Web Technologies — Week 5

Mikheil Rukhaia

International Black Sea University,
Tbilisi, Georgia

November 5, 2015

Outline

1. Positioning with CSS
2. Page Layout Strategies
3. Page Layout Techniques
4. Responsive Web Design
5. Laboratory Work

Positioning with CSS

Positioning schemes

- Normal – default scheme, block-level elements flowing from top to bottom and inline element from left to right.
- Relative – positions the element in relation to where it would otherwise sit in normal flow.
- Absolute – completely removes element's content from normal flow, allowing to fix its position relative to containing box.
- Float – take an element out of normal flow and place it as far to the left or right of a containing box as possible.

The `position` property

- To change the positioning scheme, use the `position` property with either values: `static` (default, normal flow), `relative`, `absolute`, `fixed`.
- Together with this property usually box offset properties are specified, which are `left`, `right`, `top` and `bottom`.

Relative Positioning

- Unless a background for a box is specified, it will be transparent by default, making any overlapping text an unreadable mess.
- Example:

```
p {
  border: 2px solid black;
  padding: 5px;
  background-color: white; /* necessary */
}
p.rel {
  position: relative;
  left: 40px;
  top: -40px;
}
```

Absolute positioning

- Absolutely positioned elements always come out above relatively positioned elements.
- Example:

```
p {
  border: 2px solid black;
  padding: 5px;
  background-color: white; /* necessary */
}
p.abs {
  position: absolute;
  left: 40px;
  top: -40px;
}
```

Fixed positioning

- Similar to absolute, but position is calculated relative to the top-left corner of a browser window and does not change position if the user scrolls the window.
- Example:

```
div.header {
  position: fixed;
  top: 0px;
  left: 0px;
  width: 100%;
  padding: 20px;
  font-size: 28px;
  color: white;
  background-color: blue;
  border: 2px solid black;
}
```

The `z-index` Property

- Using `z-index` property one can control which of the boxes appears on top in stacking context.
- Value of `z-index` is a number, and the higher number indicates that element should be displayed the nearer the top.
- Example:

```
p { width: 200px; background-color: white;
    padding: 5px; border: 2px solid black; }
p.one { z-index:3;
    position:absolute; left:0px; top:0px; }
p.two { z-index:1;
    position:absolute; left:10px; top:10px; }
p.three { z-index:2;
    position:absolute; left:20px; top:20px; }
```

The `float` Property

- The floated box (including its margins) will be aligned with the top of the containing box either left or right.
- The `float` property one can take the values `left`, `right`, `none`, `inherit`.
- `width` of the floating box must be specified, otherwise, it will automatically take up 100 percent of the containing box.
- Example:

```
p { width: 200px; background-color: white;
    padding: 5px; border: 2px solid black; }
span.quote { float:right; width:100px;
    padding:5px; margin:5px;
    border: 1px solid black; }
```

Page Layout Strategies

Main design issues

- Text should be structured in such a way that there is a proper line length (10 to 12 words or 60 to 75 characters per line).
- Text resizing impacts on the page layout.
- Web pages appear on browsers of all sizes (phone screens, desktop monitors, cinema displays).

Standard approaches

- Fixed layouts stay put at a specific pixel width regardless of the size of the browser window or text size.
- Fluid (or liquid) layouts resize proportionally when the browser window resizes.
- Elastic layouts resize proportionally based on the size of the text.
- Hybrid layouts combine fixed and scalable areas.

Fixed layout

- Fixed layout is the old, traditional layout, best to be viewed on monitors with similar resolutions only.
- Fixed layouts are created by specifying width values in pixel units.
- The most common width for fixed layout is 960px.
- Usually, the layout is either left-aligned or centered.

Fixed layout (ctd.)

- Pros:
  - The layout is predictable and offers better control over line length.
  - It is easier to design and produce.
- Cons:
  - Content on the right edge will be hidden if the browser window is smaller than the page.
  - There may be a lot of left over space on large screens.
  - Line lengths may grow awkwardly short at very large text sizes.
  - Takes control away from the user.

Liquid layout

- Fluid or liquid layout follows the default behavior of the normal flow, with no attempt to control the width of the content or line breaks; the text is permitted to reflow as required.
- Fluid layouts are a cornerstone of the responsive web design technique.
- Liquid layouts are created by specifying width values in percentages.

Liquid layout (ctd.)

- Pros:
  - No awkward empty space because the text fills the window.
  - On desktop browsers, users can control the width of the window and content.
  - No horizontal scrollbars.
- Cons:
  - They are less predictable; elements may be too spread out or too cramped at extreme browser dimensions (can be avoided using `min-width` and `max-width`).
  - More difficult to achieve whitespace.
  - More math involved in calculating measurements.

Elastic layout

- Elastic layout ensures that line lengths (in terms of words or characters per line) stay the same regardless of the text size.
- Elastic layouts have the same issues as fixed-width layouts and are generally not as useful as liquid layout in the mobile context.
- Elastic layouts are created by specifying width values in em units.

Elastic layout (ctd.)

- Pros:
  - Provides a consistent layout, while allowing flexibility in text size.
  - Tighter control over line lengths than liquid and fixed layouts.
- Cons:
  - Images and videos do not rescale automatically with the rest of the layout.
  - The width of the layout might exceed the width of the browser window at large text sizes.
  - Not as useful for addressing device and browser size variety.
  - More complicated to create than fixed layouts.

Hybrid layout

- Hybrid layouts are called those, that combine pixel, percentage and em unit measurements.
- Hybrid layouts usually overcome some of the disadvantages of a particular layout.
- Hybrid layout is most complicated to create and needs advanced calculation techniques to determine page and element widths.

Which one to use?

- There is no unique answer, it depends on your needs.
- Nowadays most popular layouts are liquid and hybrid layouts.
- Example:
  - Fixed layout can be used in intranet applications that run only desktop computers.
  - Liquid layout works best for small screens like mobile phones and tablets.

Page Layout Techniques

Introduction

- We will see some techniques to create multicolumn layouts using floats and positioning.
- These are not universal solutions, just a starting points.
- Your page may need more complicated approaches.

Multicolumn layout using floats

- Floats are the primary tool for creating columns on web pages.
- Advantages of floats over absolute positioning are that they prevent contents from overlapping, and make easier to keep footer at the bottom of the page.
- CSS3 provides `column-count` and `column-width` properties, but they are not well supported in browsers.
- Although, CSS3 columns has ability of laying out a single block of text across multiple columns, which is not possible via floats.

Two-column layout

- Strategy:
  - Create two block-level elements as columns and float them.
  - Set widths and margins on both column elements (make sure that total width does not exceed 100%).
  - Clear the footer to keep it at the bottom of the page.
- Example: HTML part:

```
<header> A header </header>
<article> Main part </article>
<aside> links, etc. </aside>
<footer> Copyright information </footer>
```

Two-column layout (ctd.)

- Example: CSS style:

```
article {
  float: left;
  width: 60%;
  margin: 0 2.5%;
}
aside {
  float: left;
  width: 30%;
  margin: 0 2.5%;
}
footer {
  clear: left;
}
```

Two-column layout (ctd.)

- Any ideas how to center the layout?

Two-column layout (ctd.)

- Any ideas how to center the layout?
- Solution:

```
body {
  width: 960px;
  margin: auto;
}
```

Two-column layout (ctd.)

- How to center only columns, and keep header and footer full-width?

Two-column layout (ctd.)

- How to center only columns, and keep header and footer full-width?
- Solution:

```
main {
  width: 960px;
  margin: auto;
}

<header> A header </header>
<main>
  <article> Main part </article>
  <aside> links, etc. </aside>
</main>
<footer> Copyright information </footer>
```

Two-column layout (ctd.)

- The main issue with floats is that the HTML source order matters.
- This can be avoided using negative margins.
- Example:

```
article {
   .....
  margin: 0 0 0 37.5%;
}
aside {
   .....
  margin: 0 0 0 -95%;
}
```

Positioning the layout

- The other way to create columns in a layout is to use absolute positioning.
- If you need to have a footer, then try to avoid absolute positioning.
- When you position all of the elements in a layout, they are no in the normal flow any more, meaning that nothing is holding a footer at the bottom of the page.

Positioning the layout (ctd.)

- Example:

```
main { position: relative; }
article {
  width: 60%;
  position: absolute;
  top: 0;
  left: 2.5%;
  margin: 0;
}
aside {
  width: 30%;
  position:absolute;
  top: 0;
  right: 2.5%;
  margin: 0;
}
```

Top-to-bottom column backgrounds

- Unfortunately, there is no supported way of setting the height of an element to 100% of the page height (there are some JavaScript workarounds).
- Therefore height of each column depends on its content.
- To add a background, there is a trick via setting background image on containing element.
- The trick is easy for fixed layout and becomes more complicated on liquid layouts.

Top-to-bottom column backgrounds (ctd.)

- Example:

```
main {
  overflow: hidden; /* or auto */
  background-image: url(two_cols.png);
  background-repeat: repeat-y;
  background-position: 65%;             }
article {
  float: left;
  width: 60%;
  margin: 0 2.5%;
}
aside {
  float: left;
  width: 30%;
  margin: 0 2.5%;
}
```

Responsive Web Design

Introduction

- Responsive Web Design is a CSS technique to adapt a page layout based on screen size.
- It is a complex topic, so we cover only basics here.
- Responsive page is constructed by combining three components: fluid layout, flexible images, and media queries.

Viewport

- Viewport is an abstract device, where objects are rendered before displaying on screen.
- Mobile web browsers use viewport to render pages as if it were on desktop browser window (default is 980 pixels).
- Viewport is then shrunk down to fit the width of the device screen (portrait 320 pixels on smartphone, iphone, and the like).
- This results in cramming a lot of information into a tiny space.

Viewport (ctd.)

- The first step to a responsive design is to set width of viewport to the actual width of device.
- You can do it by `meta` tag:

```
<meta name="viewport"
   content="width=device-width,
            initial-scale=1"    >
```

Flexible images

- The simplest way to treat images on responsive pages is to set `max-width` property.
- But the problem is far complicated: except loading the page properly, it is highly desired to avoid downloading unnecessary data.
- There are techniques called responsive images, but no perfect solution found so far.
- Responsive images can be a good topic for your bachelor's thesis.

Media queries

- Media queries allows to deliver styles based on media type.
- Media types are `print`, `screen`, `braille`, `projection`, `tty`, and `tv`.
- Media queries can also evaluate specific properties, and most of them can be tested for the `min-` and `max-` values as well.
- Media query starts with `@media` followed by the target media type `and` property/value pairs contained within parentheses.

Media queries (ctd.)

Property	Description
`width`	width of viewport.
`height`	height of viewport.
`device-width`	width of the whole screen.
`device-height`	height of the whole screen.
`orientation`	whether the device is in `portrait` or `landscape` orientation.
`aspect-ratio`	`width / height.`
`device-aspect-ratio`	`device-width / device-height.`
`resolution`	density of pixels in the device.

Media queries (ctd.)

- Example:

```
@media screen and (min-width: 480px) {
  /* styles for devices & browsers
     that pass this test goes here */
}
@media screen and (max-width: 700px)
              and (orientation: landscape) {
  /* styles for more complicated test */
}
@media screen and (-webkit-pointer: fine),
       screen and (-moz-pointer: fine) {
  /* styles if either test passes */
}
```

Media queries (ctd.)

- Media queries can be written in HTML `link` element as well using `media` attribute.
- Example:

```
<link rel="stylesheet" href="special.css"
    media="screen and (min-width:780px)" />
```

- This requires extra HTTP request for each additional `.css` file.
- Remember: styles lower in a stack override the styles that precede them.

Media queries (ctd.)

- A best practice for responsive sites is to adopt a mobile first mentality.
- "Mobile first" means that you create styles for the smallest devices first, and use media queries to override styles for bigger display devices.
- The point at which the media query delivers a new set of styles is known as a breakpoint.
- You can have as many breakpoints as you wish, but the most common for "mobile first" approach is to have 2 breakpoints: `min-width:481px` and `min-width:780px`.

Media queries (ctd.)

- Note that IE 8 and less does not support media queries.
- If you follow "mobile first" approach, you need to add additional `.css` file specially for IE8 and less.
- Example:

```
<!--[if (lt IE 9)&(!IEMobile)]>
<link rel="stylesheet" href="ie-layout.css">
<![endif]-->
```

Minimal example

```
// state for screens up to 480px     (phones)
article { background-color: red; }
aside { background-color: blue; }

// state for screens > 480px         (tablets)
@media screen and (min-width: 481px) {
  article, aside {
    border-radius: 1em;
    padding: 1em;
    margin: 1em 0;
  }
}
```

Minimal example (ctd.)

```
// state for screens > 780px          (computers)
@media screen and (min-width: 780px) {
  body {
    max-width: 960px;
  }
  article {
    float: left;
    margin: 1em;
    width: 60%;
  }
  aside {
    float: left;
    margin: 1em;
  }
}
```

Laboratory Work

Exercises

- Create a three-column fixed layout using floats.
- Change the layout to be liquid, then elastic; determine the differences.
- Modify the layout to use positioning and try to keep footer at the end of the page.
- From all the lab-works assemble a good looking responsive web site.

Discussion?!