# CSS: the language of Web design

Claudia Hauff
cse1500-ewi@tudelft.nl

TU Delft

# Web technology overview

1. **HTTP**: the language of web communication
2. **HTML** & web app design
3. **JavaScript**: interactions in the browser
4. **Node.js**: JavaScript on the server
5. **CSS**: adding style
6. Node.js: advanced topics
7. Cookies & sessions
8. Web security

# Learning goals

- **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
- **Write** CSS media queries
- **Create** CSS-based animations

# A word of warning ...

# A bit of context

# Cascading Style Sheets (CSS)

- **CSS1**: W3C recommendation in **1996**
  - Fonts, colours, alignment, margins, ids and classes

- **CSS2**: W3C recommendation in **1998**
  - Media queries, element positioning, etc.

- **CSS2.1**: W3C recommendation in **2011**
  - Fixed errors and already implemented browser features

- **CSS3**: **specification split up** into modules; progress varies
  - https://www.w3.org/Style/CSS/current-work
  - *shims, fallbacks, polyfills*

# A tale of many modules

| Refining | Current | Upcoming | Notes |
|---|---|---|---|
| CSS Animations Level 1 | WD | WD | |
| Web Animations | WD | WD | |
| CSS Text Level 3 | WD | CR | |
| CSS Transforms | WD | WD | |
| CSS Transitions | WD | CR | |
| CSS Box Alignment Level 3 | WD | WD | |
| Selectors Level 4 | WD | WD | |
| Motion Path Level 1 | WD | WD | |
| Preview of CSS Level 2 | FPWD | NOTE | |
| CSS Fonts Level 4 | WD | WD | |
| CSS Easing Functions Level 1 | WD | WD | |
| CSS Logical Properties and Values Level 1 | WD | WD | |

| Revising | Current | Upcoming | Notes |
|---|---|---|---|
| CSS Paged Media Level 3 | WD | WD | |
| CSSOM View | WD | WD | |
| CSS Intrinsic & Extrinsic Sizing Level 3 | WD | CR | |
| CSS Ruby Level 1 | WD | WD | |
| CSS Overflow Level 3 | WD | WD | |
| CSS Box Model Level 3 | WD | WD | |
| CSS Pseudo-Elements Level 4 | WD | WD | |

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

# CSS3

- Impossible to write complex CSS that relies on modern features and works across all browsers

- Implementation of CSS3 features should be decided based on:
  - **intended users**
  - **mode of usage**
  - **type** of Web app

# Revision chapter 3

# Chapter 3 of the course book

**CSS describes how elements in the DOM should be rendered.**

```
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

overriding

Style sheets are processed in order; **later declarations override earlier ones** **(if they are on the same or a higher *specificity* level)**

**`!important`** overrides all other declarations

# Pseudo-elements and pseudo-classes

# Pseudo-class

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

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

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

# Children & types

| | |
|---|---|
| **`: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 |

```html
<main>
  <h2>Todos</h2>
  <p>Today's todos</p>
  <p>Tomorrow's todos</p>
  <p>Saturday's todos</p>
  <p>Sunday's todos</p>
</main>
```

```css
p:nth-child(2) {
  color:red;
}

p:nth-of-type(2) {
  background-color:green;
}
```

# Children & types

**`: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**

**parent**

```
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 }
```

# Children & types

**`: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*

siblings

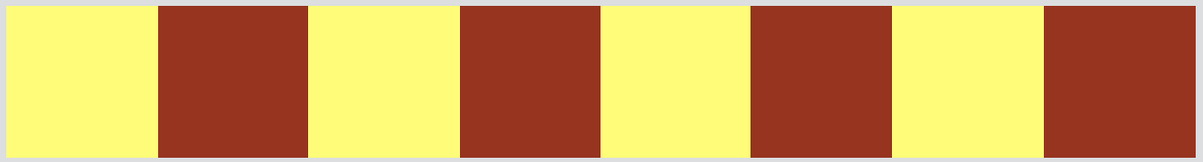
```
1 <main>
2   <h2>Todos</h2>
3   <p>Today’s todos</p>
4   <p>Tomorrow’s todos</p>
5   <p>Saturday’s todos</p>
6   <p>Sunday’s todos</p>
7 </main>
```

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

# Popular pseudo-classes

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

```html
<html>
    <head>
    <style>
    div {
        width: 20px;
        height: 20px;
        float: left;
    }
    div:nth-child(2n) {
        background-color: brown;
    }
    div:nth-child(2n+1){
        background-color: yellow;
    }
    </style>
</head>
<body>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
</body>
</html>
```



Create an Easychair instance

Send call for papers to mailing lists

Create a conference website

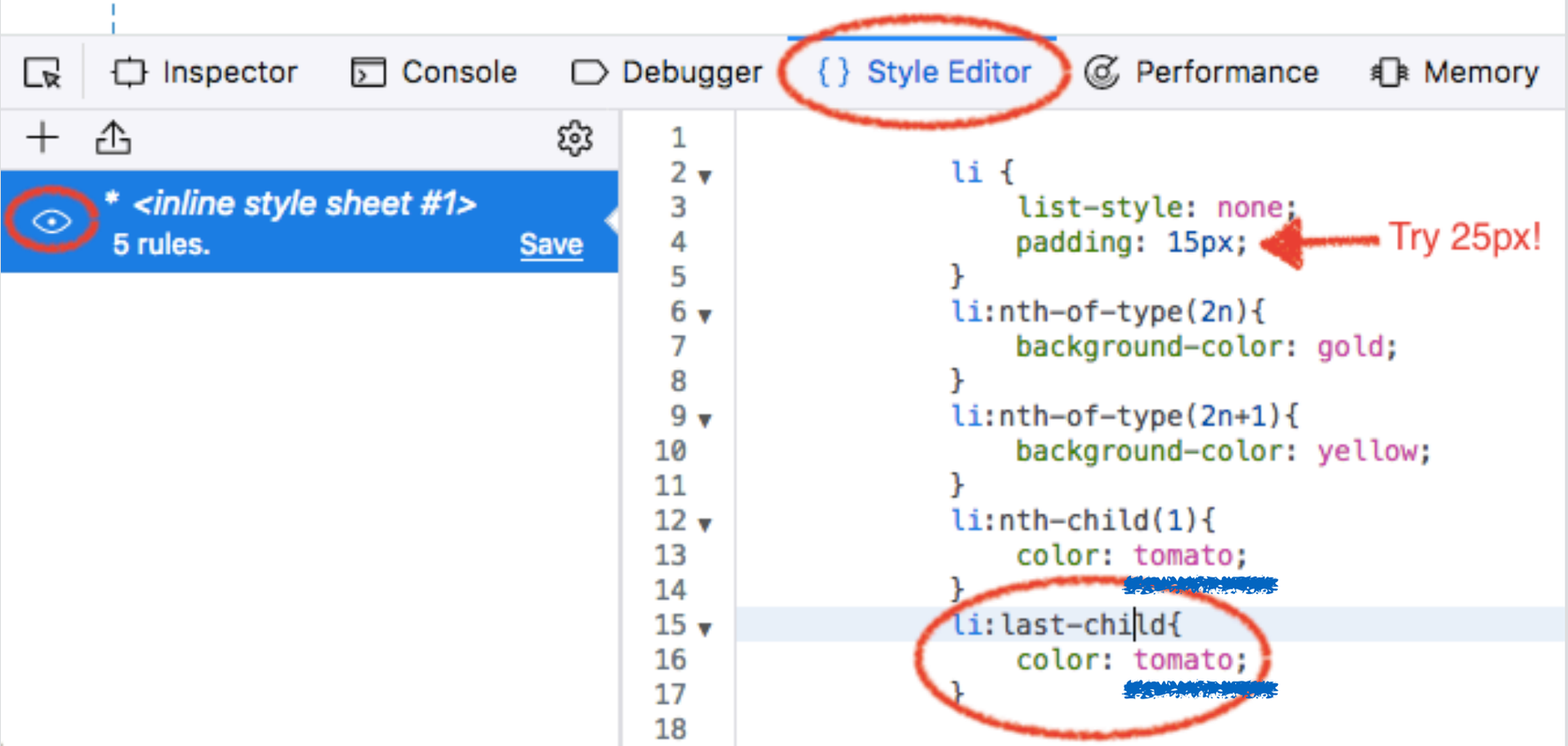
Book the venue

Find PC chairs

Find PC members

Book hotel rooms

Design program




Switch CSS files on and off

Mouse hover reveals rules' impact

Change CSS on the fly (not saved though!)

# CSS variables

*included in 2015-16 in major browsers*

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

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

*non-global CSS variables can be added everywhere*

```
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  :root {
2    --highlight:tomato;
3  }
4
5  p:nth-of-type(2n) {
6    color:var(--highlight);
7  }
```

# Hover & active

*Often employed in:*
- *image galeries*
- *dropdown menus*

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

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

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

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

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

# Enabled & disabled

Add Todo

**`:enabled`**
an element that can be clicked/selected

**`:disabled`**
an element that cannot be clicked/selected

Game project:
Not all tiles/buttons are available at all times (e.g. battleship: disable already discovered tiles)

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            function disable(el){
                document.getElementById(el.id).disabled = true;
            }
        </script>
        <style>
            button {
                background: white;
                color: darkgray;
                width: 100px;
                padding: 5px;
                font-weight: bold;
                text-align: center;
                border: 1px solid darkgray;
            }

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

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

            button:disabled {
                background: #ddd;
                color: #aaa;
                border: 1px solid #bbb;
            }
        </style>
    </head>
    <body>
        <main>
            <button id="b" onclick="disable(this)">Add Todo</button>
        </main>
    </body>
</html>
```

pseudo-classes can be combined (logical and)

# Not

**`: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 }
```


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

whitespace in selectors implies the universal selector: `*`

# Selector combinations

| Selector | Description |
|---|---|
| e1 | Selects all <e1> elements |
| e1 e2 | Selects all <e2> elements within <e1> |
| e1,e2 | Selects all <e1> elements and all <e2> elements |
| e1>e2 | Selects all <e2> elements that have <e1> as parent |
| e1+e2 | Selects all <e2> elements that follow <e1> immmediately |

# Valid & invalid

**`:valid :invalid`**

style `<input>` elements depending on their input

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

```
input[type=text] {
  border: 0px;
  width: 150px;
}
input[type=number] {
  width: 40px;
}
```

attribute selector

```
input[type=number]:valid+label::after {
  content:" \2714";
  color: rgba(0,100,0,0.7);
}

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

adjacent selector

unicode

rgb & alpha

# Pseudo-elements

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

# Two firsts

```
::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>
```

Relative font size changes

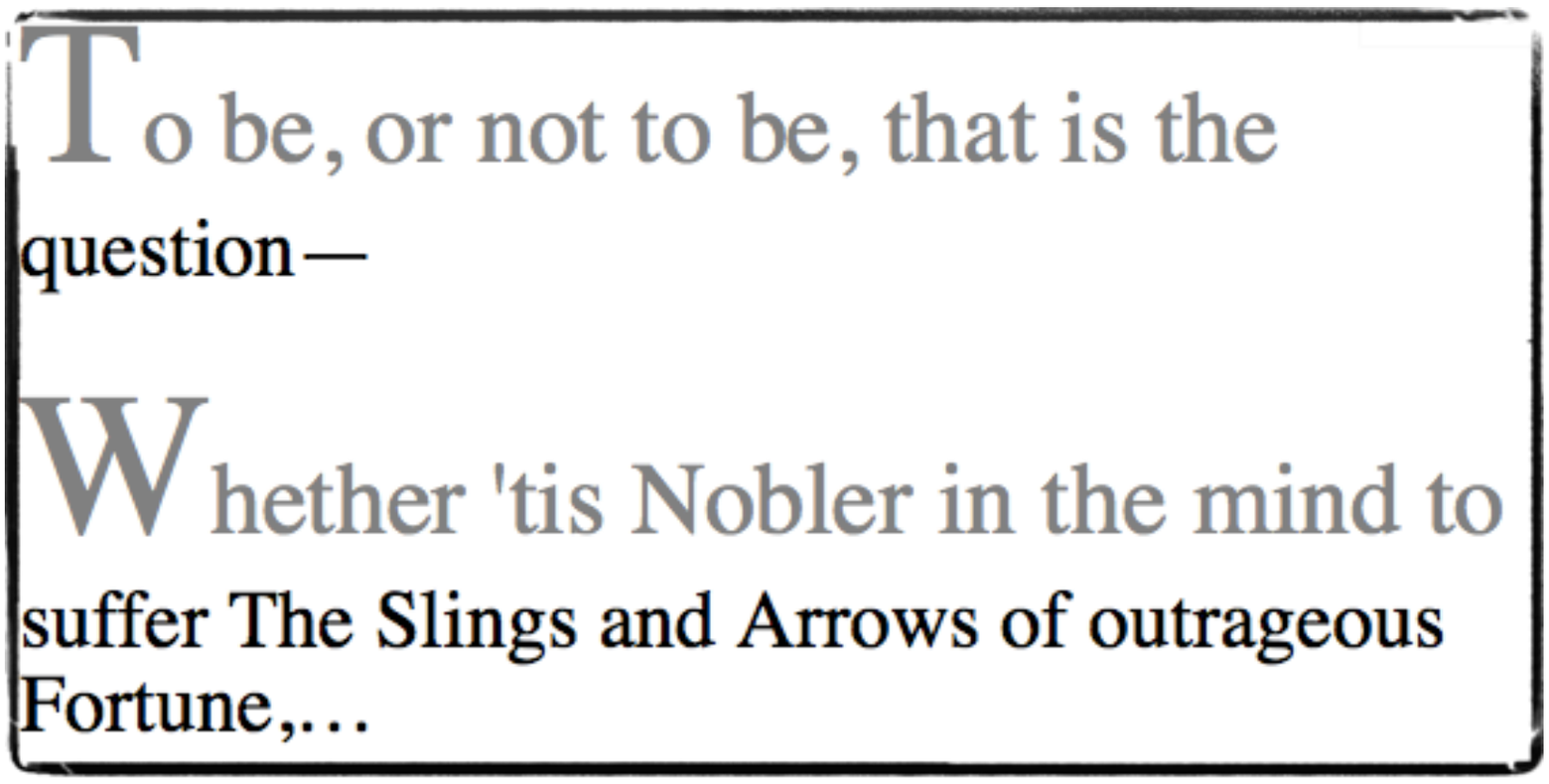
# Two firsts

```
::first-letter
::first-line
```

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

Browser resizing

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




Relative font size changes

# Before and after

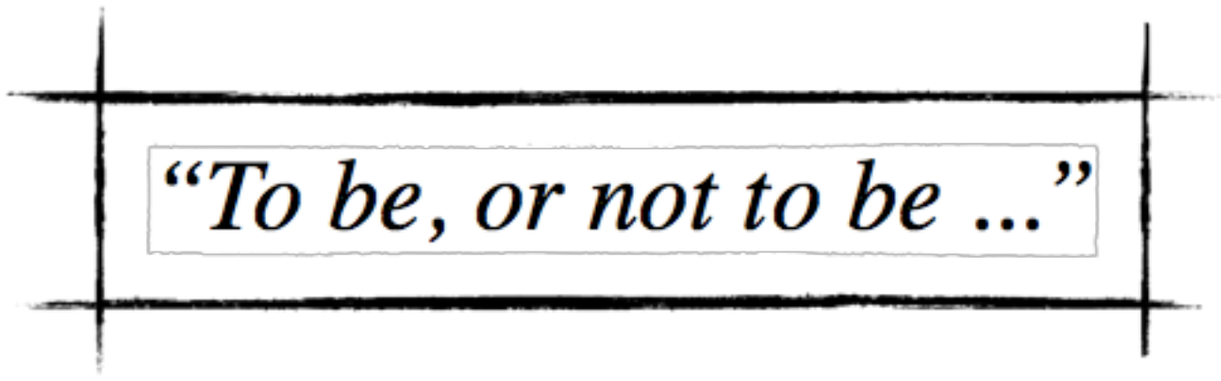
**`::before`** add (cosmetic) content before an element

**`::after`** add (cosmetic) content after 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

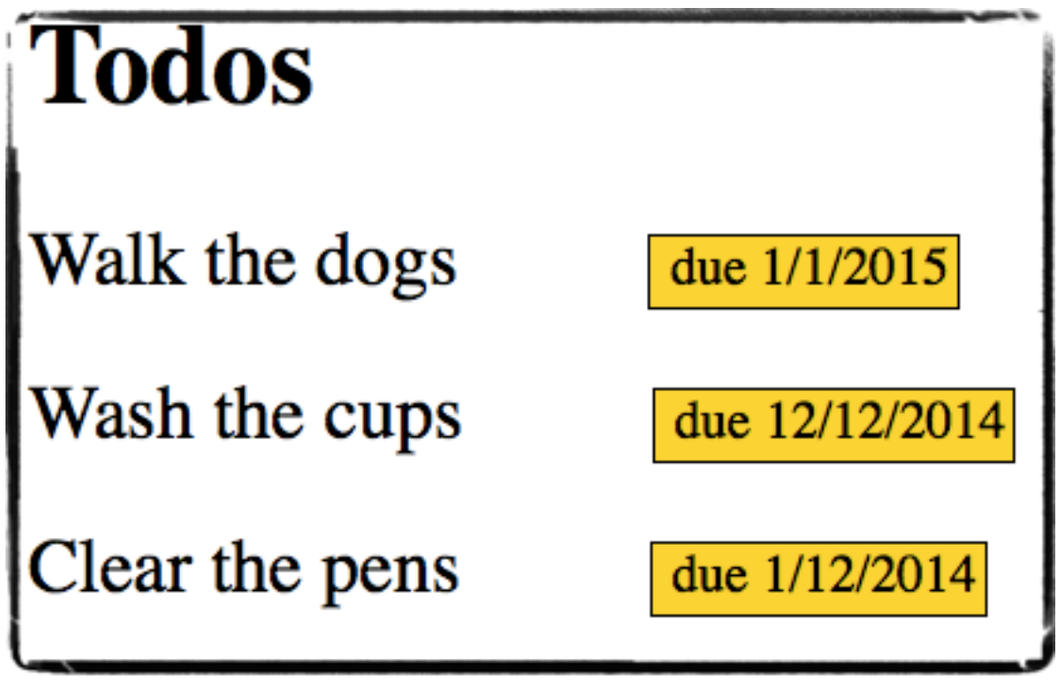


# Data in CSS

# 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 }
```

# CSS & data (one way)

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

```
<main>
  <h2>Todos</h2>
    <p id="t1">Walk the dogs</p>
```

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

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

Wash the cups due 12/12/2014

Clear the pens due 1/12/2014

**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)

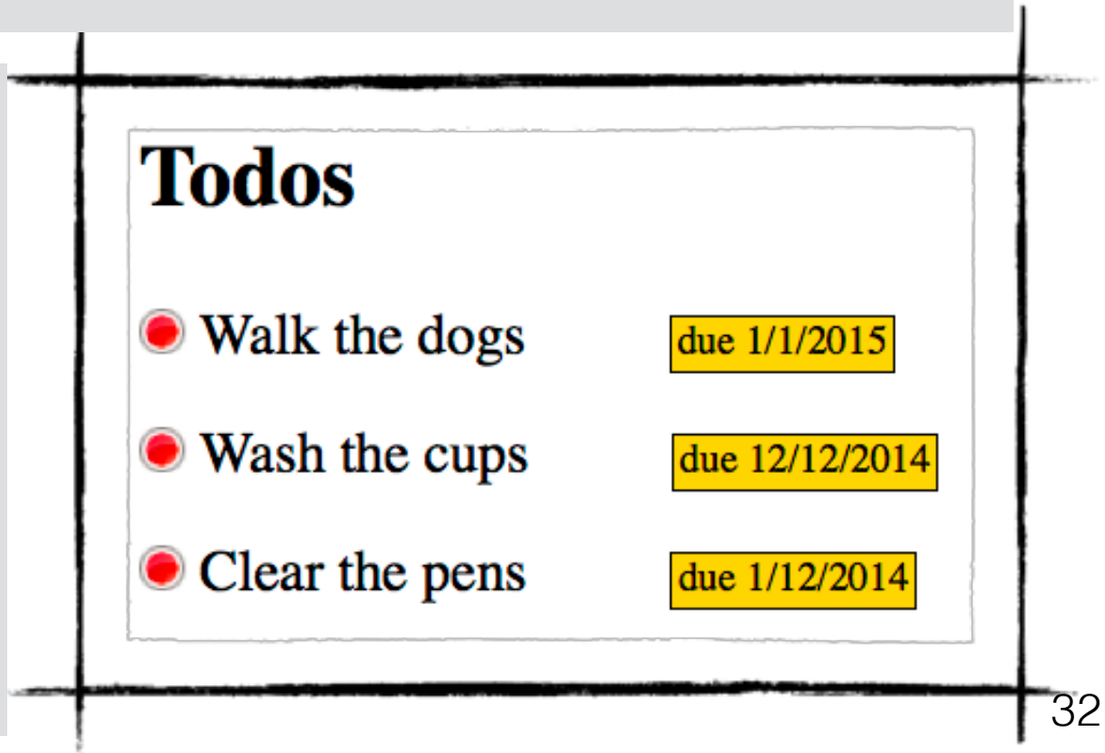
# CSS & data-* (the preferred way)

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

**Recall**: *valid* HTML elements can have `data-*` attributes.

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

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



# CSS & data-* (the preferred way)

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

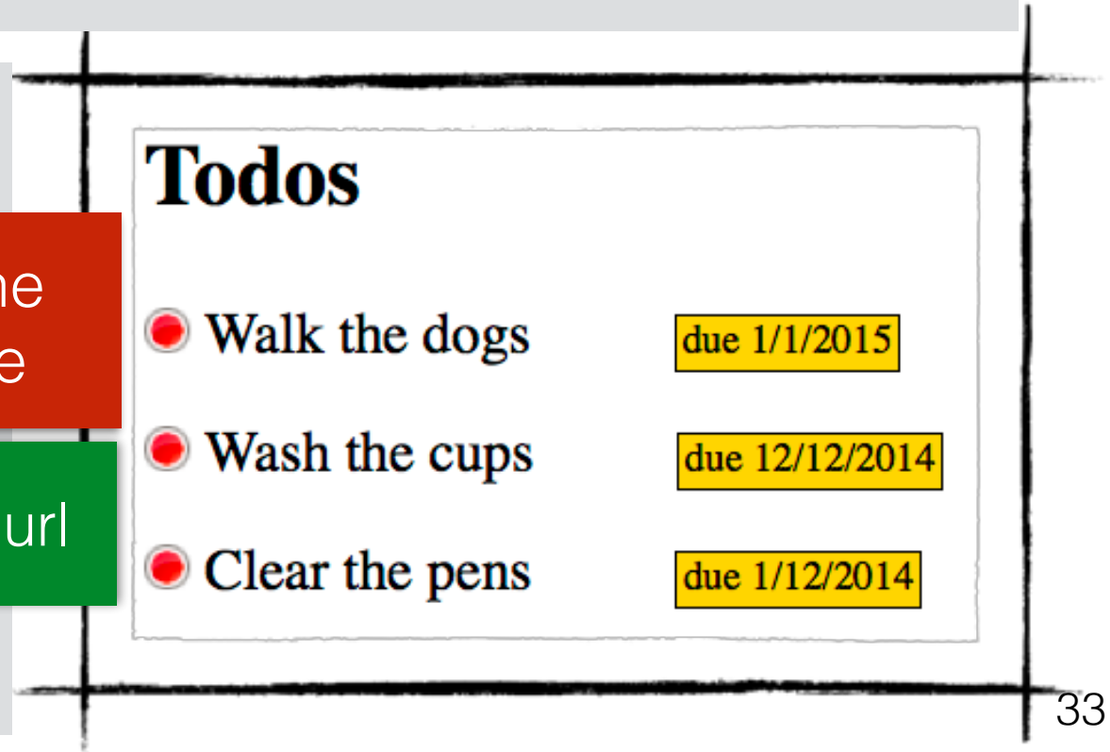
**Recall**: *valid* HTML elements can have `data-*` attributes.

```
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   cont
8 }
9 p::bef
10  content: url(http://www.abc.de/dot.png);
11 }
```

**`attr()`** retrieves the value of an attribute

content attribute can also reference a url



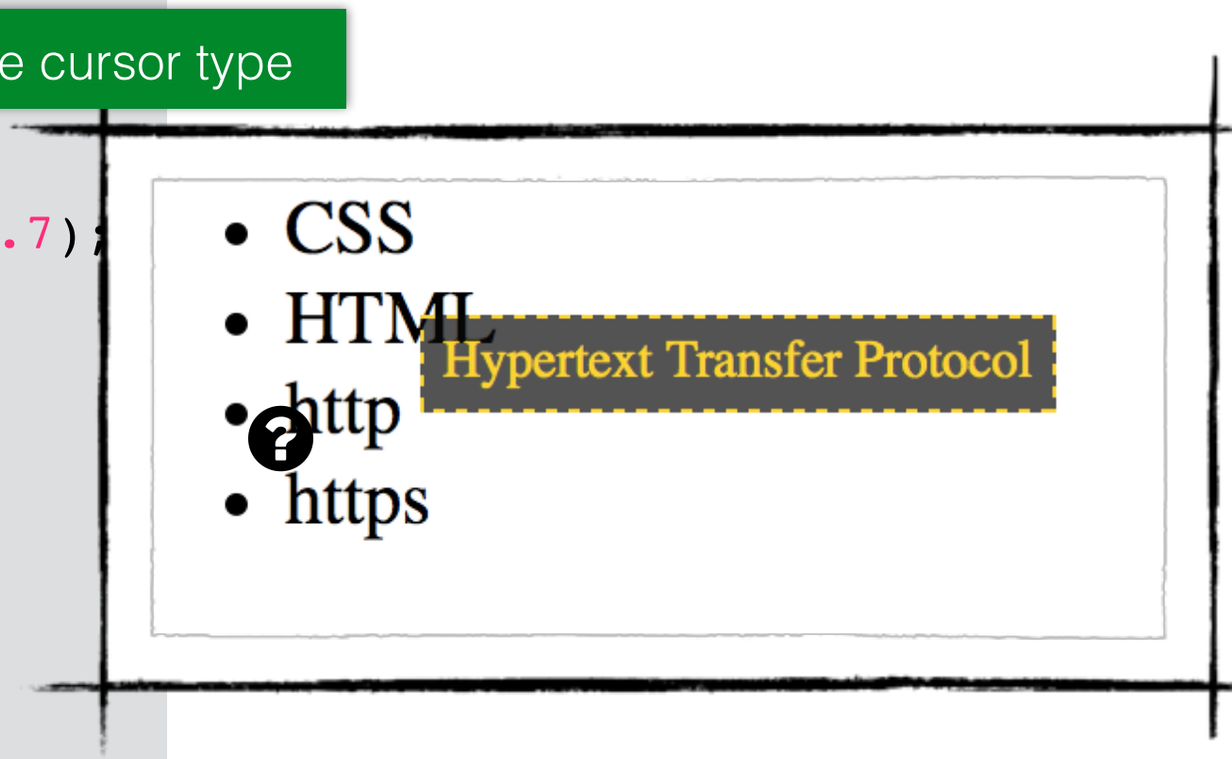
# CSS & data-* (the preferred way)

## A simple tooltip

```html
<ul>
 <li data-name="Cascading Style Sheets">CSS</li>
 <li data-name="HyperText Markup Language">HTML</li>
 <li data-name="Hypertext Transfer Protocol">http</li>
 <li data-name="Hypertext Transfer Protocol Secure">https</li>
</ul>
```

```css
li {
     cursor:help;
}
li:hover::after {
    background-color:rgba(10,10,10,0.7);
    color: gold;
    border: 1px dashed;
    padding: 5px;
    font-size: 70%;
    content: attr(data-name);
    position:relative;
    bottom:15px;
    left:5px;
}
```

we can change the cursor type



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

# Overview

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

# Elements “flow” by default

HTML 4 terminology

**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. `<span>` or `<a>`

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

# Elements “flow” by default

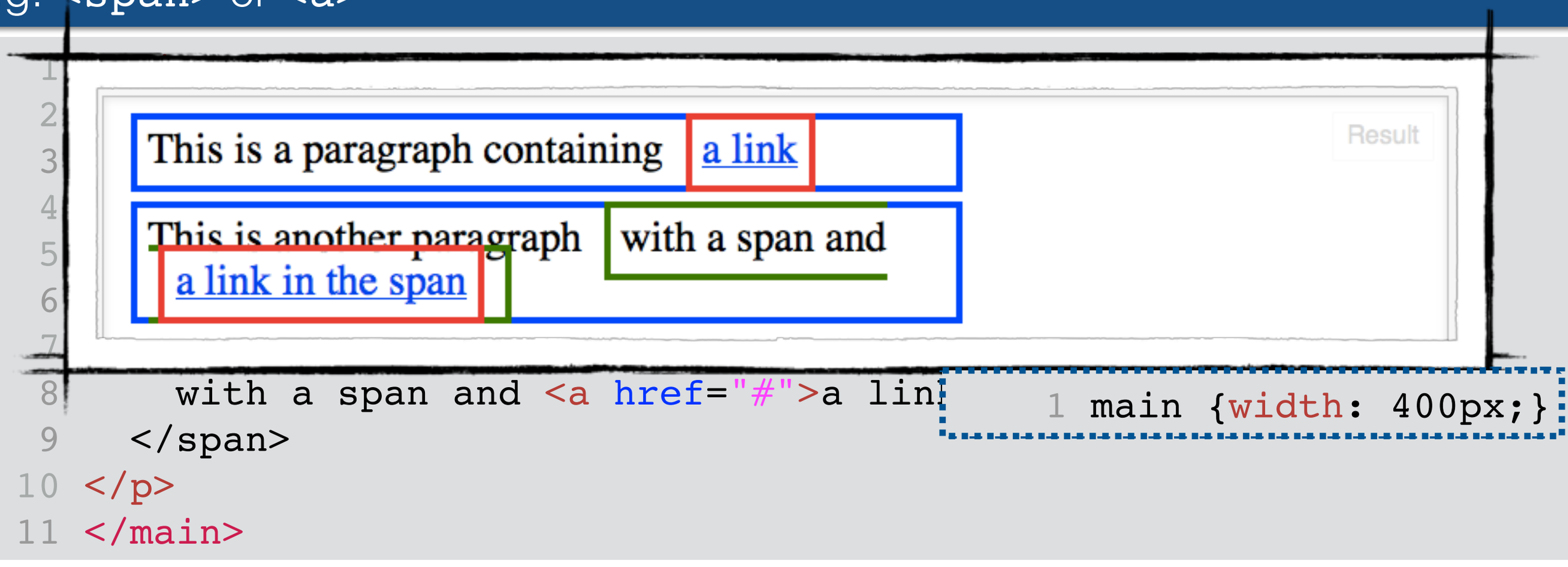
HTML 4 terminology

**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. `<span>` or `<a>`



# Elements “flow” by default

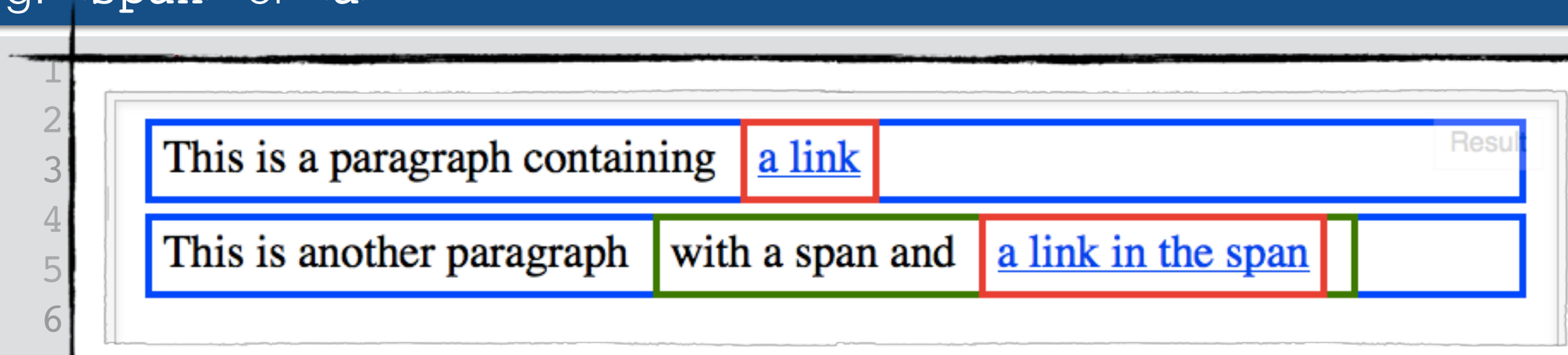
HTML 4 terminology

**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. `<span>` or `<a>`

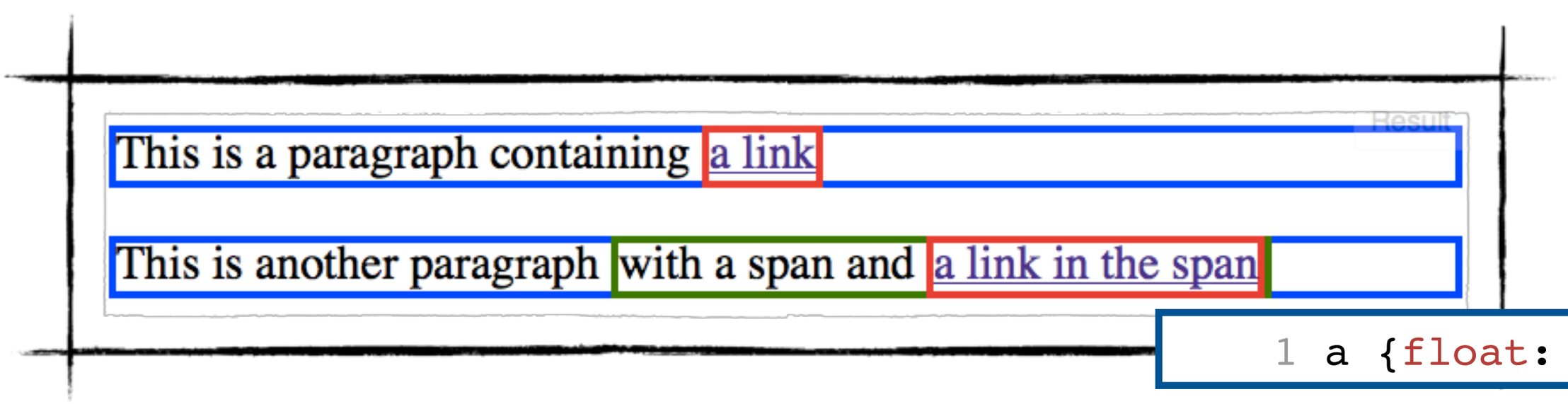



```
<p>
  This is another paragraph
    with a span and <a href="#">a link
  </span>
</p>
</main>
```

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

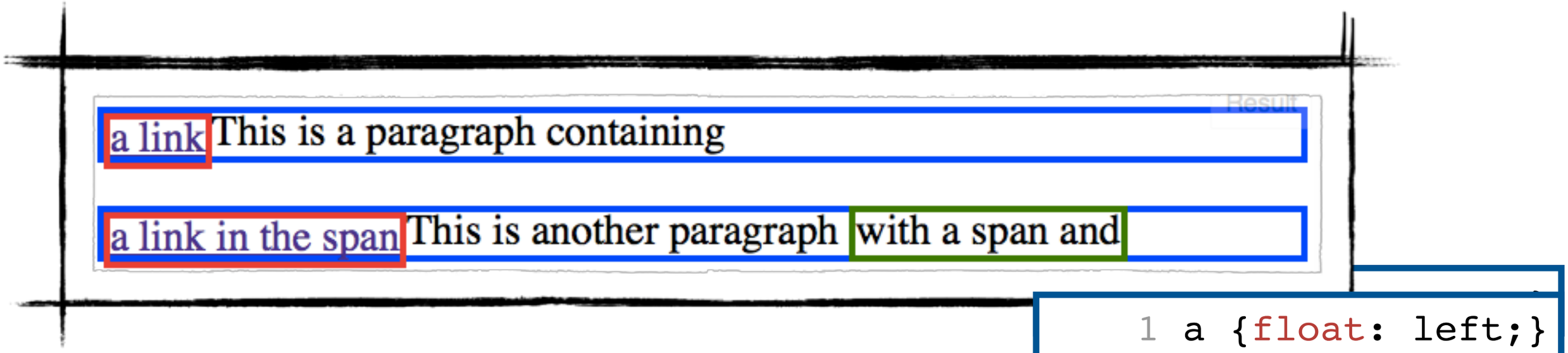
# Taking elements out of the flow

**`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.



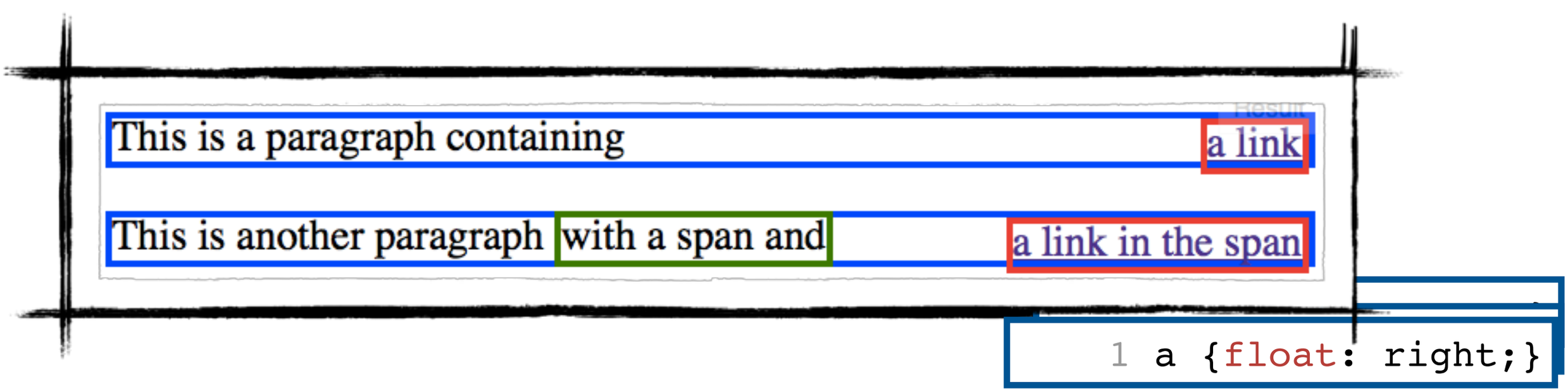
# Taking elements out of the flow

**`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.



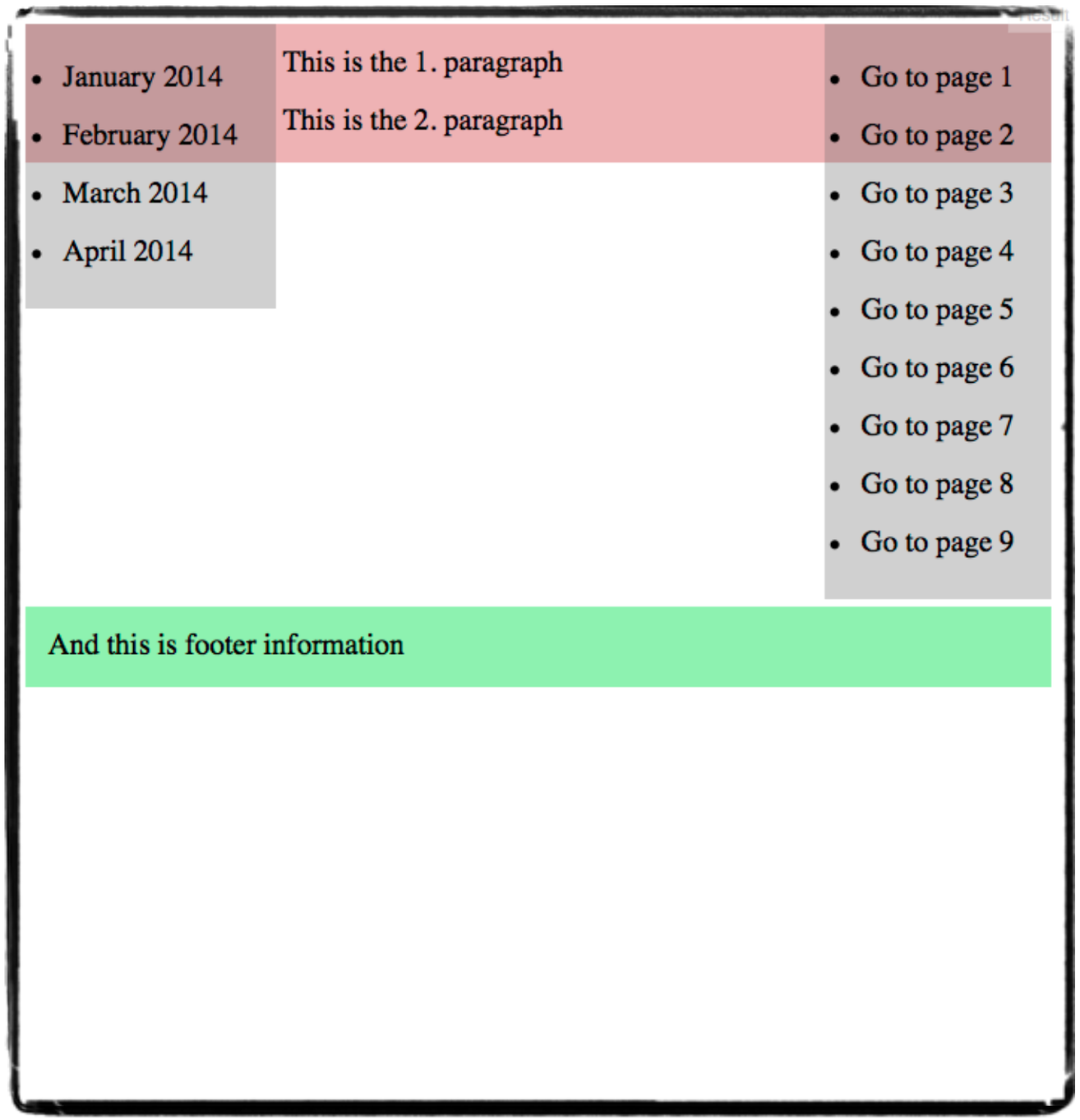
# Taking elements out of the flow

**`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.



# 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**

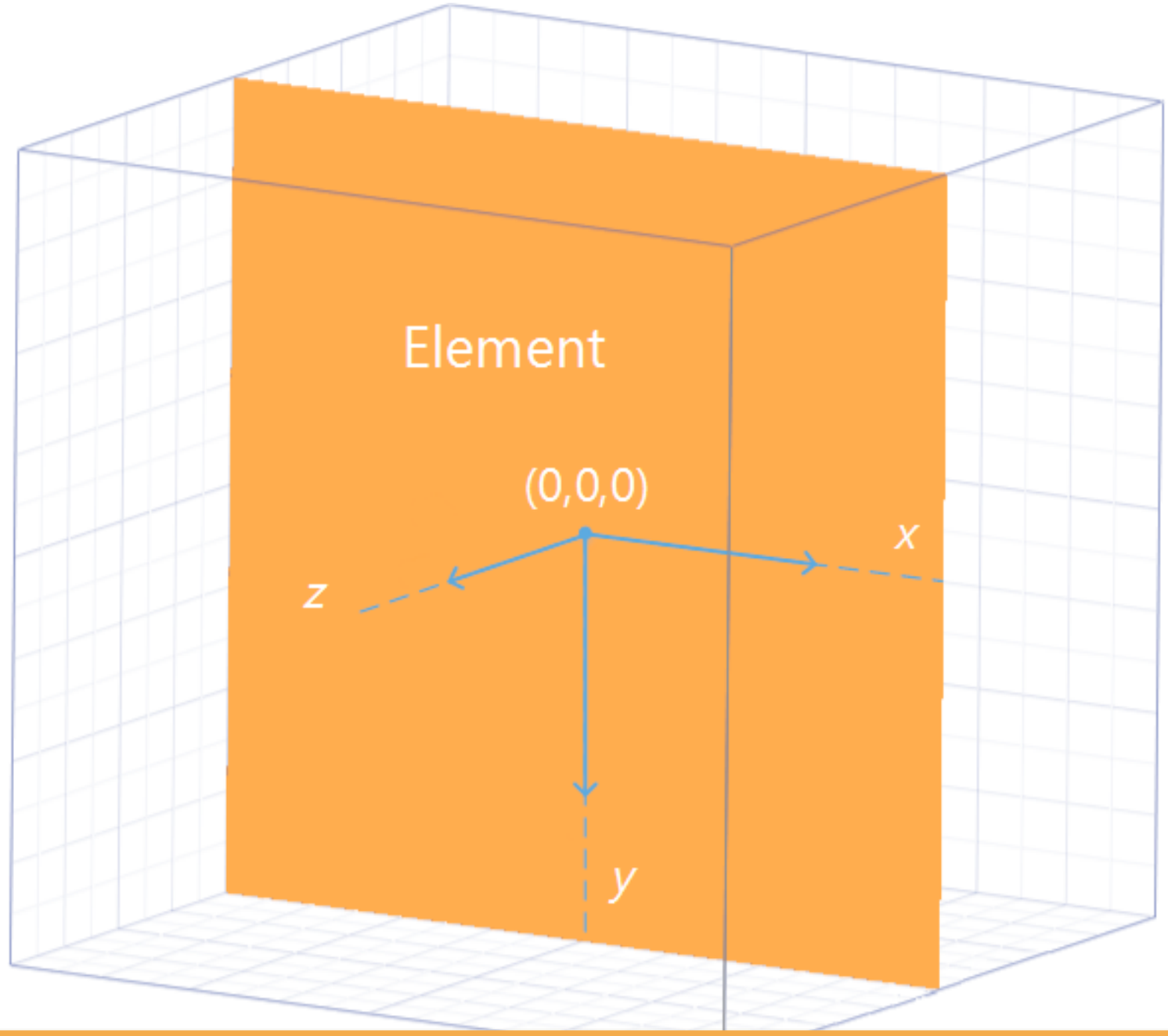
# Fine-grained movement of elements: `position`

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

| | |
|---|---|
| **`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 |

the area currently being viewed

# CSS coordinate system




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

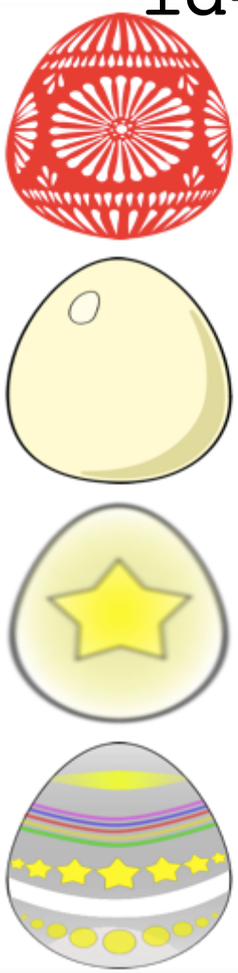
Image source: http://msdn.microsoft.com/en-us/library/ie/jj665791%28v=vs.85%29.aspx

# position:relative

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

movement is **relative** to its original position

id="egg1"

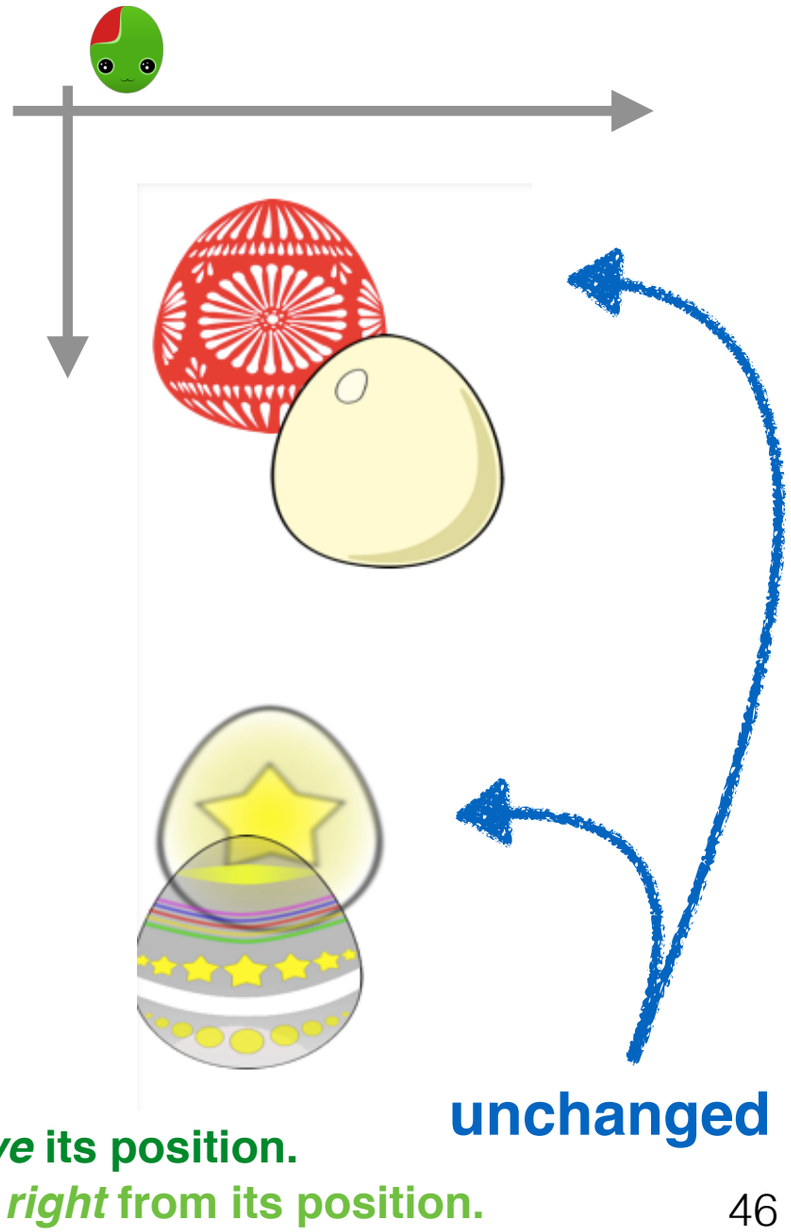


id="egg4"

```
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 }
```

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

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

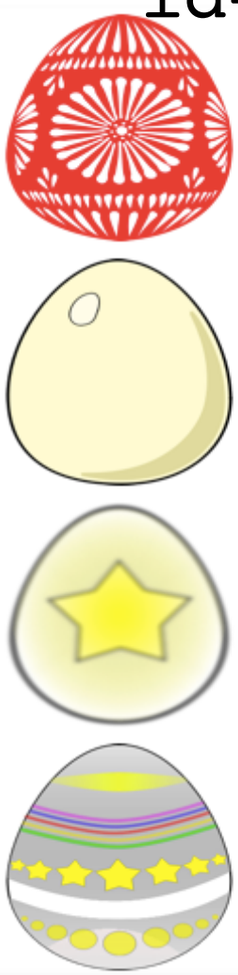


# position:absolute

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

positioning is relative to nearest ancestor or the window

id="egg1"

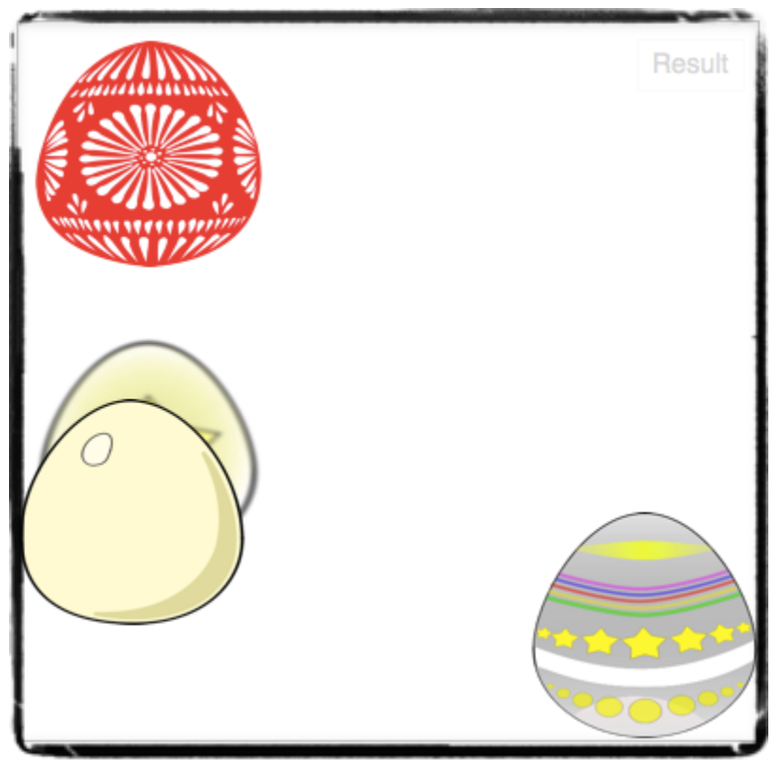


id="egg4"

```
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 }
```

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

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



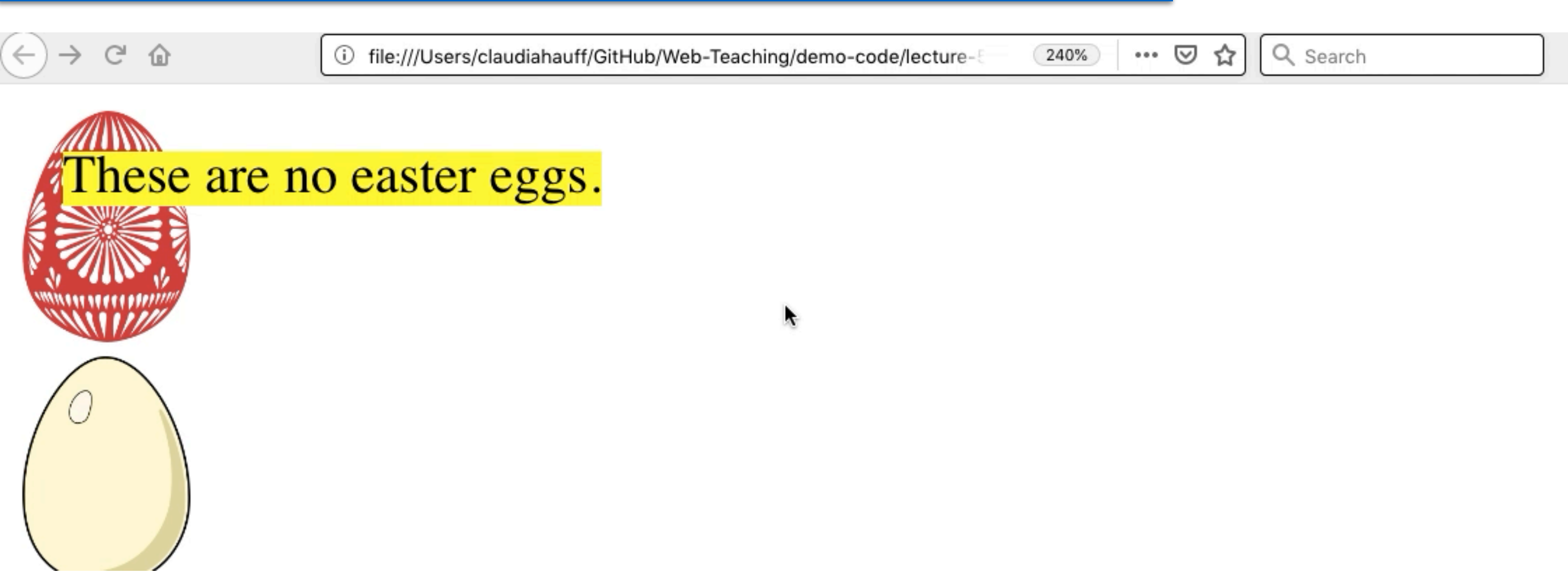

window

# `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



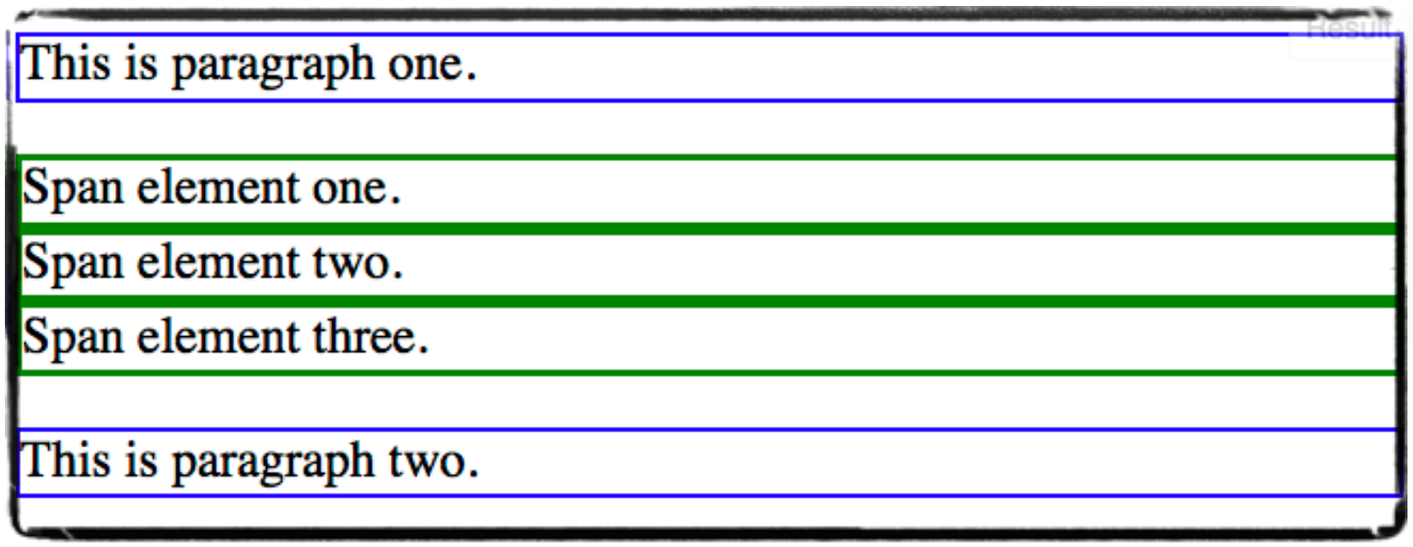
# display

**`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; }
```

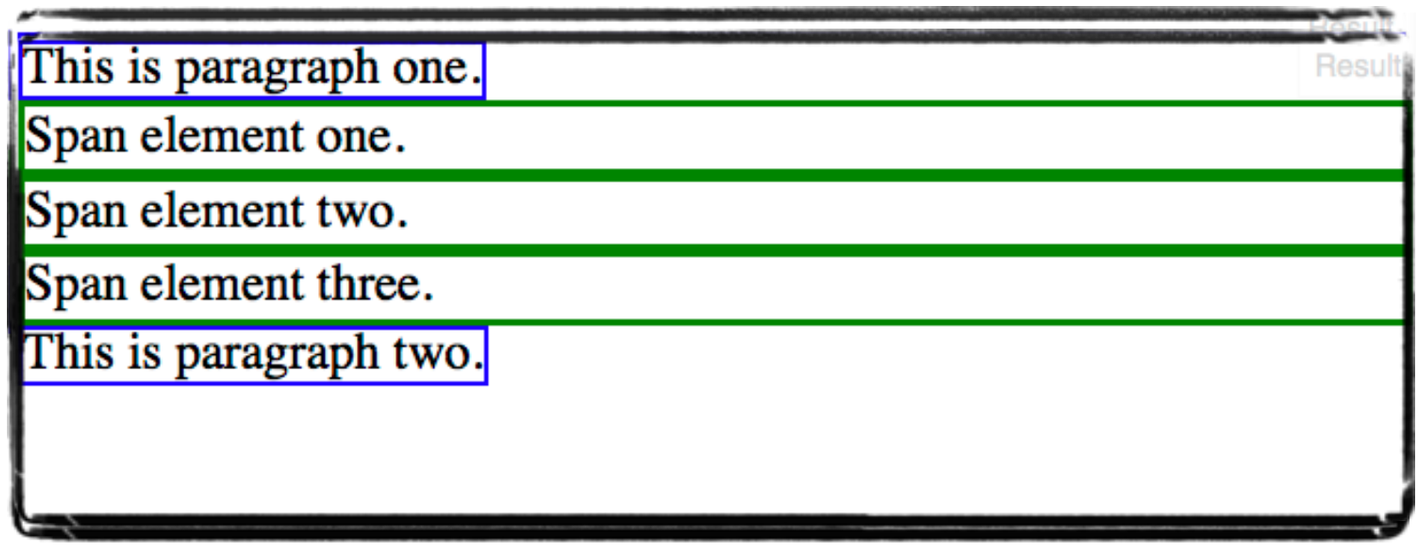
# display

**`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;}
```

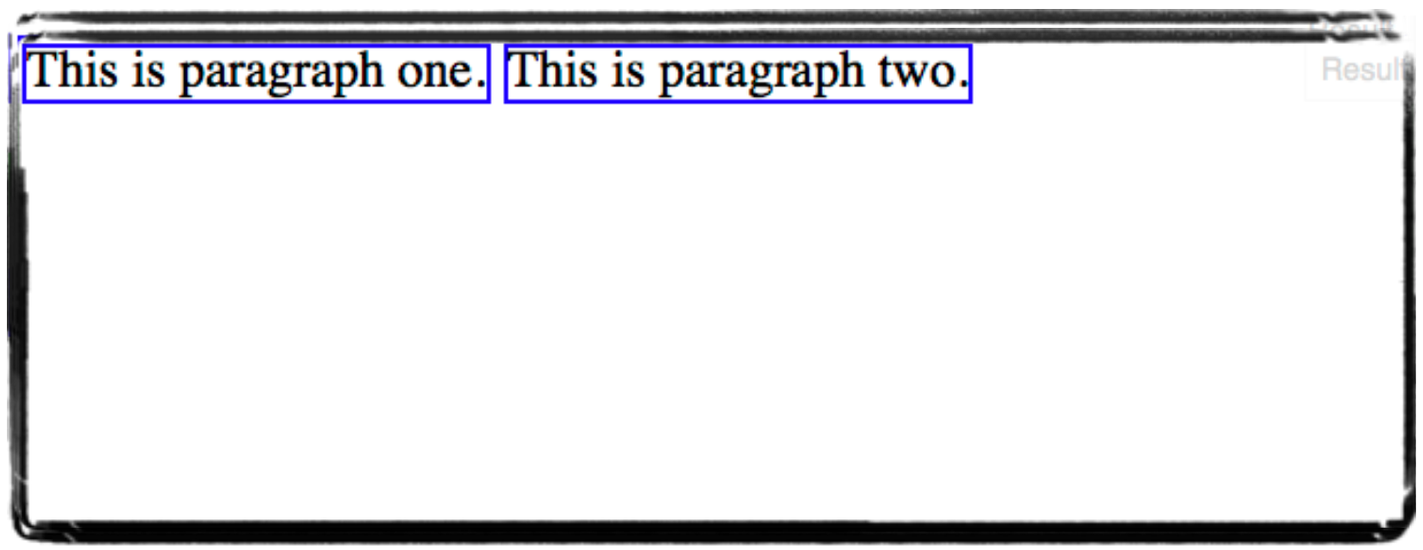
# display

**`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…

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

HTML: write once

CSS: write once per device

# Four device types

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

at-rules starting with @

responsive design mode

# Media queries can be complex

```
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

# Rendering engines

(layout engine, browser engine)

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

# In general …

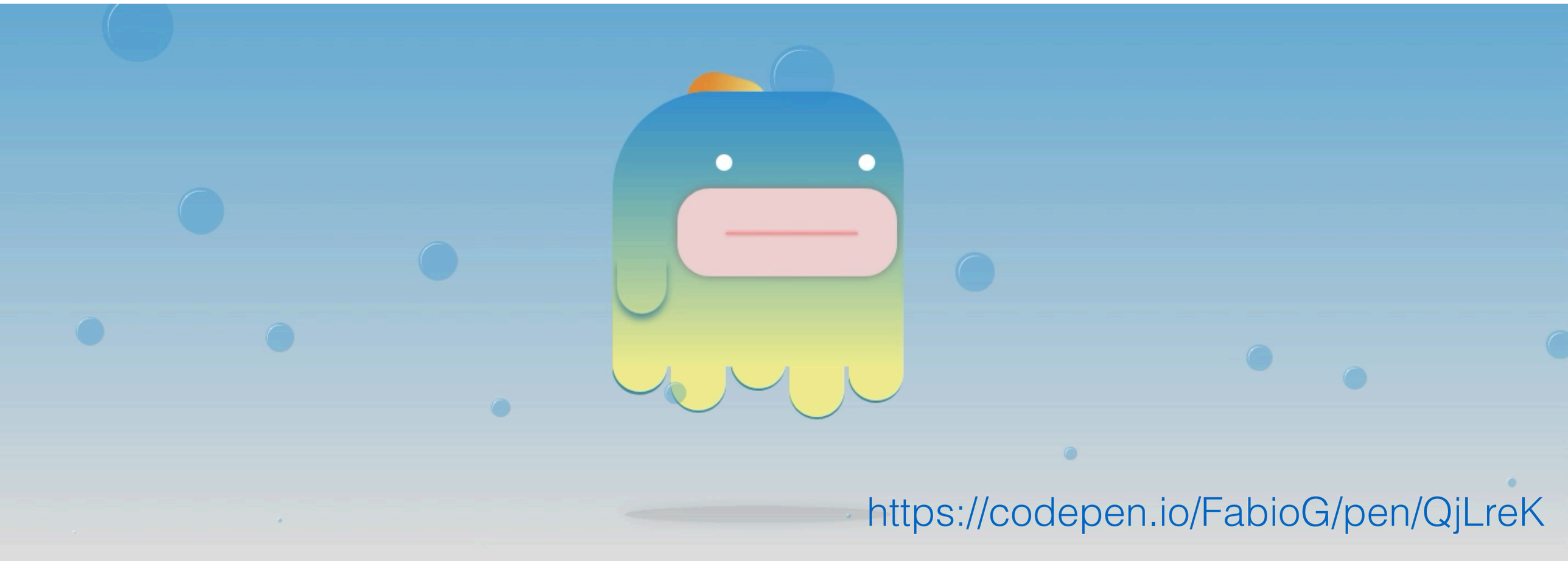
- CSS styles (states) are defined by the developer, the rendering engine takes care of the transition

- **Animations** consist of:
  - an animation **style** (`linear`, etc.)
  - **keyframes** acting as transition waypoints

- **Transitions** are animations that consist of exactly two states: start and end state

# CSS vs. JavaScript animations

- **Easy to use** and debug
- Rendering engines are **optimised** for CSS-based animations
- CSS animations can do more than animating buttons



https://codepen.io/FabioG/pen/QjLreK

TU

DELFT

```
<span class="flicker-slow">D</span>
```

```css
body{
  background: black;
}

.neon{
  font-family: 'Monoton', cursive;
  font-size: 150px;
  color: #00A6D6;
  text-shadow: 10px 10px 10px #E64616, 5px 5px 60px #A5CA1A;
}

.flicker-slow{
  animation: flicker 3s linear infinite;
}

/*
.flicker-slow{
  animation-name: flicker;
  animation-duration: 3s;
  animation-timing-function: linear;
  animation-iteration-count: infinite;
}
*/

.flicker-fast{
  animation: flicker 1s linear infinite;
}

@keyframes flicker {
  0%, 30%, 33%, 55%, 100% {
    opacity: .99;
  }

  31%, 50%, 56% {
    opacity: 0.3;
  }
}
```

**animation name (@keyframes)**

**intermediate states**

instead of 0% and 100% we can also use from and to

```css
.box {
  width: auto;
  height: 100px;
  background-color: #00a6d6;
  margin: 0 auto;
  float: left;
  font-size: 100px;
  padding: 20px;
  color: white;
  text-shadow: 3px 3px 2px black;
}

@keyframes move {
  from {
    transform: translate(0px, 1000px)
  }
  to {
    transform: translate(0px, 0px)
  }
}

#box1, #box7 {
  animation: move 1s;
}

#box2, #box8 {
  animation: move 4s;
}
```

T U D

elements are moved out of view

moving at different speeds

# CSS animation control

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

# 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;
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.

- Work on **Assignment 2**.

- Start working on **Assignment 3**.

- Check the werkcollege materials.  Are these exercises for you?

flickr@courtneyrian