs. The first paragraph contains the text "a link" followed by "This is a paragraph containing". The word "link" is highlighted with a red border. The second paragraph contains the text "a link in the span" followed by "This is another paragraph with a span and". The phrase "link in the span" is highlighted with a red border, and the word "span" is highlighted with a green border. The browser's status bar at the top right says "Result". A code block at the bottom right shows the CSS rule: "1 a {float: left;}".

```
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.

The diagram illustrates the effect of the `float` CSS property. It shows two paragraphs within a container. The first paragraph contains the text "This is a paragraph containing" followed by a link "a link". The second paragraph contains the text "This is another paragraph with a span and" followed by a link "a link in the span". The link in the first paragraph is positioned at the end of its text content, while the link in the second paragraph is positioned at the end of its span. This demonstrates that floating an element moves it to the leftmost possible position within its containing element's boundaries.

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


Resetting the flow with clear

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

The diagram shows a layout structure with three main horizontal sections. The top section is a sidebar labeled "nav#nav1" containing a list of links from "Go to page 1" to "Go to page 9". The middle section is another sidebar labeled "nav#nav2" containing a list of months from "January 2014" to "April 2014". The bottom section is a main content area labeled "main footer" containing the text "This is the 1. paragraph", "This is the 2. paragraph", and "And this is footer information". The sidebar sections overlap the main content area. The "nav#nav1" section overlaps the top part of "nav#nav2". The "nav#nav2" section overlaps the "main footer" section. The "main footer" section overlaps the bottom part of "nav#nav2". The "nav#nav1" section has a black border and a white background. The "nav#nav2" section has a light gray background. The "main footer" section has a pink background.

- 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

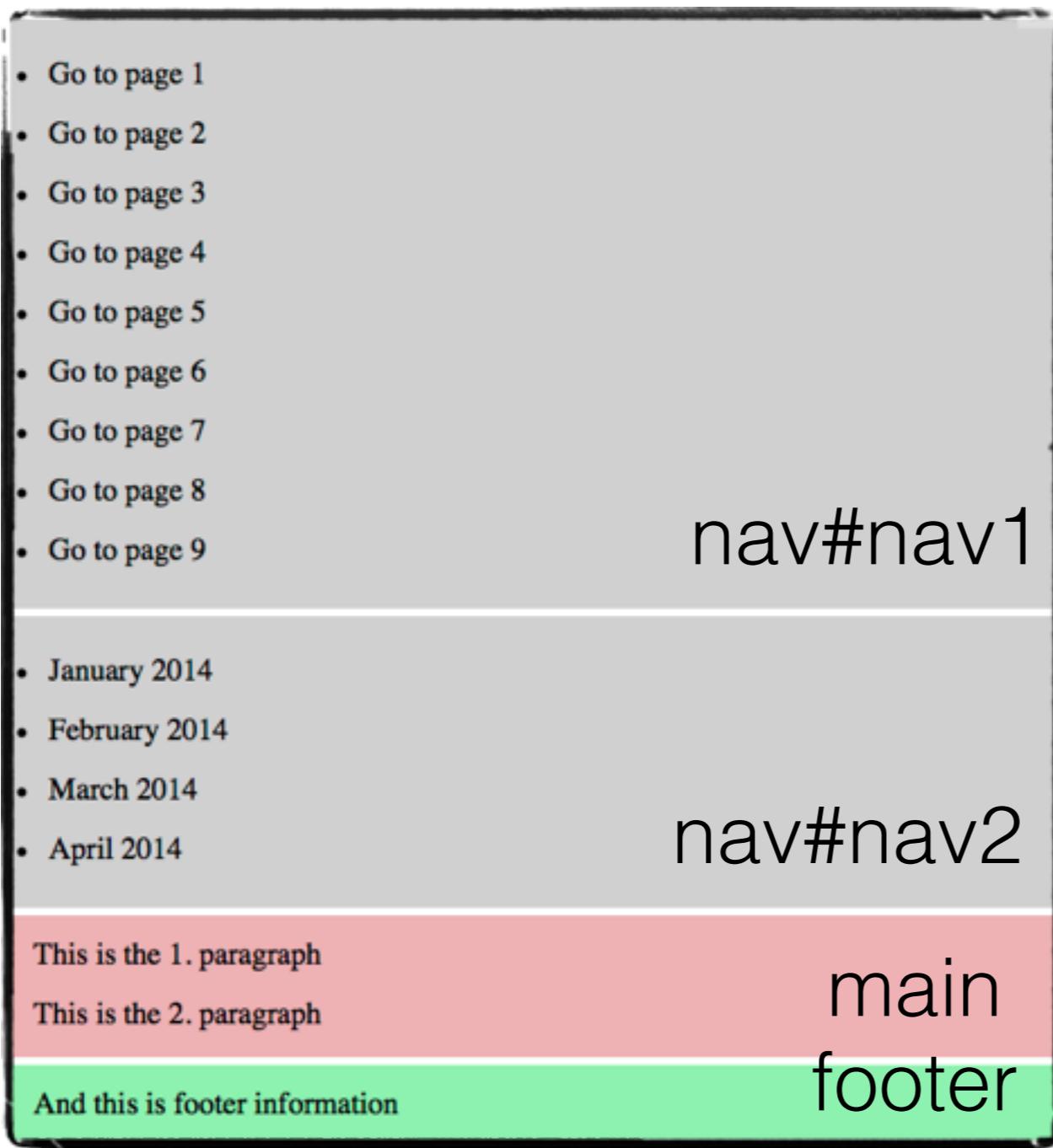
This is the 2. paragraph

And this is footer information

main
footer

Resetting the flow with clear

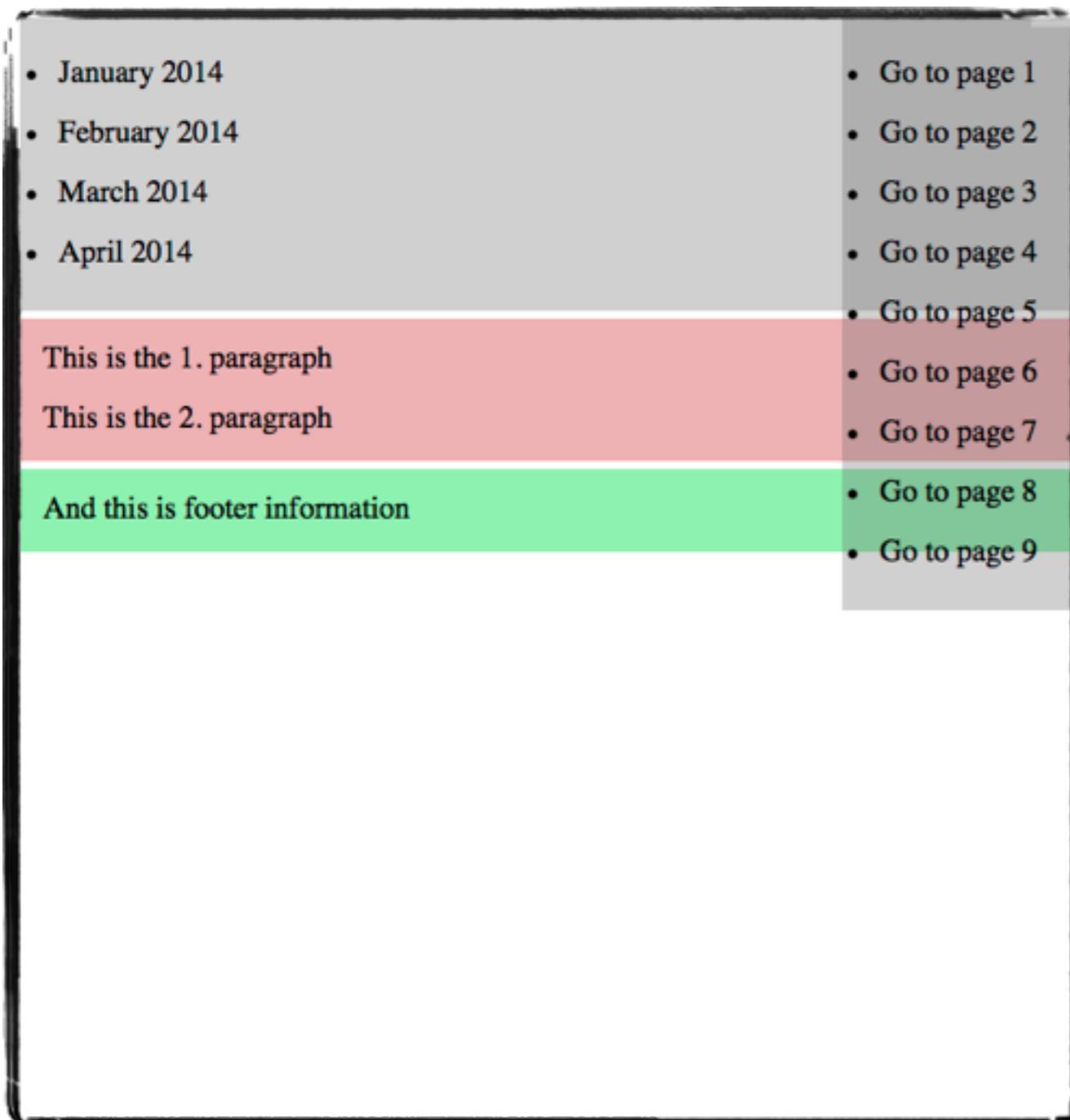
Canonical example: **adding sidebars** to a layout



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

Resetting the flow with clear

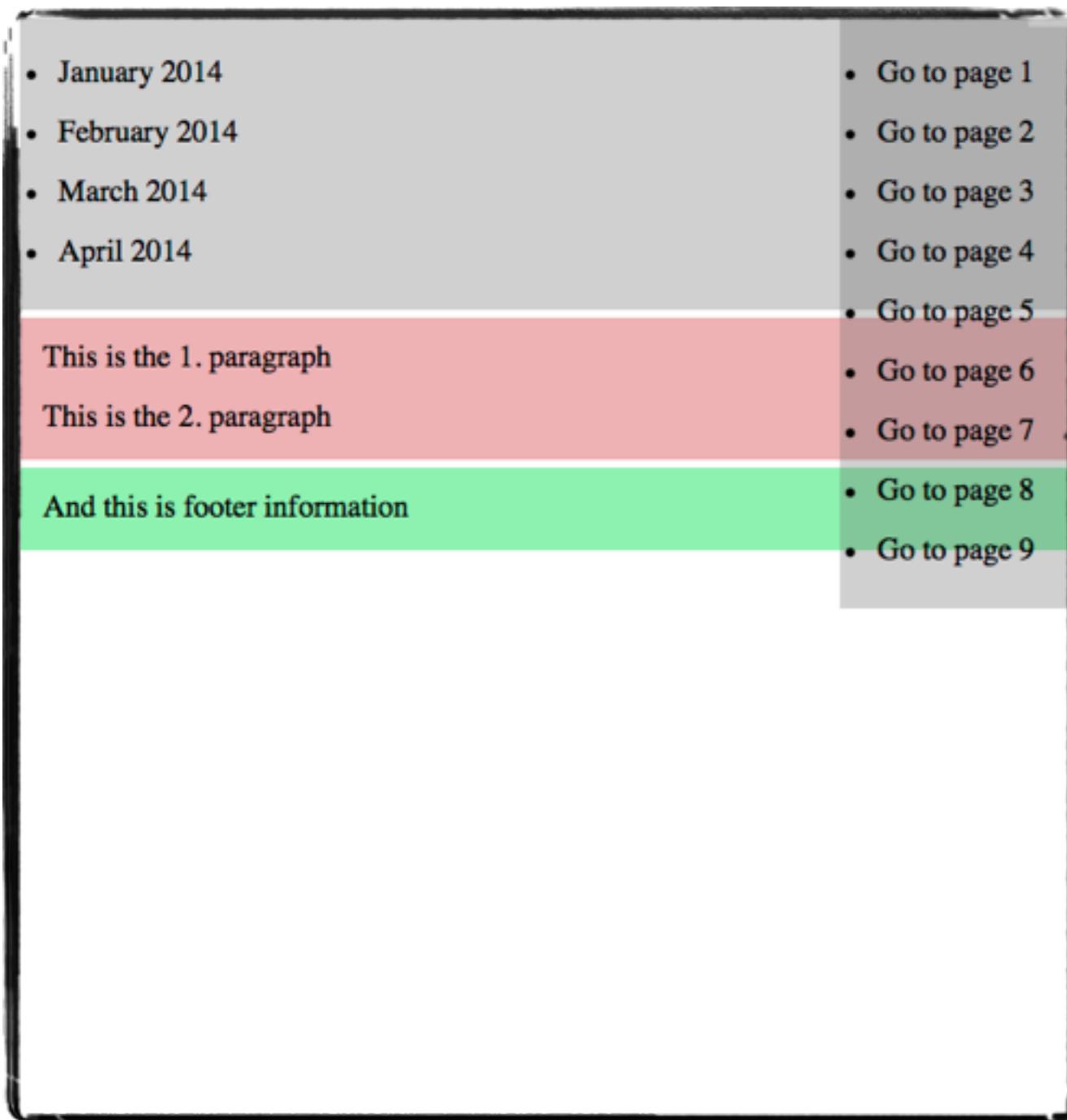
Canonical example: **adding sidebars** to a layout



```
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

The diagram illustrates a layout structure with a main content area and two floating sidebars. The left sidebar (nav1) is positioned on the left and contains a list of months from January 2014 to April 2014. The right sidebar (nav2) is positioned on the right and contains a list of links from 'Go to page 1' to 'Go to page 9'. The main content area is centered and contains two paragraphs: 'This is the 1. paragraph' and 'This is the 2. paragraph', followed by a footer message: 'And this is footer information'.

• January 2014	This is the 1. paragraph	• Go to page 1
• February 2014	This is the 2. paragraph	• Go to page 2
• March 2014	And this is footer information	• 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

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

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

Resetting the flow with clear

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

The diagram illustrates a layout structure with a sidebar on the left and two main content areas on the right. The sidebar contains a list of months from January 2014 to April 2014. The first main content area is pink and contains two paragraphs: "This is the 1. paragraph" and "This is the 2. paragraph". The second main content area is green and contains the text "And this is footer information". To the right of these areas is a vertical sidebar with a list of links: "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", and "Go to page 9".

```
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
- February 2014
- March 2014
- April 2014

This is the 1. paragraph

This is the 2. paragraph

- 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;}
```

Resetting the flow with clear

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

The screenshot shows a layout with three main sections. On the left, there is a sidebar with a purple header containing a bulleted list of months from January 2014 to April 2014. Below this is a green footer section with the text "And this is footer information". In the center, there is a red header section with two paragraphs: "This is the 1. paragraph" and "This is the 2. paragraph". To the right of the red header is another sidebar with a purple header containing a bulleted list of links from "Go to page 1" to "Go to page 9". The layout demonstrates how clearing floats (using the clear property) affects the vertical flow of content.

```
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

• January 2014 This is the 1. paragraph

• February 2014 This is the 2. paragraph

• March 2014

• April 2014

This is the 1. paragraph

This is the 2. paragraph

- 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;}
```

Resetting the flow with clear

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

The layout consists of three columns:

- Left Column:** Contains a sidebar with a list of months: January 2014, February 2014, March 2014, and April 2014.
- Middle Column:** Contains two paragraphs of text: "This is the 1. paragraph" and "This is the 2. paragraph".
- Right Column:** Contains a list of nine items, each labeled "Go to page 1" through "Go to page 9".

A green footer bar at the bottom contains the text "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 of elements: position

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

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.

position:static

the default

position:relative

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

position:absolute

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

position:fixed

similar to absolute, but fixed to the **viewport**

position:sticky

in-between relative and fixed

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.

position:static

the default

position:relative

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

position:absolute

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

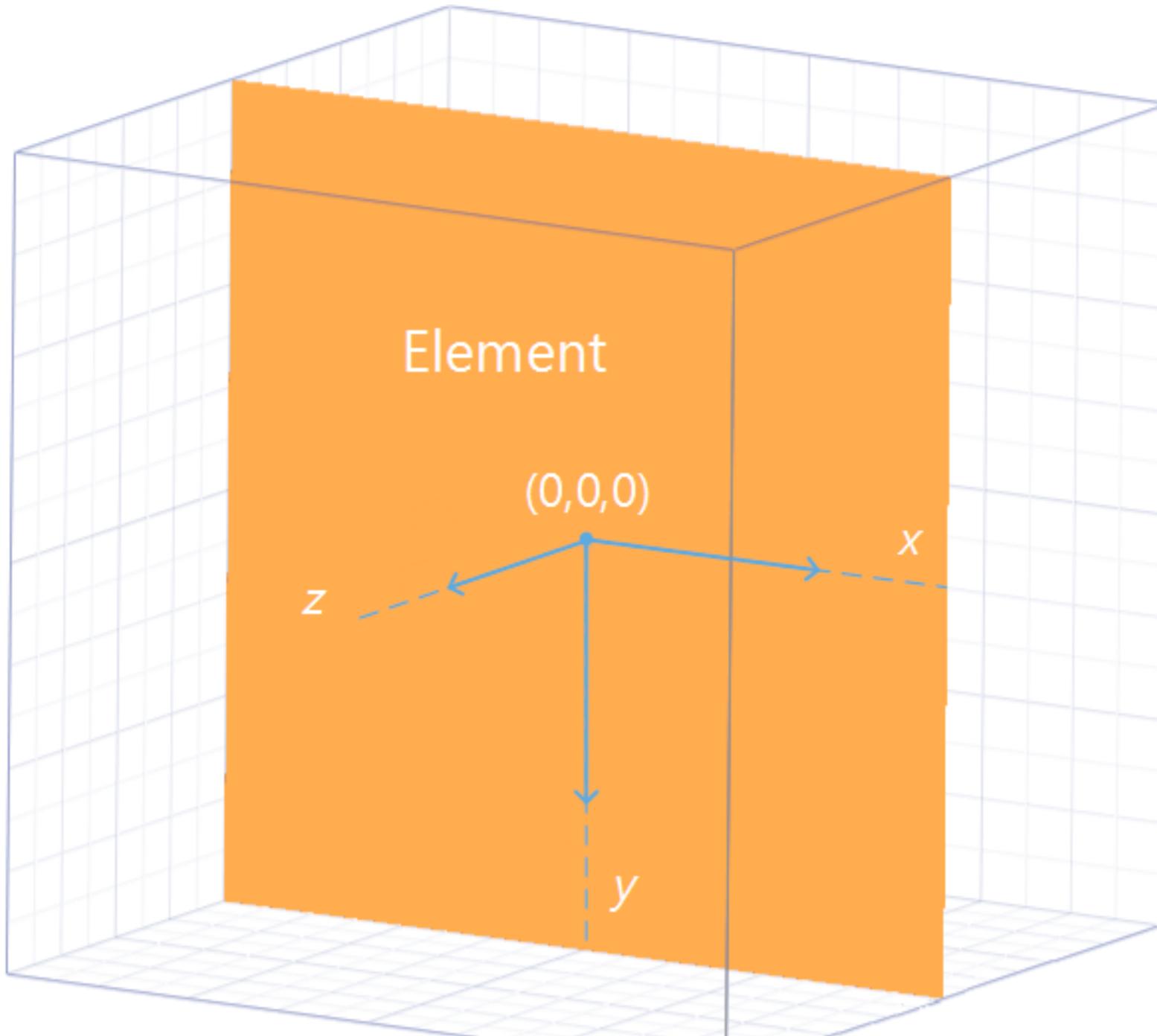
position:fixed

the area currently being viewed
viewport the

position:sticky

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

position: relative

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

movement is **relative** to its original position



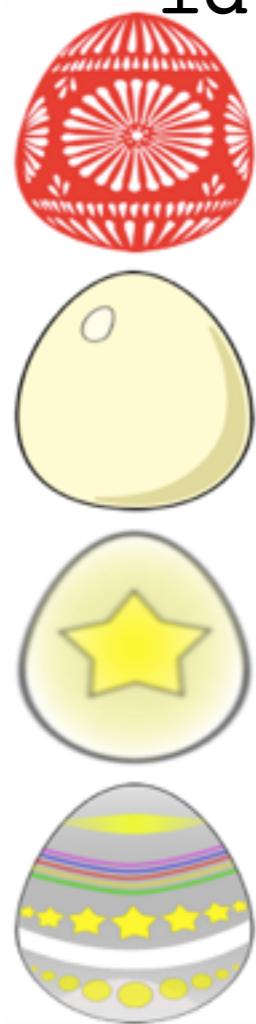
`id="egg1"`

`id="egg4"`

position: relative

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

movement is **relative** to its original position



`id="egg1"`

```
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"`

position: relative

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

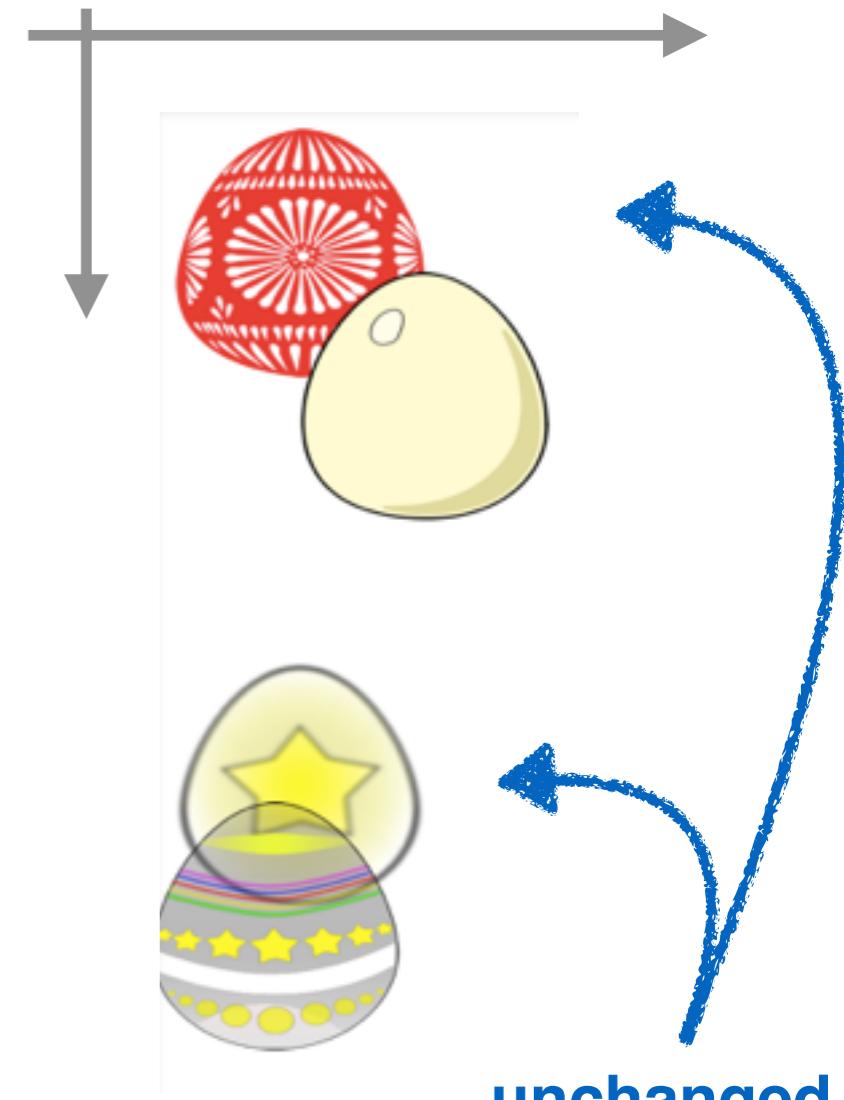
movement is **relative** to its original position

id="egg1"



id="egg4"

```
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 }
```



unchanged

position: absolute

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

position: absolute

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

positioning is relative to nearest ancestor or the window

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"`



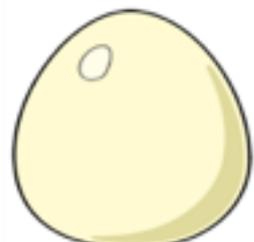
`id="egg4"`

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"`



`id="egg4"`

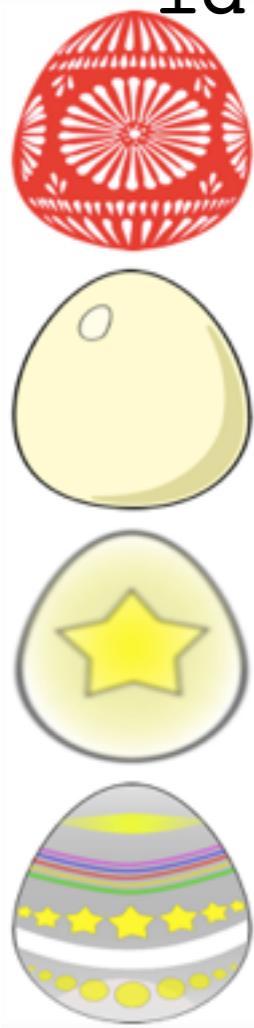
```
1 #egg2 {  
2   position: absolute;  
3   bottom: 50px;  
4   left: 0px;  
5 }  
6  
7 #egg4 {  
8   position: absolute;  
9   bottom: 0px;  
10  right: 0px;  
11 }
```

position: absolute

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

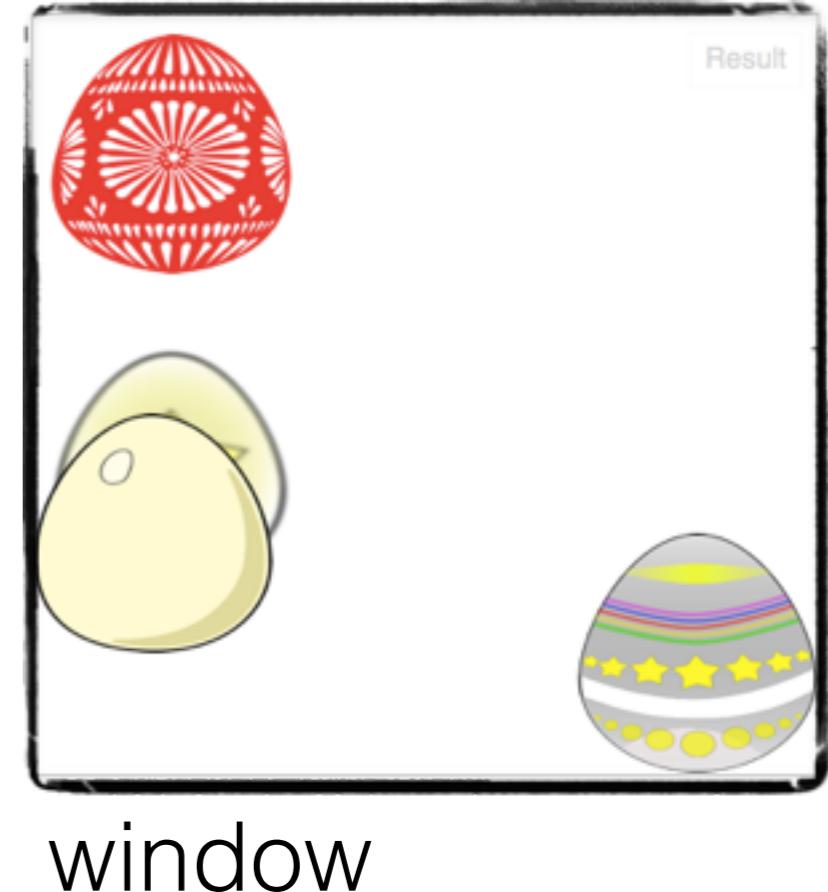
positioning is relative to nearest ancestor or the window

`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"`



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 Mozilla Thimble editor interface. The top bar includes tabs for 'FILES' and 'EDITOR - index.html'. The 'EDITOR' tab displays the following HTML code:

```
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' tab shows a todolist application with three sections: 'Today's todos' (27/11/2015), 'Tomorrow's todos' (28/11/2015), and 'Looking ahead'. Each section contains a list of tasks and an 'ADD TODO' button. A yellow callout box on the right side of the screen contains the text: 'Ads should always be visible.'

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 Mozilla Thimble editor interface. The top bar includes tabs for 'FILES' and 'EDITOR - index.html'. The 'EDITOR' tab displays the following HTML code:

```
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' tab shows a todolist application with three sections: 'Today's todos' (27/11/2015), 'Tomorrow's todos' (28/11/2015), and 'Looking ahead'. Each section contains a list of tasks and an 'ADD TODO' button. A yellow callout box on the right side of the screen contains the text: 'Ads should always be visible.'

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

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

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

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.

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

display

display:inline

element rendered with an inline element box

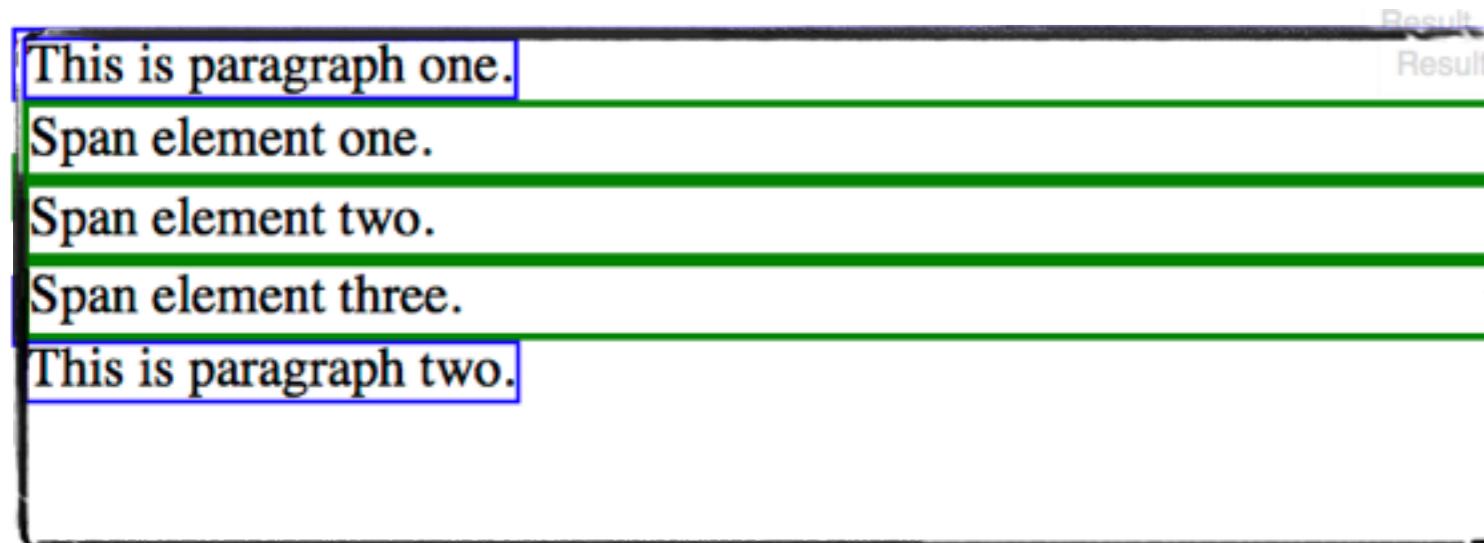
display:block

element rendered with a block element box

display:none

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

most useful to us



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

display

display:inline

element rendered with an inline element box

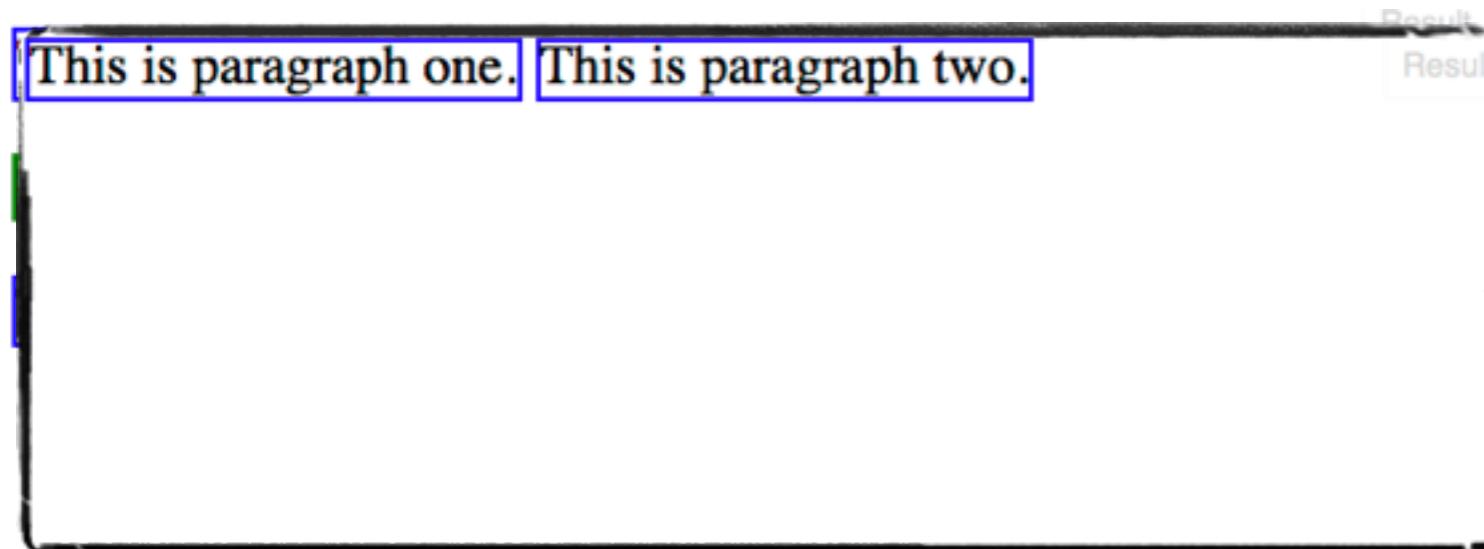
display:block

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...

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/>)

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

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

HTML: write once

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

HTML: write once

CSS: write once per device

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/>) **HTML5 technology**
- CSS media queries enable the use of **device-dependent** (i.e. media-type dependent) stylesheets

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>
```

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>
```

dedicated CSS files

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 { rules for different devices in one file
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>
```

dedicated CSS files

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 { rules for different devices in one file
8     body {
9       color: black !important;
10      width: 100%; when printing, use black and white
11    }
12  }
13  @media screen and (max-width: 300px) {
14    #sidebar {
15      display: none;
16    }
17  }
18 </style>
```

dedicated CSS files

rules for different devices in one file

when printing, use black and white

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 { rules for different devices in one file
8     body {
9       color: black !important;
10      width: 100%; when printing, use black and white
11    }
12  }
13  @media screen and (max-width: 300px) {
14    #sidebar {
15      display: none; hide the sidebar for small devices
16    }
17  }
18 </style>
```

dedicated CSS files

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 { rules for different devices in one file
8     body {
9       color: black !important;
10      width: 100%; when printing, use black and white
11    }
12  } "and": logical and
13  @media screen and (max-width: 300px) {
14    #sidebar {
15      display: none; hide the sidebar for small devices
16    }
17  }
18 </style>
```

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

Media queries can be complex

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

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

Animations and transitions

Mozilla Thimble Mozilla Foundation [US] https://thimble.mozilla.org/user/claudiahauff/64593

Animations

FILES index.html style.css

EDITOR - style.css

PREVIEW

```
167 border-style: solid;
168 border-color: #333;
169 border-radius: 50%;
170
171 /* ... your other orbit styles ... */
172
173 -webkit-animation: spin-left 35s linear infinite;
174 -moz-animation: spin-left 35s linear infinite;
175 -ms-animation: spin-left 35s linear infinite;
176 -o-animation: spin-left 35s linear infinite;
177 animation: spin-left 35s linear infinite;
178 }
179
180
181 @-webkit-keyframes spin-left {
182 100% {
183 -webkit-transform: rotate(-360deg);
184 -moz-transform: rotate(-360deg);
185 -ms-transform: rotate(-360deg);
186 -o-transform: rotate(-360deg);
187 transform: rotate(-360deg);
188 }
189 }
190
191 @keyframes spin-left {
192 100% {
193 -webkit-transform: rotate(-360deg);
194 -moz-transform: rotate(-360deg);
195 -ms-transform: rotate(-360deg);
196 -o-transform: rotate(-360deg);
197 transform: rotate(-360deg);
198 }
199 }
200
201 }
```

Mozilla Thimble Mozilla Foundation [US] https://thimble.mozilla.org/user/claudiahauff/64593

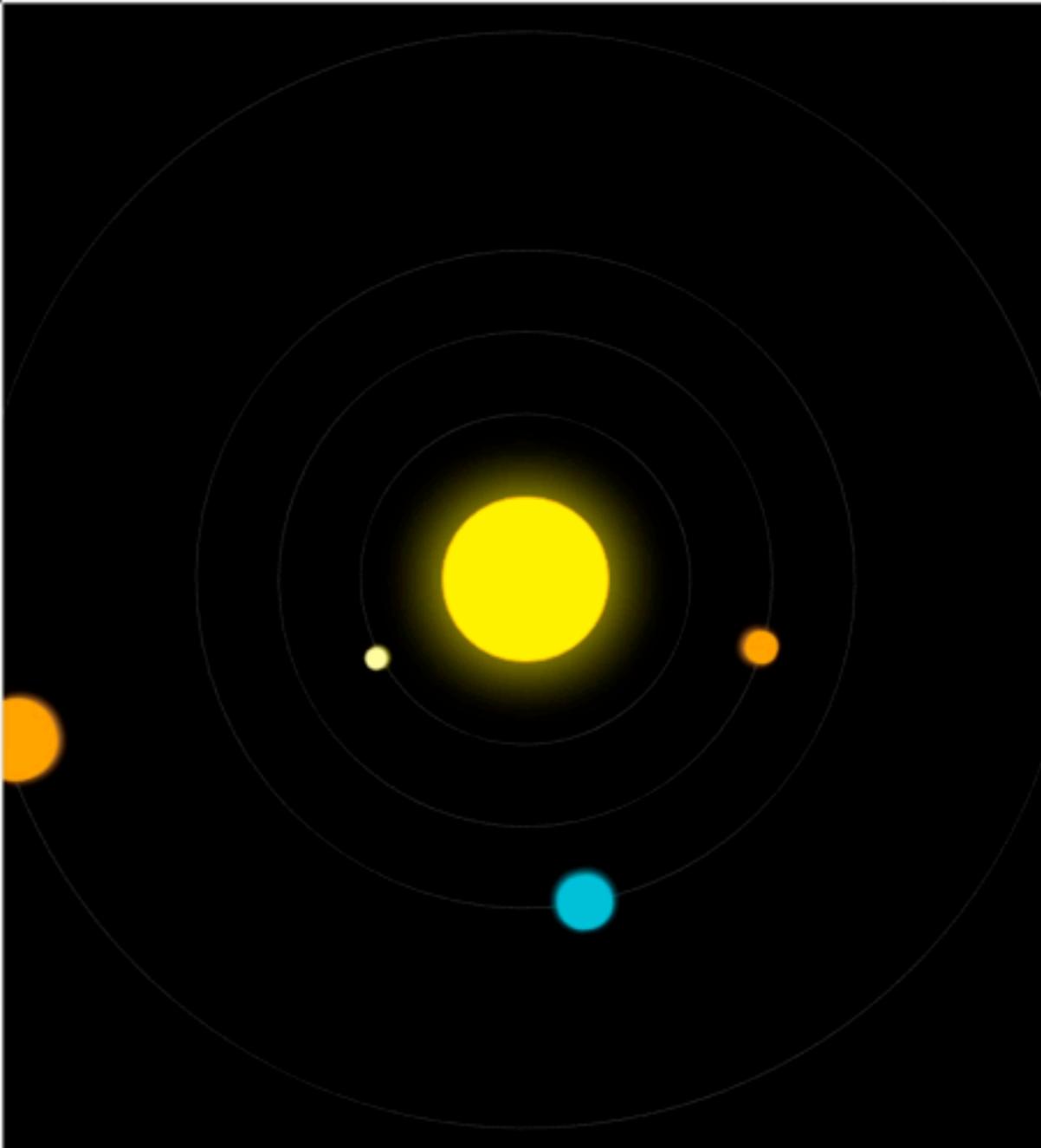
Animations

FILES index.html style.css

EDITOR - style.css

PREVIEW

```
167 border-style: solid;
168 border-color: #333;
169 border-radius: 50%;
170
171 /* ... your other orbit styles ... */
172
173 -webkit-animation: spin-left 35s linear infinite;
174 -moz-animation: spin-left 35s linear infinite;
175 -ms-animation: spin-left 35s linear infinite;
176 -o-animation: spin-left 35s linear infinite;
177 animation: spin-left 35s linear infinite;
178 }
179
180
181 @-webkit-keyframes spin-left {
182 100% {
183 -webkit-transform: rotate(-360deg);
184 -moz-transform: rotate(-360deg);
185 -ms-transform: rotate(-360deg);
186 -o-transform: rotate(-360deg);
187 transform: rotate(-360deg);
188 }
189 }
190
191 @keyframes spin-left {
192 100% {
193 -webkit-transform: rotate(-360deg);
194 -moz-transform: rotate(-360deg);
195 -ms-transform: rotate(-360deg);
196 -o-transform: rotate(-360deg);
197 transform: rotate(-360deg);
198 }
199 }
200
201 }
```



In general ...

In general ...

- CSS styles (states) are defined by the user, the rendering engine takes care of the transition between styles

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

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

CSS vs. JavaScript animations

- **Easy to use** (standard CSS) — no need to learn JavaScript

CSS vs. JavaScript animations

- **Easy to use** (standard CSS) — no need to learn JavaScript
- Rendering engines are **optimised** for CSS-based animations

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; duration of animation (seconds)  
3     animation-name: pToRight; animation name (@keyframes)  
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 }
```

CSS animation example (Firefox)

```
1 #p1 {  
2     animation-duration: 5s; duration of animation (seconds)  
3     animation-name: pToRight; animation name (@keyframes)  
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 }
```

start state

CSS animation example (Firefox)

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

CSS animation example (Firefox)

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

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

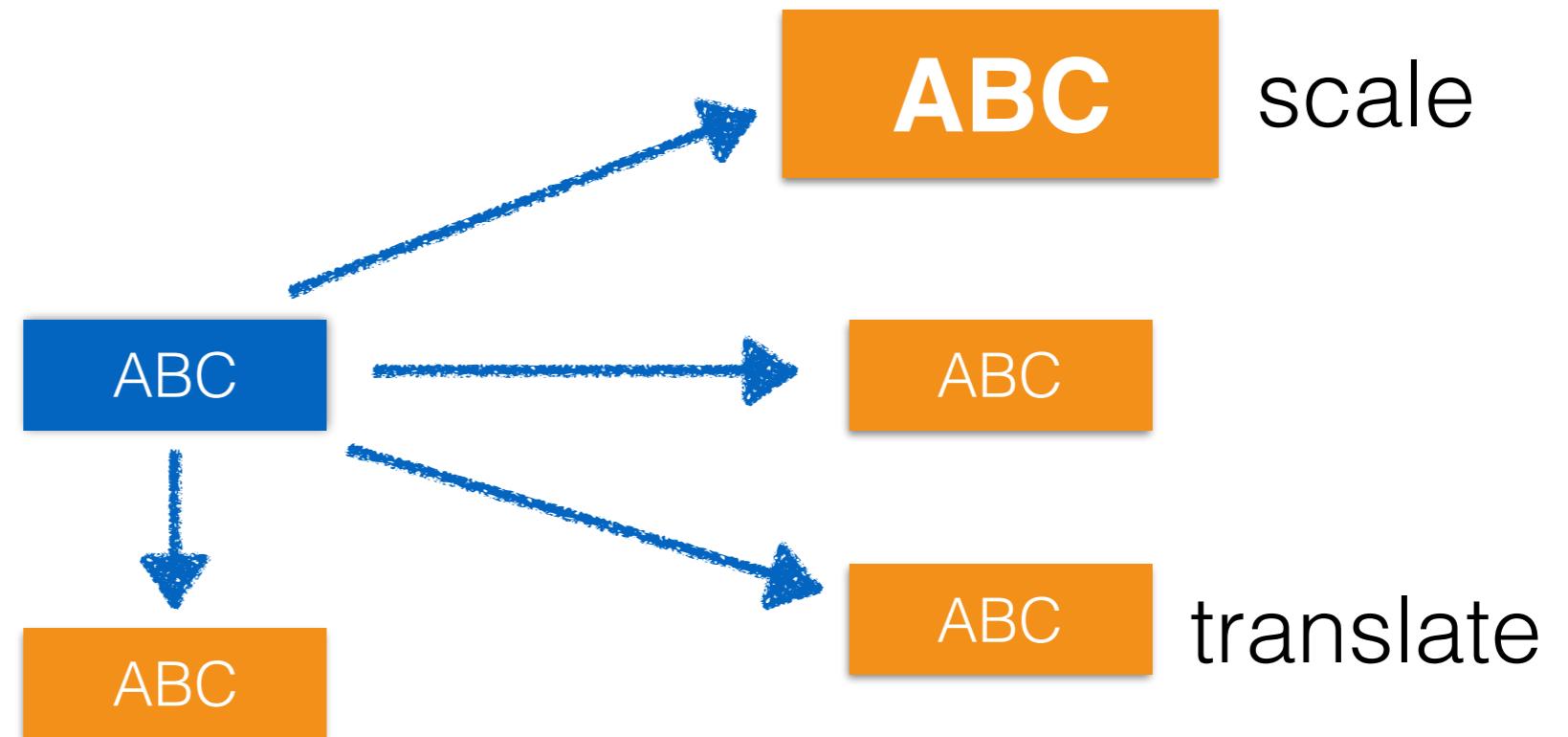
by default the animation restarts at the starting keyframe; if set to **alternate** the animation direction change every iteration

animation-delay

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

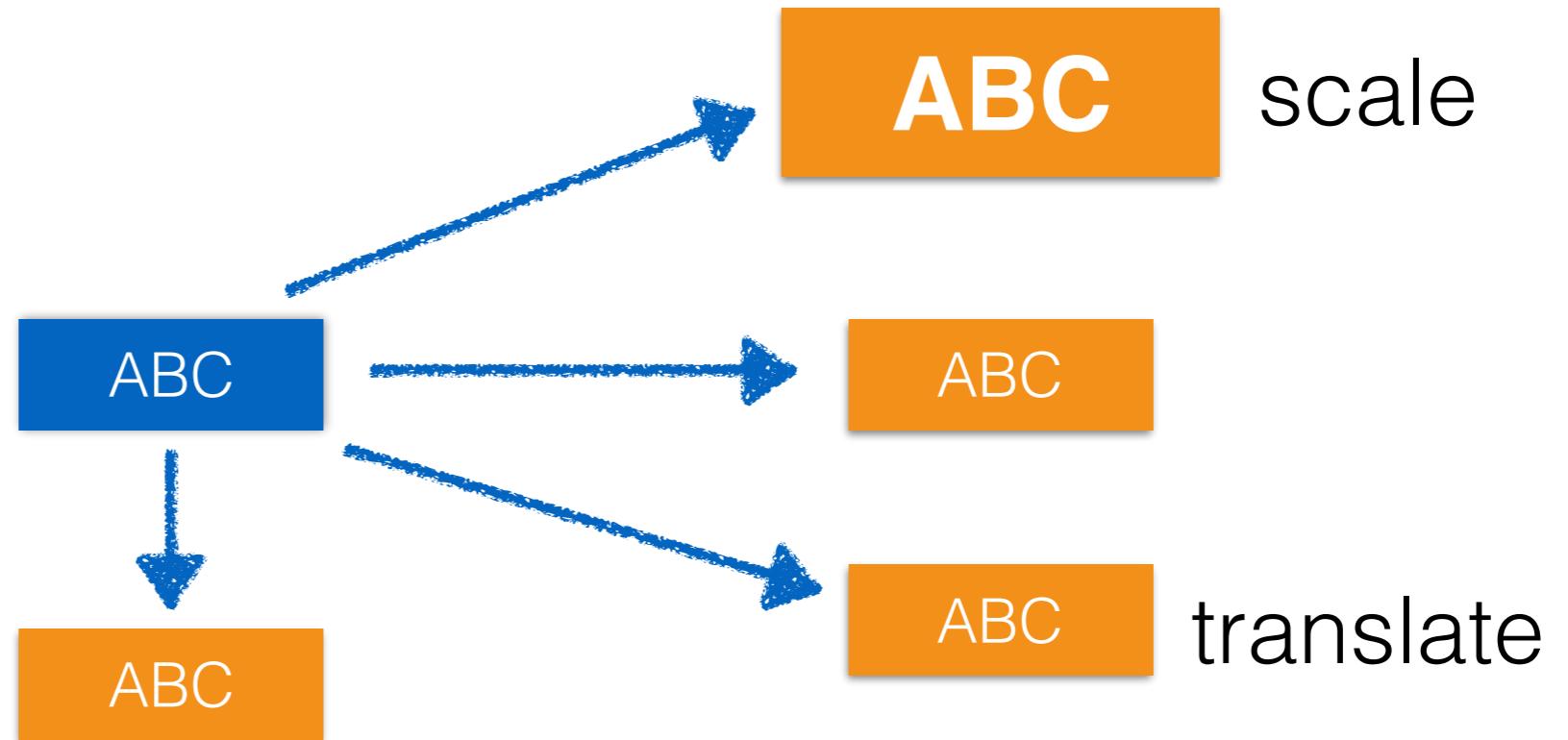
transform

So far we can:



transform

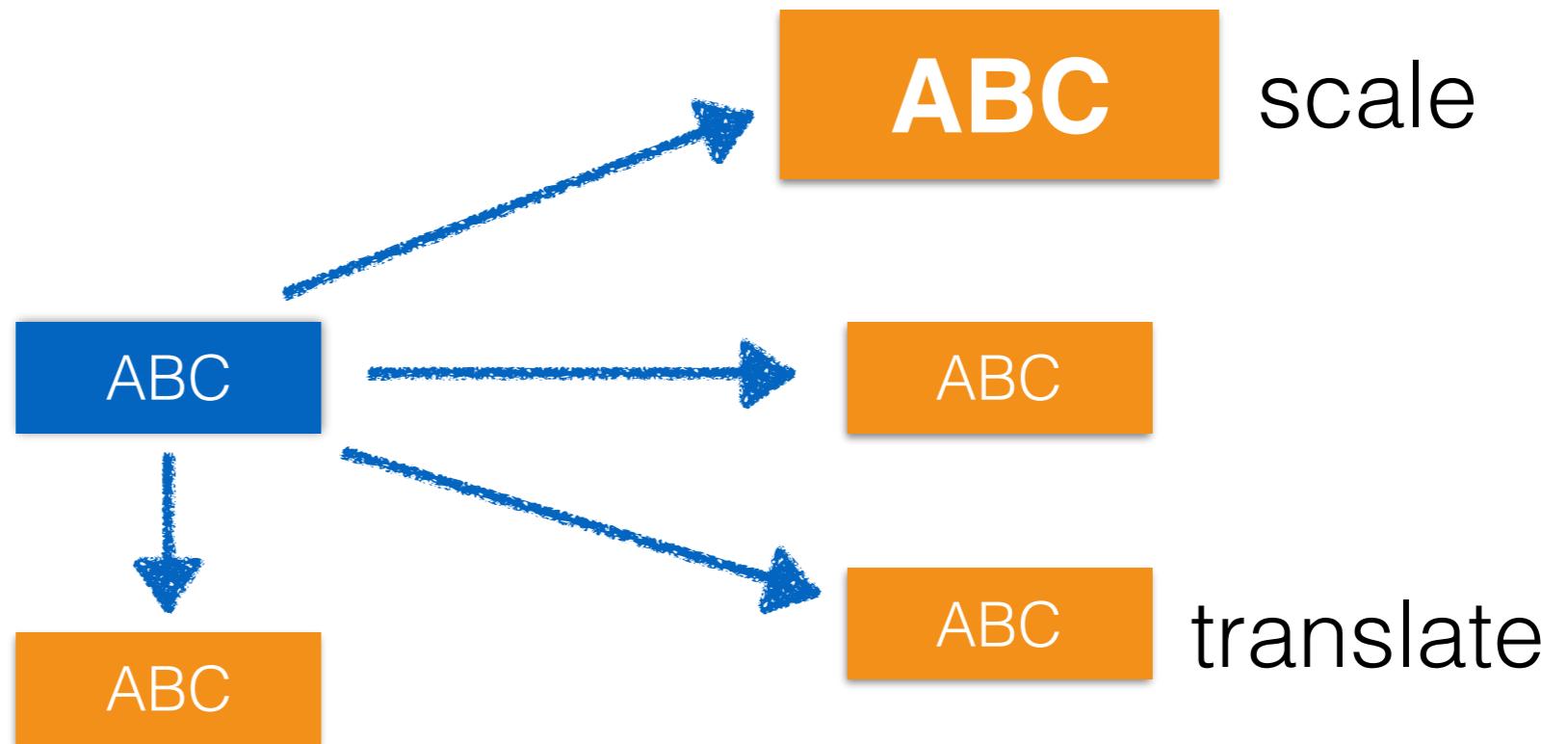
So far we can:



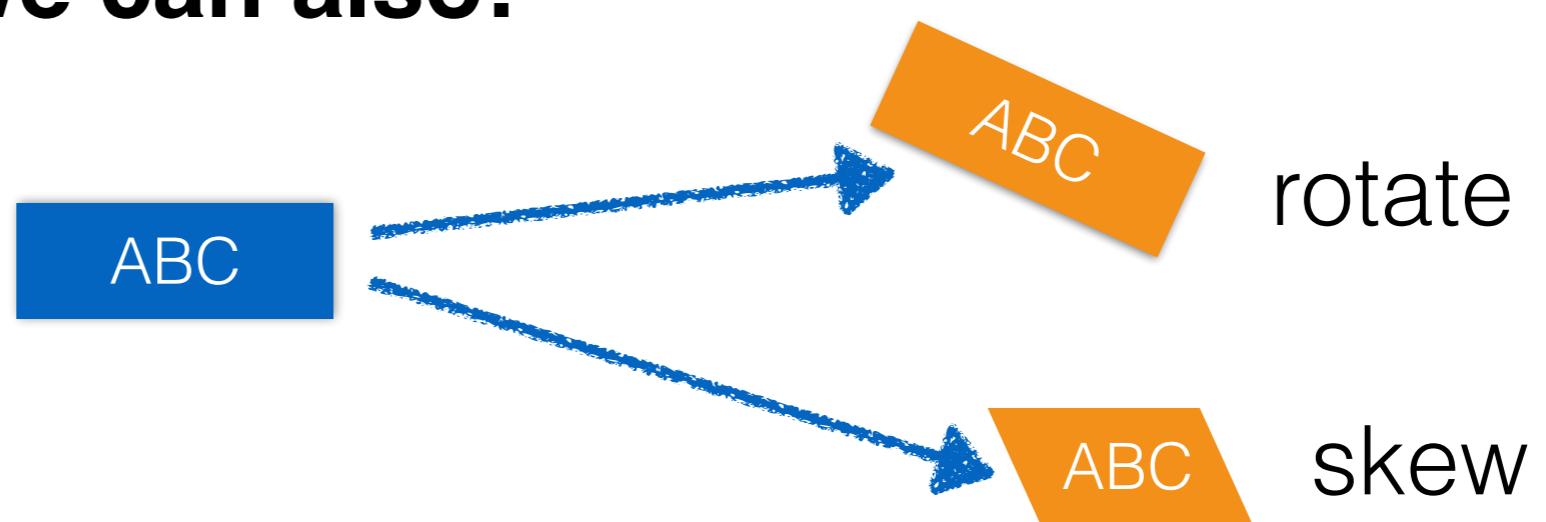
With transform we can also:

transform

So far we can:

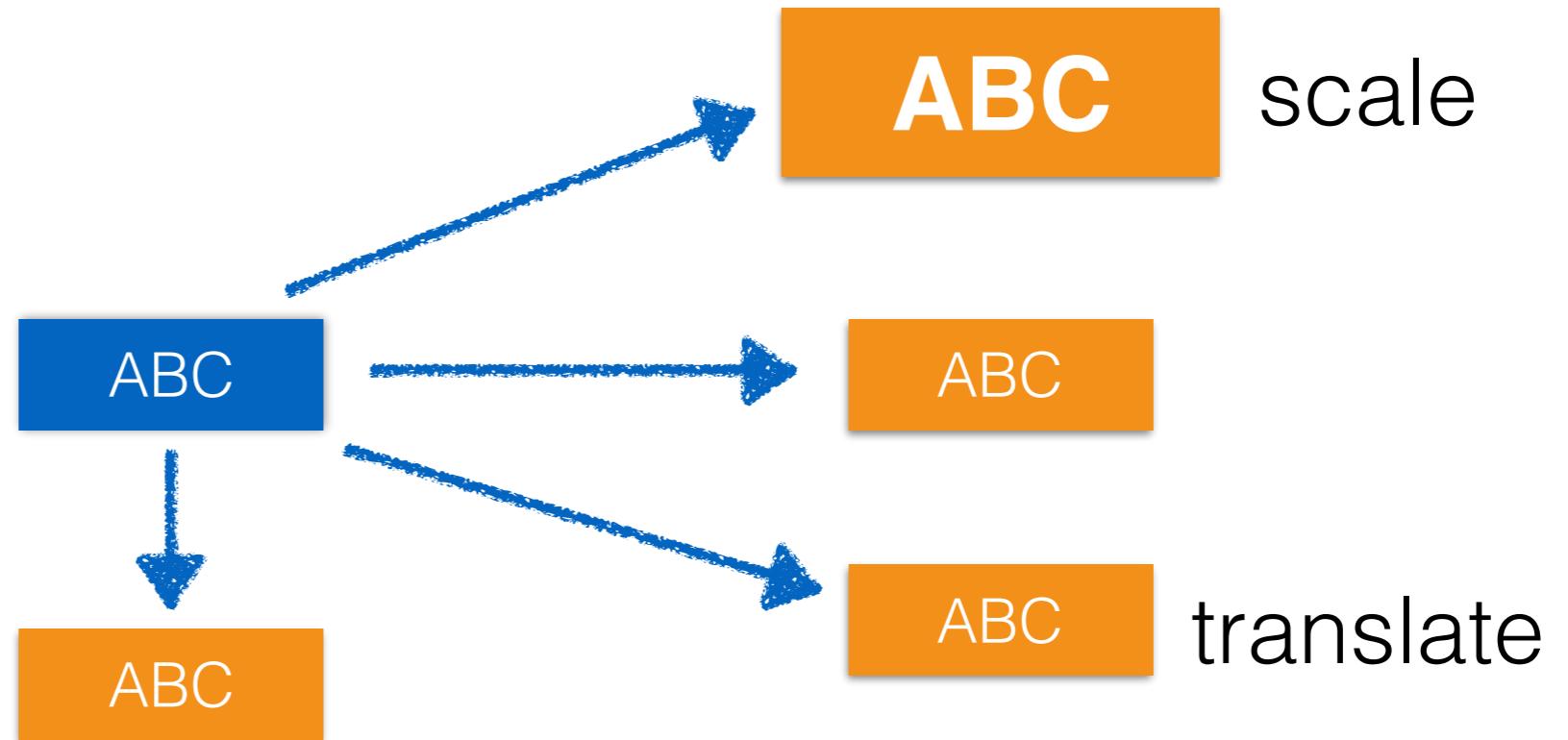


With `transform` we can also:

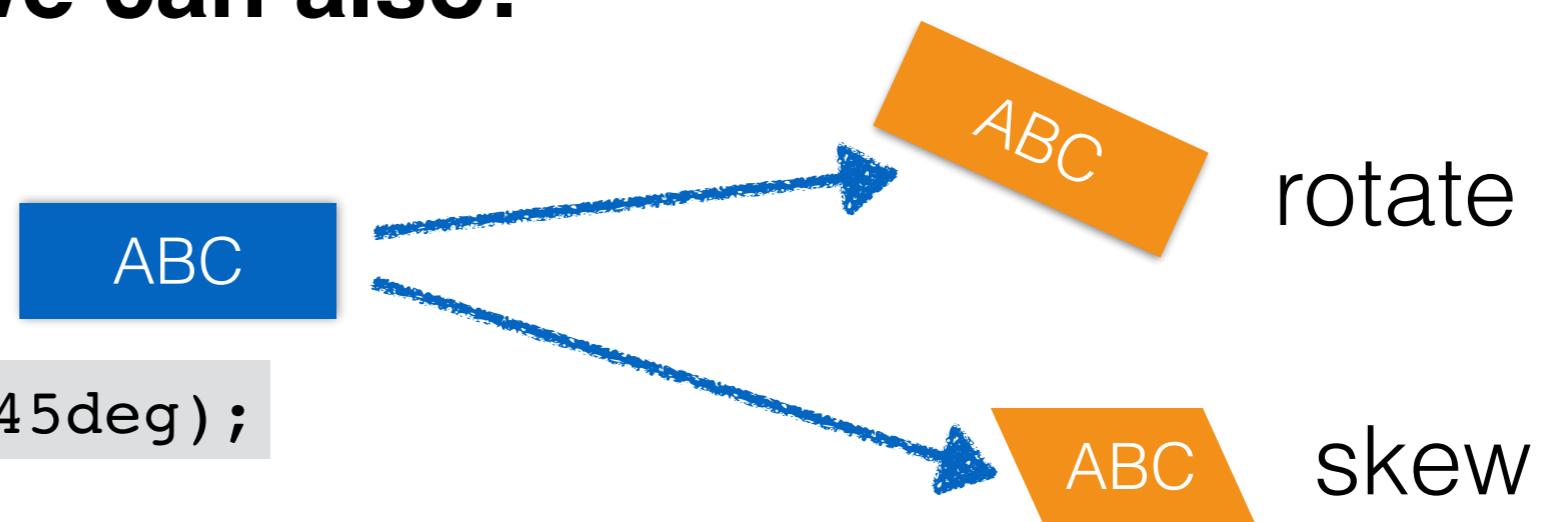


transform

So far we can:



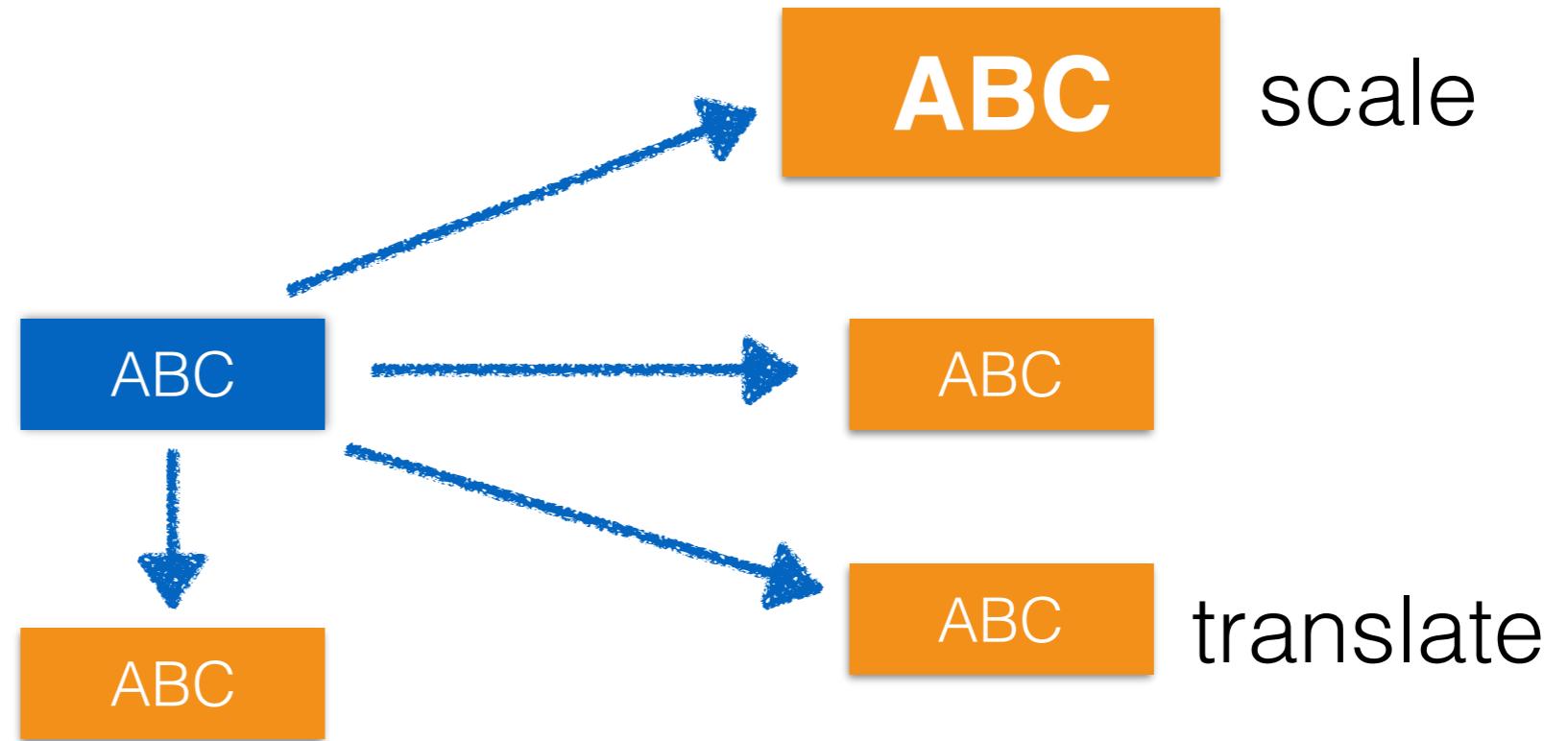
With **transform** we can also:



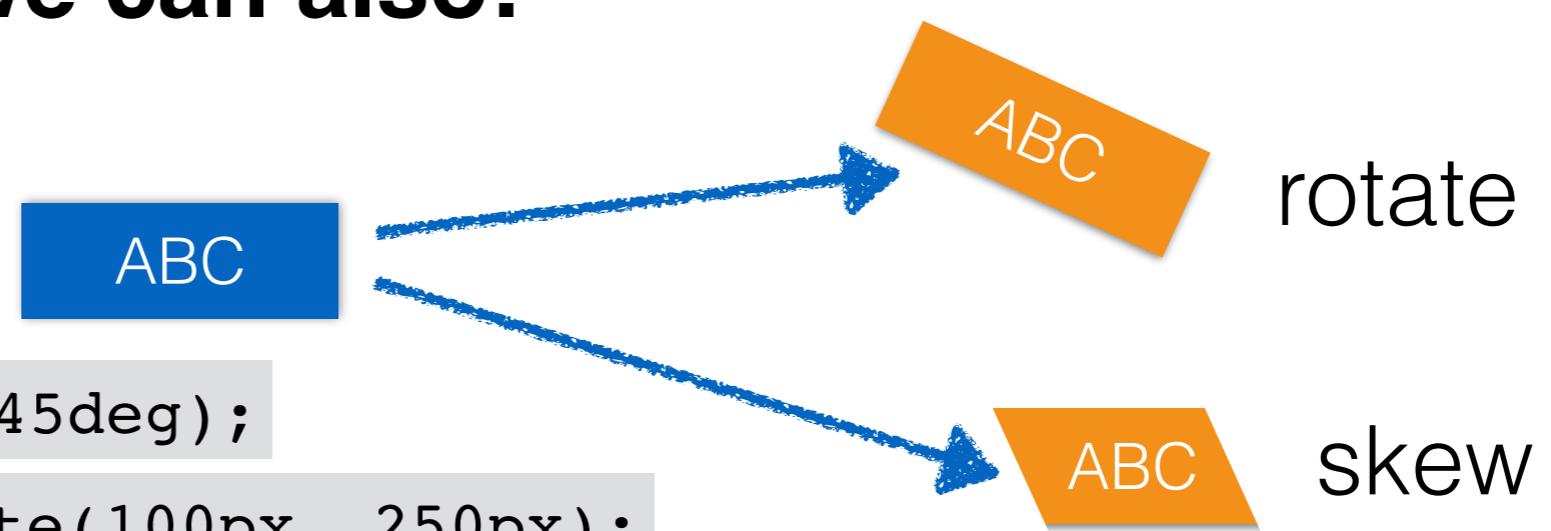
```
1 transform: rotate(45deg);
```

transform

So far we can:



With `transform` we can also:

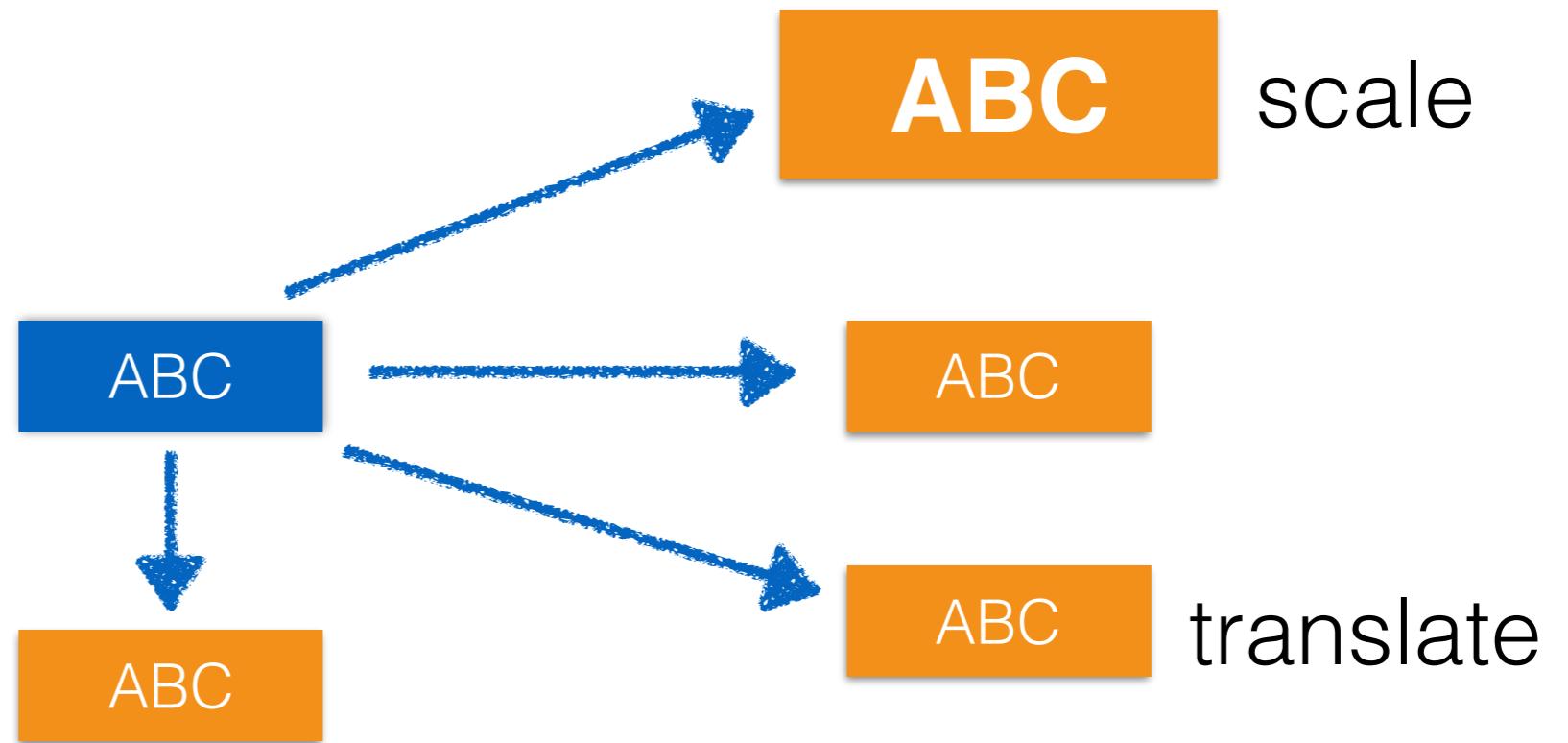


```
1 transform: rotate(45deg);
```

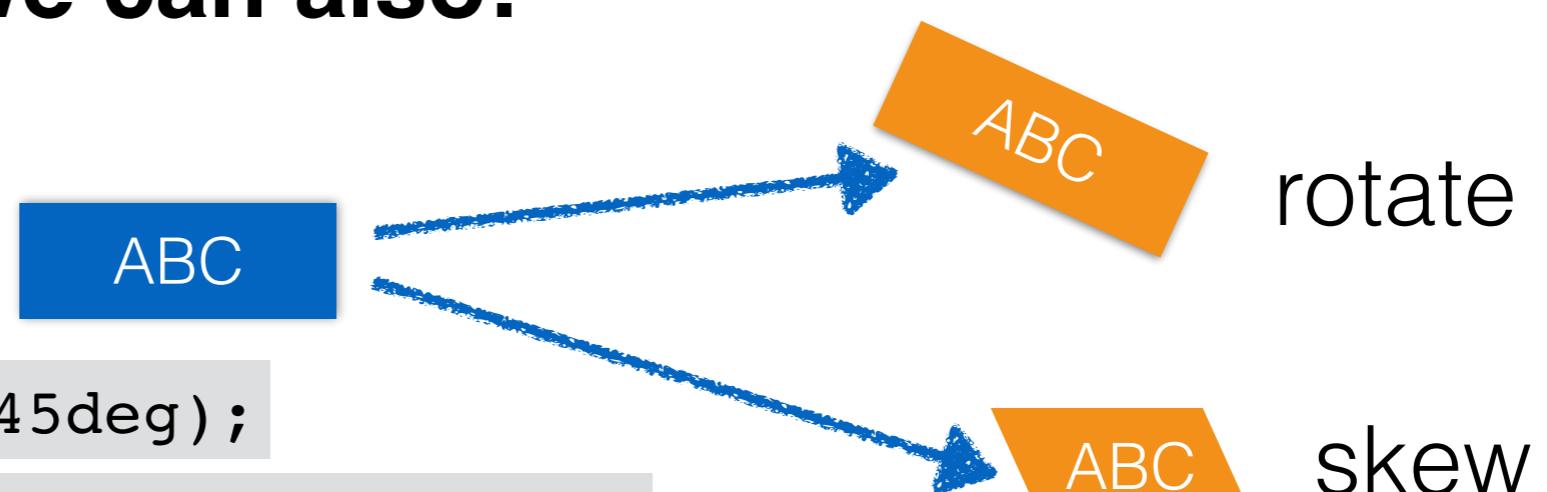
```
2 transform: translate(100px, 250px);
```

transform

So far we can:



With **transform** we can also:



```
1 transform: rotate(45deg);
```

```
2 transform: translate(100px, 250px);
```

```
3 transform: translate(10px,10px) skew(20deg);
```

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 }
```

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

CSS transitions

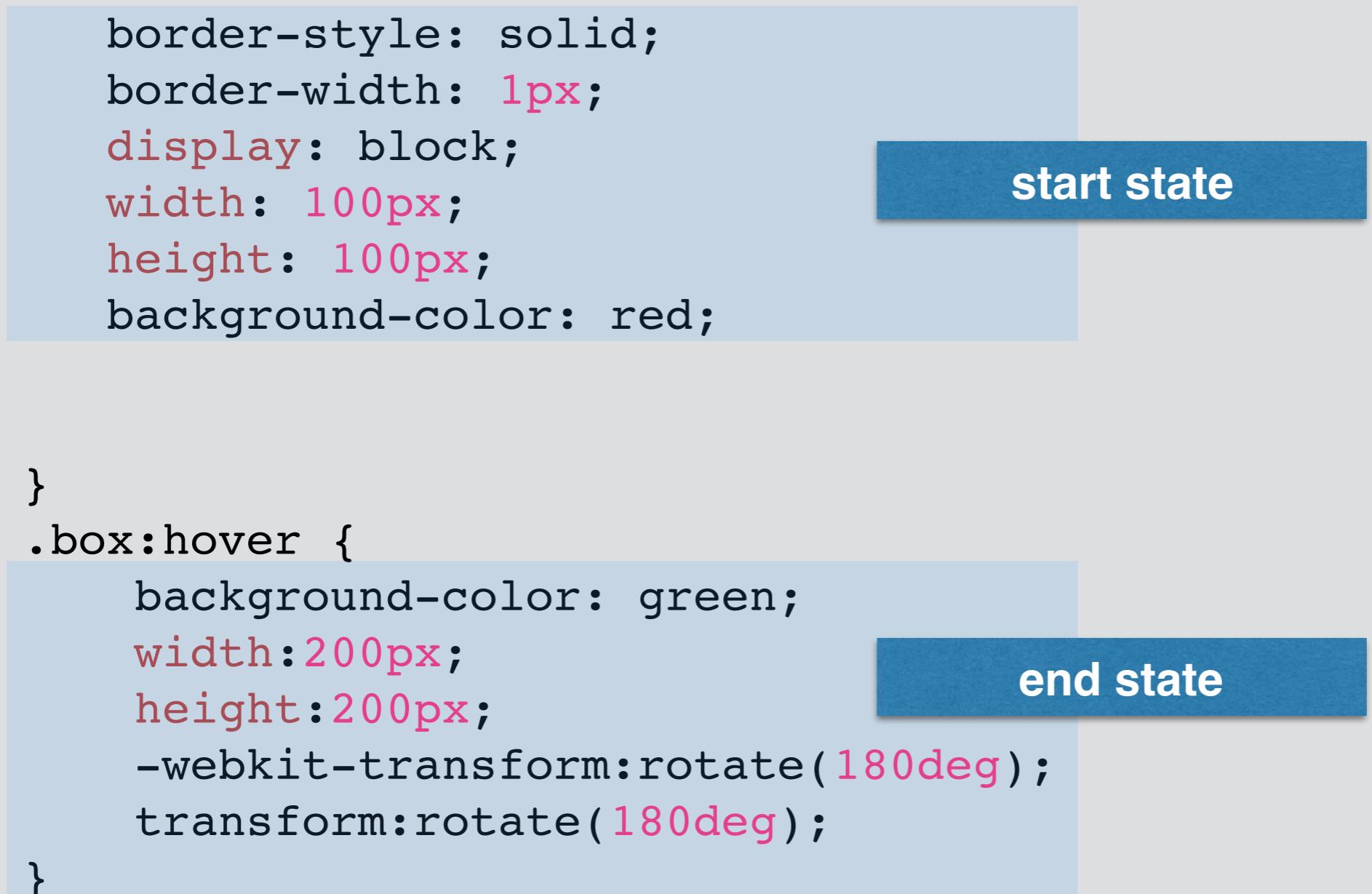
```
1 .box {  
2     border-style: solid;  
3     border-width: 1px;  
4     display: block;  
5     width: 100px;  
6     height: 100px;  
7     background-color: red;  
8  
9  
10 }  
11 .box:hover {  
12     background-color: green;  
13     width: 200px;  
14     height: 200px;  
15     -webkit-transform: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

start state

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

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 }
```

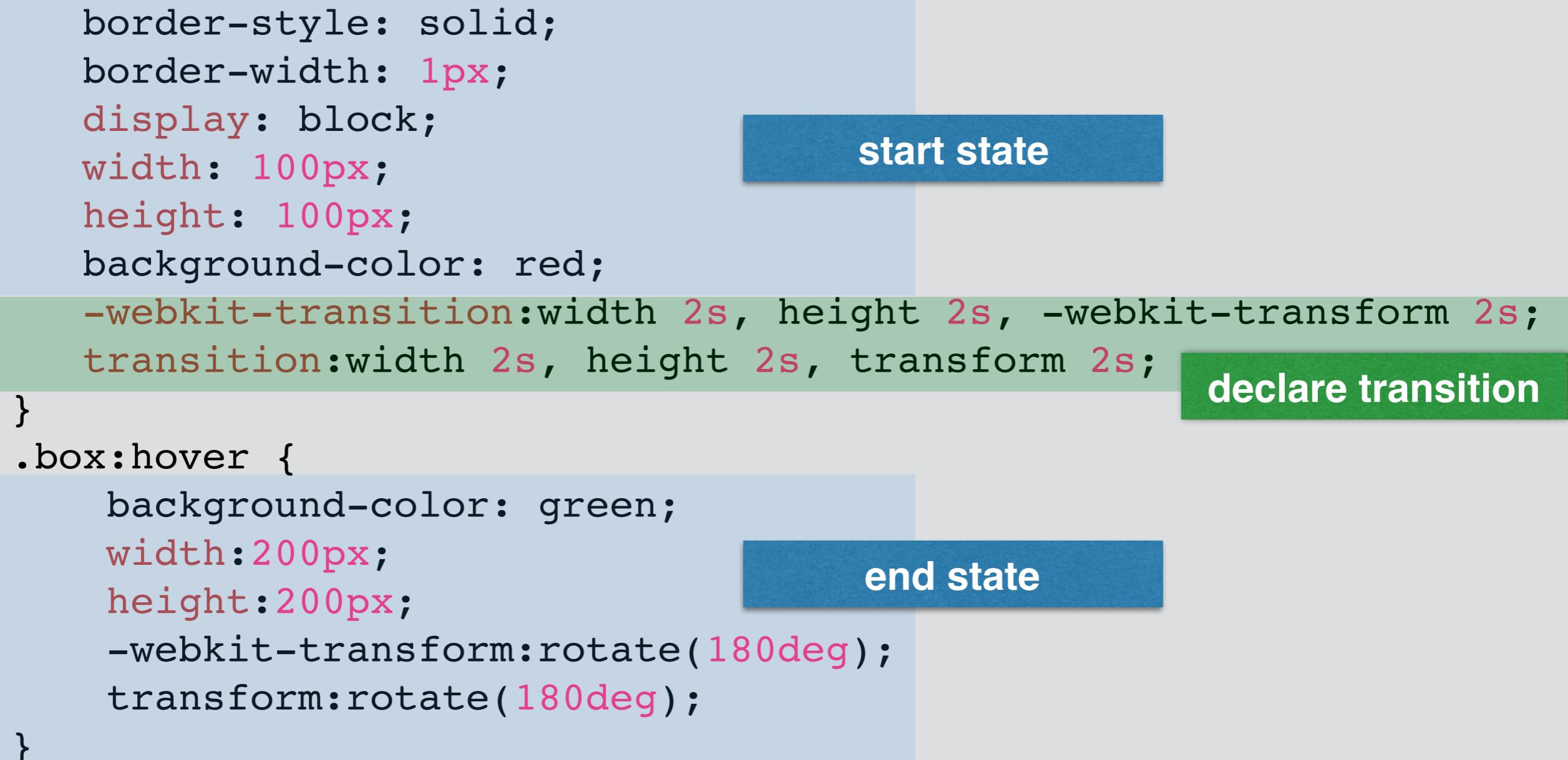


The diagram illustrates the start state and end state of a CSS transition. The start state is a red box with dimensions 100px by 100px. The end state is a green box with dimensions 200px by 200px, rotated 180 degrees. The code above defines these states.

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

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     -webkit-transition:width 2s, height 2s, -webkit-transform 2s;  
9     transition:width 2s, height 2s, transform 2s;      declare transition  
10 }  
11 .box:hover {  
12     background-color: green;  
13     width:200px;  
14     height:200px;  
15     -webkit-transform:rotate(180deg);  
16     transform:rotate(180deg);  
17 }
```



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

**Ensure the IMDB database is
available in your VM.**

Needed in Monday's lecture.