COMP519 Web Programming
Lecture 8: Cascading Style Sheets: Part 4
Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool
Contents

1 Layout Via Divisions
   Relevant Properties

2 CSS Grid Layout
   Overview
   Defining a Grid Layout
   Placing Elements on a Grid
   Example

3 CSS Flexbox Layout
   Overview
   Defining a Flexbox Layout
   Example

4 Adaptive versus Responsive Design

5 Further Reading
For a long time, web page layout was based on the extensive use of `div` elements.

A web page would typically consist of roughly a handful of `div` elements as follows:

```
<div id="header"> ... </div>
<div id="nav"> ... </div>
<div id="main">
  <div id="content"> ... </div>
  <div id="ads"> ... </div>
</div>
<div id="footer"> ... </div>
```

possibly with additional `div` elements inside each of those

Layout is then a matter of arranging those `div` elements.

Decisions on layout are a matter of design, not of technical possibility.

→ there is typically not one right answer

→ this is not a topic for this module (web programming vs web design)
Divisions and Properties (1)

- By default, a `div` element takes up the whole width of a browser window and there is a line break before and after it. Changes almost always need to be made to achieve the desired layout.

- **CSS properties** that we can use to make those changes include:

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation / Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Width of an element</td>
</tr>
<tr>
<td></td>
<td>1000px</td>
</tr>
<tr>
<td></td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>90% of the width of the containing element</td>
</tr>
<tr>
<td>height</td>
<td>Height of an element</td>
</tr>
<tr>
<td></td>
<td>200px</td>
</tr>
<tr>
<td></td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>10% of the height of the containing element</td>
</tr>
<tr>
<td>margin</td>
<td>All four margins of an element</td>
</tr>
<tr>
<td></td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>centre horizontally within containing element</td>
</tr>
</tbody>
</table>
Divisions and Properties (1)

- By default, a `div` element takes up the whole width of a browser window and there is a line break before and after it.
  - Changes almost always need to be made to achieve the desired layout.
- **CSS properties** that we can use to make those changes include:

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation / Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>float</code></td>
<td>Whether and in which direction an element should float</td>
</tr>
<tr>
<td></td>
<td><code>left</code> element floats to the left of its container</td>
</tr>
<tr>
<td></td>
<td><code>right</code> element floats to the right of its container</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>Whether and how an element must be (cleared) below floating elements</td>
</tr>
<tr>
<td></td>
<td><code>left</code> element moves down to clear past left floats</td>
</tr>
<tr>
<td></td>
<td><code>right</code> element moves down to clear past right floats</td>
</tr>
<tr>
<td></td>
<td><code>both</code> element moves down to clear past all floats</td>
</tr>
</tbody>
</table>
A common layout of the top-level div elements is the following

```html
<body>
  <div id="header">
    header
  </div>
  <div id="nav">
    nav
  </div>
  <div id="main">
    <div id="content">
      content
    </div>
    <div id="ads">
      ads
    </div>
  </div>
  <div id="footer">
    footer
  </div>
</body>
```

with the width of header, nav, main, footer fixed to a value between 900px and 1000px
A common layout of the top-level div elements is the following

```css
#header { width: 1000px;
height: 100px;
background-color: blue;
margin: auto; }

#nav { width: 1000px;
height: 50px;
background-color: green;
margin: auto; }

#main { width: 1000px;
margin: auto; }

#content { width: 800px;
height: 400px;
background-color: yellow;
float: left; }

#ads { width: 200px;
height: 400px;
background-color: orange;
float: right; }

#footer { width: 1000px;
height: 50px;
clear: both; margin: auto;
background-color: blue; }
```

with the width of header, nav, main, footer fixed to a value between 900px and 1000px
Layout Via HTML5 Elements

- In the example, we assigned unique a `id` to each `div` element and associated a style directive with each of those `ids`.
- Alternatively, we could have assigned a unique `class` to each `div` element and associated a style directive with each of those `classes`.
- In HTML5, we would use the appropriate `elements` like `header`, `nav`, etc instead of `div` elements.
- We would then associate a style directive with each of those elements.

```html
<header>
  width: 1000px;
  height: 100px;
  background-color: blue;
  margin: auto;
</header>

<nav>
  width: 1000px;
  height: 50px;
  background-color: green;
  margin: auto;
</nav>
```
Fixed Positioning (1)

- So far, we have positioned elements relative to each other.
- This means the arrangements of elements as a whole can move and can move out of view if the user scrolls up or down in a browser window.
- **CSS properties** that we can use to change that include:

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation / Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Specifies how an element is positioned in a document</td>
</tr>
<tr>
<td></td>
<td><strong>fixed</strong> The element is removed from the normal document flow; no space is created for the element in the page layout; it is positioned relative to the screen’s viewport using properties top, bottom, left, right and does not move when scrolled</td>
</tr>
</tbody>
</table>
## Fixed Positioning (2)

- **CSS properties** required for `position` include:

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation / Example values</th>
</tr>
</thead>
</table>
| `top`    | When `position` is set to `absolute` or `fixed`, specifies the distance between the element’s top edge and the top edge of its containing block.  
  | `10px` 10px off top edge |
| `bottom` | Analogous to `top` for the element’s bottom edge and the bottom edge of its containing block.  
  | `20%` 20% of the width of the containing block |
| `left`   | Analogous to `top` for the element’s left edge and the left edge of its containing block.  
  | `auto` |
| `right`  | Analogous to `right` for the element’s left edge and the left edge of its containing block.  
  | `inherit` inherit from parent element |
Fixed Positioning: Example

We want to achieve the same layout as before but with header, nav and footer fixed in position.

We do so with slightly different approaches used in the style directives for each of these three elements.

```
header { width: 1000px;  
        height: 100px;   
        background-color: blue;  
        position: fixed; 
        top: 0px;       
        left: 50%;      
        margin-left: -500px; }

nav  { width: 1000px;       
      height: 50px;          
      background-color: green; 
      position: fixed;       
      top: 100px;            
      left: 0px;             
      right: 0px;            
      margin: auto;          }

article { width: 1000px;  
          padding-top: 142px; 
          margin: 0 auto;     }
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/layout2.html
Fixed Positioning: Example

We want to achieve the same layout as before but with header, nav and footer fixed in position

```html
section { width: 800px; height: 1000px; background-color: yellow; float: left; }

aside { width: 200px; height: 1000px; background-color: orange; float: right; }

footer { width: 1000px; height: 50px; background-color: blue; position: fixed; bottom: 0px; left: 50%; transform: translate(-50%,0%); }
```

We do so with slightly different approaches used in the style directives for each of these three elements.

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/layout2.html
Adaptive Design Revisited

- One fixed, rigid layout is unlikely to be suitable for every medium that a user might use to access an HTML document
- We have seen that the media-attribute of the link element allows us to select which external style sheets to use for which medium, e.g.

```html
<link rel="stylesheet" type="text/css" media="screen" href="browser.css">
<link rel="stylesheet" type="text/css" media="print" href="print.css">
```

- However, if the style directives in the these different style sheets are largely identical, this is not an optimal approach
  - the same style directives exist in several files, changes are error prone
- HTML5 provides three mechanisms to better deal with such a situation
  - Import rules
  - Media rules
  - Support rules
Importing CSS style files

• The `@import` CSS at-rule is used to import style directives from other style sheets

```css
@import url;
@import url list-of-media-queries;
```

Examples:
```css
@import url("http://cgi.csc.liv.ac.uk/styles/common.css");
@import "screen-specific.css" screen;
@import 'print-specific.css' print;
```

• These rules must precede all other types of rules and directives except `@charset` rules

• A `@charset` CSS at-rule specifies the character encoding used in a style sheet, for example:

```css
@charset "utf-8";
```

• The default character encoding is UTF-8
• Useful / used when attributes like `content` are given values involving non-ASCII characters
**Media Rules and Media Queries**

- Within a style sheet, `@media` at-rules can be used to conditionally apply styles to a document depending on the result of media queries.

```
@media list-of-media-queries { group-rule-body }
```

where `group-rule-body` is either another `@media` at-rule, `@supports` at-rule, or list of style directives.

**Examples:**

```
@media print {
  body { font-size: 10pt; }
}
@media screen and (resolution > 150dpi) {
  body { font-size: 13px; }
}
```

- The language for media queries is an extension of the one we have seen for the `media` attribute.
Feature Queries

- Within a style sheet, `@support` at-rules can be used to conditionally apply styles to a document depending on the result of feature queries.

  ```css
  @supports feature-query { group-rule-body }
  ```

- A feature query is basically a boolean combination (using `and`, `or`, `not`) of `property:value` pairs.

- For each `property:value` it will be evaluated whether the browser used to process the style sheet supports this specific CSS feature and then works out the truth value for the feature query overall.

Examples:

```css
@supports (display: flex) {
  div { display: flex; }
}
@supports not(display: flex) {
  div { float: left; }
}
```
CSS Grid Layout

- **CSS Grid Layout** is a two-dimensional grid-based layout system.
- Such layout systems are motivated by the observation that most web layouts can be seen as grids where elements are placed on one or more grid cells.
- Height and width of grid columns and grid rows will in general vary.

**Sample page layout**

**Underlying grid and allocation of elements to grid cells**

<table>
<thead>
<tr>
<th></th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td></td>
<td>header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 2</td>
<td></td>
<td>nav</td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 3</td>
<td>content</td>
<td></td>
<td>ads</td>
<td></td>
</tr>
<tr>
<td>row 4</td>
<td></td>
<td>footer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Defining a Grid Layout

CSS properties of Grid include

- **display: grid**
  defines an element as grid container

- **grid-template-columns: track-size | [col-name] track-size ...**
  specifies the size and names of columns

- **grid-template-rows: track-size | [row-name] track-size ...**
  specifies the size and names of rows

- **track-size** can be auto, a length, a percentage, or a fraction of the free space

These properties allow to specify a grid, including the size of each column and each row
Placing Elements on a Grid

One way to place an element on the grid is to specify
– in which column/row it starts (top, left corner) and
– in which column/row it ends (bottom, right corner)
using the following properties

- `grid-column-start: cell`
- `grid-column-end: cell`
- `grid-row-start: cell`
- `grid-row-end: cell`

- `cell` can take the following values:
  - `number` column number / row number
  - `name` name of a column / row
  - `span number` number of tracks covered
  - `span name` span until `name` is reached
  - `auto` automatic
Placing Elements on a Grid

An alternative way to place elements on the grid is to assign grid names to the elements and to use a grid template that references those names:

- **grid-area:** `area-name`
  assign a grid area name to an element

- **grid-template-areas:** "`area-name` | . | none | ..."
  associates grid area names with grid cells

This is only a glimpse of the possibilities of the CSS Grid Layout System.
CSS Grid Layout Example

We want to replicate the same layout as before:

```html
<article>
  <header> </header>
  <nav> </nav>
  <section> </section>
  <aside> </aside>
  <footer> </footer>
</article>

article { display: grid;
  grid-template-rows: 100px 50px auto 50px;
  grid-template-columns: auto 800px 200px auto;
  grid-template-areas: " . header header ."
                        " . nav nav ."
                        " . content ads ."
                        " . footer footer .";
}

header { grid-area: header; background-color: blue; }
nav { grid-area: nav; background-color: green; }
section { grid-area: content; background-color: yellow; }
aside { grid-area: ads; background-color: orange; }
footer { grid-area: footer; background-color: blue; }
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/layout3.html
CSS Flexbox Layout

- **CSS Flexbox Layout** is a simpler layout system, typically used for parts of a web page, not the whole page.
- Flexbox distinguishes between flex containers and flex items within those containers.
- Unlike Grid, Flexbox distinguishes between a primary **main axis** and a secondary **cross axis**.
- The **main axis** is not necessarily horizontal, its direction is determined by **flex-direction**.
Defining a Flexbox Layout

CSS properties of Flexbox include

- **display: flex**
  defines an element as a flexbox container

- **flex-direction:** row | row-reverse | column | column-reverse
  defines the direction of the main axis,
  for example, with row the direction is left to right (horizontally)

- **flex-wrap:** nowrap | wrap | wrap-reverse
  whether and how flex items wrap when the main axis is ‘full’,
  for example, with wrap-reverse, flex items will wrap onto multiple
  ‘lines’ from bottom to top along the cross axis

- **flex-flow:** direction-option || wrap-option
  combines flex-direction and flex-wrap
Flexbox Layout Properties

CSS properties of Flexbox include

- **justify-content**: *justify-option*
  defines the alignment along the main axis

*justify-option* can take the following values:

- **flex-start**
- **flex-end**
- **center**
- **space-between**
- **space-around**
- **space-evenly**

Flexbox Layout Properties

CSS properties of Flexbox include

- **align-content:** `align-option`
defines the use of extra space along the cross axis

`align-option` can take the following values:

CSS Flexbox Layout: Example

HTML

```html
<nav>
  <a href="#">Computer Science</a>
  <a href="#">Electrical Engineering and Electronics</a>
  <a href="#">Physics</a>
</nav>
```

CSS

```css
a { text-align: center }
nav {
  background-color: LightGreen;
  display: flex;
  flex-direction: row;
  justify-content: space-around;
}

/* Narrow screen width */
@media all and (max-width: 900px) {
  nav {
    flex-direction: column;
  }
}
```

Width = 1000px

- Computer Science
- Electrical Engineering and Electronics
- Physics

Width = 900px

- Computer Science
- Electrical Engineering and Electronics
- Physics

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/layout4.html
Adaptive versus Responsive Design

Adaptive Design
Uses a limited number of different web pages and/or different styles depending on media devices and media attributes

Responsive design
Uses a single web page and style that through the use of
• media queries,
• flexible grids,
• relative units and
• responsive images
tries to adjust to any media device with any media attributes at any time
Adaptive versus Responsive Design

Adaptive Design
Uses a limited number of different web pages and/or different styles depending on media devices and media attributes.

Responsive design
Uses a single web page and style that through the use of media queries, flexible grids, relative units and responsive images tries to adjust to any media device with any media attributes at any time.

There are no generally agreed definitions of adaptive design and responsive design.
It is often debatable whether a website uses adaptive design or responsive design (or neither).
There is even more debate which one is better.
Most/all of the examples we have seen use adaptive design, but this was done for effect.
Style Guide

• HTML and CSS provide a lot of features, but these must be used sensibly
  ↳ just because a feature exists does not mean it be used

• Do not use features that distract from the content of your web page

• Use (non-default) colours and fonts carefully
  ↳ no purple text on pink background
  ↳ no “weird” fonts (that includes Comic Sans)
  ↳ mainly use a dark font on a light background

• Remember that an estimated 8-10% of people have some type of colour-blindness
  ↳ avoid red/green colour combinations

• Remember that some people use screen readers to read the content of web pages
  ↳ always include alt properties for images
Style Guide

- Use **relative units** to specify font sizes, not fixed pixel sizes
- Use **images** appropriately
  - Avoid bright background images that make foreground text hard to read
  - Avoid clickable images instead of standard buttons for links as they can slow down the download of your page
- Do **not** rely on specific window size or specific font size for layout as the user might change those
  - Use an **adaptive or responsive design**
- Break a large web page into several smaller ones or provide a menu for navigation
- Utilise **style sheets** to make changes to style and layout easy and ensure consistency across a set of web pages
- Stick to standard features and test several browsers
Revision and Further Reading

Read

• Chapter 15: Floating and Positioning
• Chapter 16: CSS Layout with Flexbox and Grid

E-book https://library.liv.ac.uk/record=b5647021
Read

