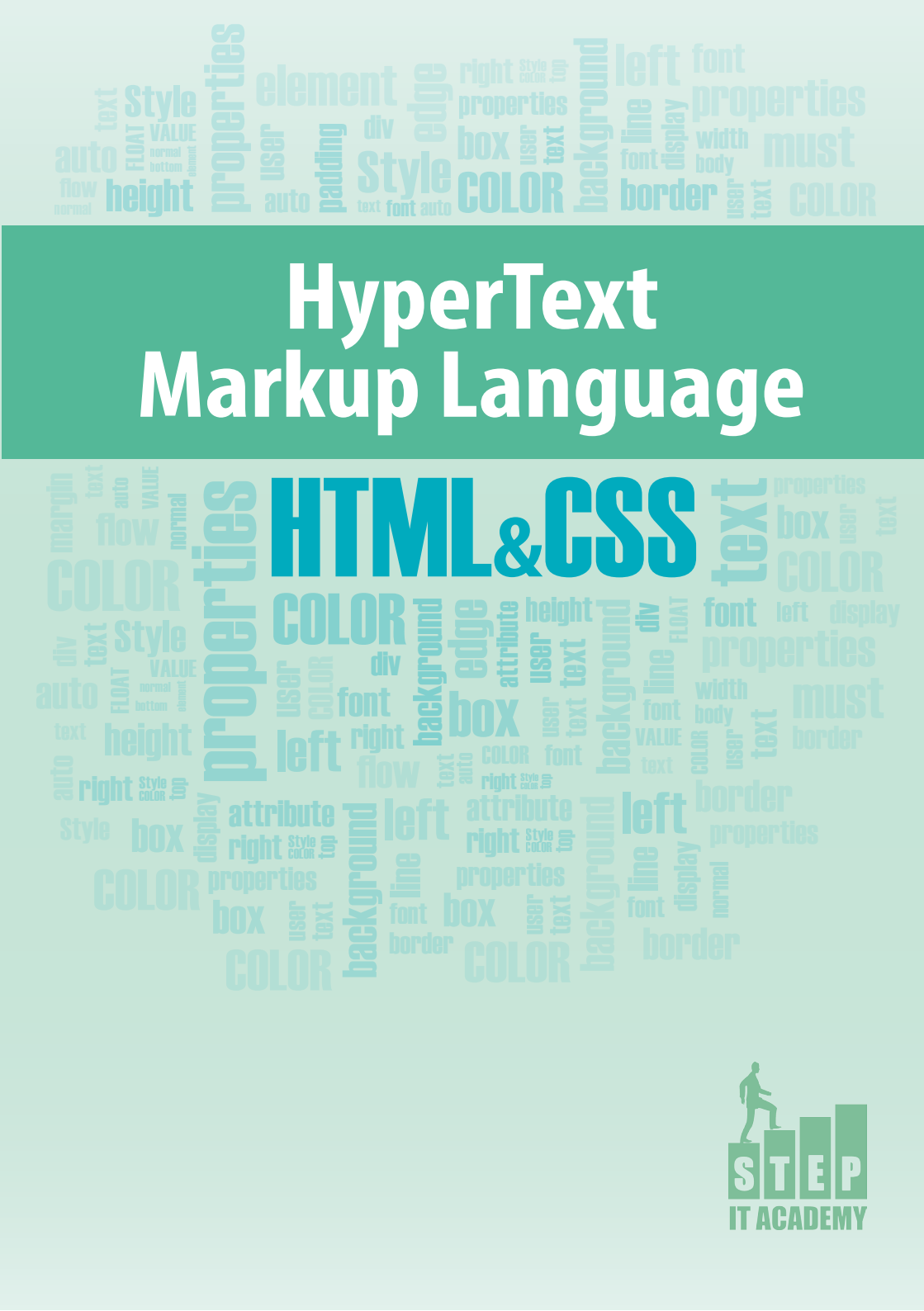
HyperText Markup Language



STEP IT ACADEMY

Unit 11

Creating pages with HTML5 and CSS3

Contents

New Structural Tags

Condition to create new structural tags was the wish to divide web pages into logical parts using elements whose descriptive names correspond to the content they contain. Conceptually, a web page can be considered as a document. Documents contain headers, footers, sections, and other designations that divide documents into logical parts.

This section reviews the current methods of dividing an HTML document using generic sample code. In the rest of this lesson, you'll revise the original code using the new HTML5 structural tags to see step-by-step how the document is transformed into logical sections.

HTML 4 Approach

If you've created even the simplest of HTML documents, then you're familiar with the div tag. The div tag is the major mechanism in the pre-HTML5 era for creating blocks of content in an HTML document. For example, Listing 2 shows how you can use div tags to create a simple page with a header, content area, and footer.

Listing 2. Simple HTML page using div tags

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>

<head>
    <title>
        A Simple HTML Page Using Divs
    </title>
</head>
```

```
<body>
    <div id='header'>Header</div>
    <div id='content'>Content</div>
    <div id='footer'>Footer</div>
</body>

</html>
```

This works fine; the div tag is a nice general purpose tag. However, other than by looking at the id attribute of each div tag, it's hard to tell what section of the document each div tag represents. Though you could argue that the id is enough of an indicator if properly named, the id attributes are arbitrary. There are many variations that can be considered equally valid ids. The tag itself gives no indication of the type of content it is intended to represent.

HTML5 Approach

HTML5 addresses this issue by providing a set of tags that more clearly defines the major blocks of content that make up an HTML document. Regardless of the final content displayed by a web page, most web pages consist of varying combinations of common page sections and elements.

The code in Listing 2 creates a simple page with a header, content area, and footer. These, and other sections and page elements, are quite common, so HTML5 includes tags that break documents into the common sections and indicates the content contained in each section. The new tags are:

- header
- section
- article

- aside
- footer
- nav.

The rest of this article gives an overview of each tag. You will also learn about the intended use of the tags by revising the original div-based code example from Listing 2 to use the new HTML5 structural tags.

The header Tag

As the name suggests, the header tag is intended to mark a section of the HTML page as the header. Listing 3 shows the code example from Listing 2 modified to use a header tag.

Listing 3. Adding a header tag

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      A Simple HTML Page
    </title>
  </head>

  <body>
    <header>Header</header>
    <div id='content'>Content</div>
    <div id='footer'>Footer</div>
  </body>
</html>
```

The doctype in Listing 3 was also changed to indicate that the browser should use HTML5 to render the page. From this point on, all of the examples assume you are using the correct doctype.

The section Tag

The section tag is meant to identify significant portions of the content on the page. This tag is somewhat analogous to dividing a book into chapters. Adding a section tag to the code example results in the code in Listing 4.

Listing 4. Adding a section tag

```
<!DOCTYPE html>
<html>

    <head>
        <title>
           A Simple HTML Page
        </title>
    </head>

    <body>
        <header>Header</header>
          <section>
            <p>
              This is an important section of the page.
            </p>
          </section>
       <div id='footer'>Footer</div>
  </body>

</html>
```

The article Tag

The article tag identifies major sections of content within a web page. Think of a blog, where each individual post constitutes a significant piece of content. Adding article tags to the code example results in the code shown in Listing 5.

Listing 5. Adding article tags

```
<!DOCTYPE html>
<html>

    <head>
        <title>
            A Simple HTML Page
        </title>
    </head>

    <body>
        <header>
            Header
        </header>
        <section>
            <article>
                <p>
                 This is an important section
                 of content on the page.
                 Perhaps a blog post.
                </p>
            </article>
            <article>
                <p>
                This is an important section
                of content on the page.
                Perhaps a blog post.
                </p>
            </article>
        </section>

        <div id='footer'>
             Footer
        </div>
      </body>
</html>
```

The aside Tag

The aside tag indicates that the content contained within the tag is related to the main content of the page but is not part of it. It's somewhat analogous to using parentheses to make a comment in the body of text (like this one). The content in the parentheses provides additional information about the element that encloses it. Adding an aside tag to the code example results in the code in Listing 6.

Listing 6. Adding an aside tag

```
<!DOCTYPE html>
<html>

    <head>
        <title>
           A Simple HTML Page
        </title>
    </head>

    <body>
        <header>
            Header
        </header>

        <section>
            <article>
               <p>
                  This is an important section
                  of content on the page.
                  Perhaps a blog post.
               </p>
              <aside>
               <p>
                  This is an aside for the first
                   blog post.
```

```
                </p>
              </aside>
            </article>

            <article>
                <p>
                    This is an important section
                    of content on the page.
                    Perhaps a blog post.
                </p>
            </article>

        </section>
        <div id='footer'>
            Footer
        </div>

    </body>
</html>
```

The footer Tag

The footer tag marks the contained content of the element as the footer of the document. Adding a footer tag to the code example results in the code shown in Listing 7.

Listing 7. Adding a footer tag

```
<!DOCTYPE html>
<html>

    <head>
        <title>
            A Simple HTML Page
        </title>
    </head>
```

```
    <body>
        <header>
            Header
        </header>
            <section>
              <article>
                <p>
                   This is an important section of
                   content on the page.
                   Perhaps a blog post.
                </p>
                <aside>
                <p>
                   This is an aside for the first
                   blog post.
                </p>
                </aside>
              </article>

              <article>
                <p>
                   This is an important section
                   of content on the page.
                   Perhaps a blog post.
                </p>
              </article>
            </section>

            <footer>
                Footer
            </footer>
    </body>
</html>
```

At this point, all of the original div tags have been replaced with HTML5 structural tags.

The nav Tag

The content contained within the nav tag is intended for navigational purposes. Adding a nav tag to the code example results in the code in Listing 8.

Listing 8. Adding a nav tag

```
<!DOCTYPE html>
<html>

    <head>
        <title>
            A Simple HTML Page
        </title>
    </head>

    <body>
        <header>Header
           <nav>
              <a href='#'>Some Nav Link</a>
              <a href='#'>Some Other Nav Link</a>
              <a href='#'>A Third Nav Link</a>
           </nav>
        </header>

        <section>
            <article>
                    <p>
                        This is an important section
                        of content on the page.
                        Perhaps a blog post.
                    </p>
                <aside>
                    <p>
                        This is an aside for the first
                        blog post.
                    </p>
```

```
                </aside>
            </article>

            <article>
                    <p>
                        This is an important section
                        of content on the page.
                        Perhaps a blog post.
                    </p>
            </article>
        </section>

        <footer>
           Footer
        </footer>
    </body>
</html>
```

Finished Example

Listing 9 shows the result of replacing the original div tags with the new HTML5 structural tags.

Listing 9. Finished example

```
<!DOCTYPE html>
<html>
    <head>
        <title>
             A Simple HTML Page
        </title>
    </head>

    <body>
        <header>Header
            <nav>
```

```
                <a href='#'>Some Nav Link</a>
                <a href='#'>Some Other Nav Link</a>
                <a href='#'>A Third Nav Link</a>
            </nav>
        </header>

        <section>
            <article>
                <p>
                    This is an important section
                    of content on the page.
                    Perhaps a blog post.
                </p>
                <aside>
                    <p>
                       This is an aside for the first
                        blog post.
                    </p>
                </aside>
            </article>

            <article>
                <p>
                    This is an important section
                    of content on the page.
                    Perhaps a blog post.
                </p>
            </article>
        </section>

        <footer>
            Footer
        </footer>

  </body>
</html>
```

Though the example is simple, for demonstration purposes, when you compare the original div-based example from Listing 2 to the result in Listing 9, the intent of the new structural tags should be clear.

Conclusion

The new HTML5 tags describe the types of content that they contain, and they help divide the document into logical sections. It's still up to you to decide when and where to use the new tags within a document, similar to an author writing a book. While two authors writing the same book may choose different ways of dividing the book into chapters, the act of using chapters still provides a consistent method of dividing the book into sections. Similarly, while two authors of a given web page may choose different structures, the new HTML5 structural tags provide new conventions that web page developers can use that the old div tags do not provide.

Transformations and Transitions

Learn how to use transformations to alter elements and how to use transitions and animation to create visual effects in a document.

- You can create `transition` effects using the new CSS3 transition property. To create transitions, specify which CSS property will be changed and change speed in seconds. In order to add a transition effect to several properties, simply list their names separated by commas.
- Transition smoothness is controlled by the timing options. There are several types of such functions in CSS3:

`linear`; `ease`; `ease-in-out`; `ease-in`; `ease-out`; `cubic-bezier`: manual set of transition value.

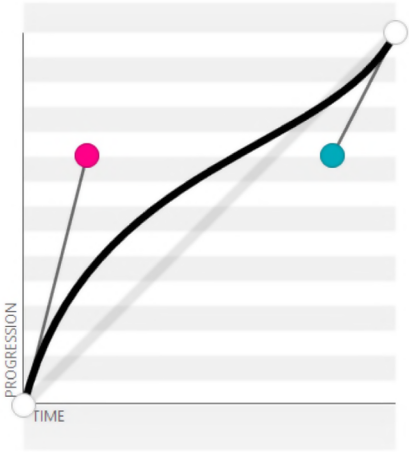
- Transition properties:
  - ▷ `transition-property` defines animated property;
  - ▷ `transition-duration` is duration of animation;
  - ▷ `transition-timing-function` is a function that defines animation speed;
  - ▷ `transition-delay` is a delay before animation start transition: background-color 2s linear 0.5s.
- Animatable properties:

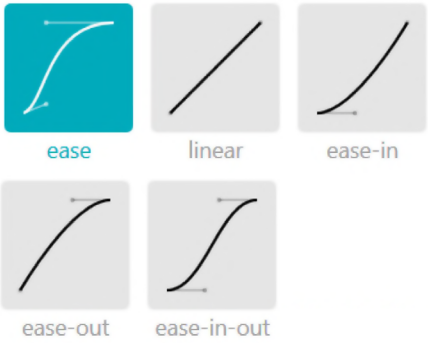
```
color
font-size
font-weight
letter-spacing
line-height
text-indent
text-shadow
vertical-align
word-spacing
background
background-color
background-image
background-position
border-spacing
border-left-width
border-top-left-radius, etc. visibility
box-shadow z-index clip crop
height, width, min-height etc.
```

```
margin-left etc.
opacity
outline-width
outline-offset
outline-color
padding-left etc.
bottom
top
left
right.
```

- In order to create animation in CSS3, the `@keyframes` property is used.
- In order to set CSS3 animation for an element, use the CSS3 animation property. This property is a shorthand notation of various animation properties:
  - ▷ `animation-name` sets animation name;
  - ▷ `animation-duration` sets time of animation playback;
  - ▷ `animation-timing-function` describes calculation method of intermediate property values for animation;
  - ▷ `animation-delay` sets animation delay;
  - ▷ `animation-iteration-count` sets number of animation loops;
  - ▷ `animation-direction` sets animation direction;
  - ▷ `animation-fill-mode` shows which animation frame should be used as an element state after animation;
  - ▷ `animation-play-state` determines whether the animation is played or paused.

Transition-timing-function





http://cubic-bezier.com

Transition Properties

N	Property	Purpose
1	animation-name	Animation name
2	animation-duration	Duration of animation
3	animation-timing-function	Function defining animation speed
4	animation-delay	Delay before animation start
5	animation-iteration-count	Amount of animation iterations
6	animation-direction	Should the animation be played in reverse
7	animation-fill-mode	Which animation frame should be used as an element state after animation

```
animation: move 4s linear 1.5s infinity alternate none;
```

- You can choose the most relevant curve type for interpolation using the `animation-timing-function` (-webkit-animation-timing-function) property.

- We can use four transition properties:
  - ▷ `transition-property` — defines name (names) of CSS properties to which transitions should be applied.
  - ▷ `transition-duration` — defines time during which transition should be performed.
  - ▷ `transition-timing-function` — defines how intermediate transition values are calculated.
  - ▷ `transition-delay` — defines when transition is started.

Transition Properties

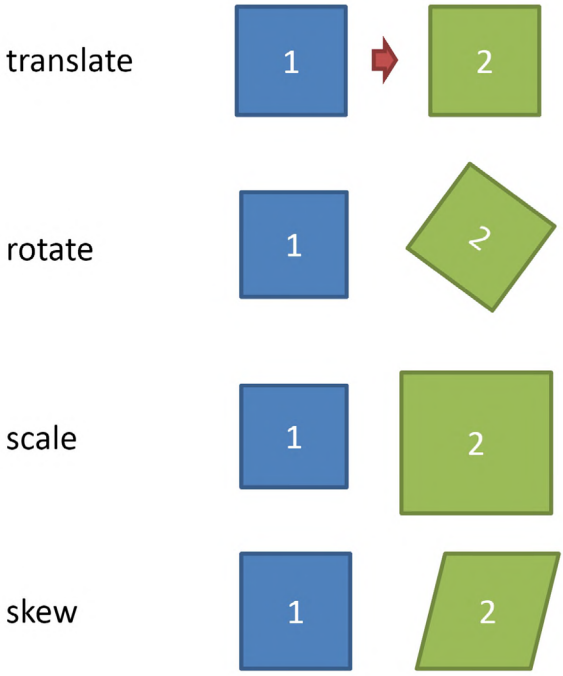
N	Property	Purpose
1	transition-property	Defines animated property
2	transition-duration	Duration of animation
3	transition-timing-function	Function that defines animation speed
4	transition-delay	Delay before animation start


```
transition: background-color 2s linear 0.5s;
```

- Each key frame describes how an animated element should behave in this time point of animation. Key frames take percent values to define time: 0% is start of animation, while 100% is its end. You can add intermediate frames for animations.
- In order to transfer an object, you can use `transform: translate(x,y)`. The transform parameter will only transfer an object from one point to another, but it won't create an animation of this transition. Add the transition parameter in the class in order to fix this.
- To transit an object diagonally, combine x and y parameters. Syntax will be as follows: `transform: trans-`

`late(x,y)`. Depending on the direction, x and y values can be both positive and negative.

- Rotation is controlled by degree (from 0° to 360°). To rotate an object, apply the following parameters: `transform: rotate(ndeg)`; where n is degree.
- In order to rotate an object clockwise, apply positive value for `rotate(ndeg)`.
- Using the `scale(n)` or `scale(x,y)` parameter, we can increase or decrease an object within HTML. The object will change size depending on the n/x,y value, where X axis is width, and Y is height.



- Apply multiple commands by simply listing commands separated by space.

- The blur filter blurs the image.
  `filter: blur(Add, direction, strength)` — add — object (0 is filter result, 1 means the filter will be placed on the source pattern).
  - ▷ `direction` is an object blur direction in degrees (0 is up and clockwise further in increments of 45°).
  - ▷ `strength` is blur degree.
- The chroma filter makes the set color transparent.
  `filter: chroma(color)`
- The glow filter adds glow along the outer borders of an object, creating the border flame effect.
  `filter: glow(strength, color)`
- The gray filter removes color set of an object and displays it in gray hues.
  `img{ filter: gray() }`
- The invert filter changes hue, depth, and brightness of an object to the opposite ones.
  `img{ filter: invert() }`
- The wave filter distorts an object in a sinusoidal way along the vertical axis.
  Filter: `wave(add, freq, lightStrength, phase, strength)`.
- The xray filter displays an object in black and white, like in an X-ray picture.
  `img{ filter:xra() }`
- A new specification, Filter Effects 1.0, was proposed at the end of last year.
  - ▷ `hue-rotate;`
  - ▷ `sepia;`

- ▷ `brightness;`
- ▷ filters can be combined: `-webkit-filter: brightness(60%) sepia (100%);`
- ▷ `contrast;`
- ▷ `saturate`.

Quick Check

- Name ways to create animation in CSS3.
- What transformation do you remember?
- How is the animation start moment defined? Provide examples.
- What are filters?

Additional Task

Add several images to a page. Place a hidden text block describing the image next to each image. Create a style, which starts animation and smoothly displays description when hovering over image.

Individual Task

Task 1

Learn main concepts discussed in the lesson.

Task 2

Create a page, place several images rotated at different angles on a page. Elaborate animation that will turn an image clockwise or counter-clockwise when hovering over it in such a way that the image becomes vertical for the user.

Task 3

Create a horizontal menu and develop an animated dropdown submenu.

Task 4

Create several links on the page. Develop animation that starts when hovering over it. Animation should smoothly change the link background and make text size of the link slightly bigger.

Recommended Resources

Animation:
http://www.css3files.com/.

Transition:
http://www.css3files.com/transition/.

Transform:
http://www.css3files.com/transform-rotate/.

Responsive Web Design: What It Is And How To Use It

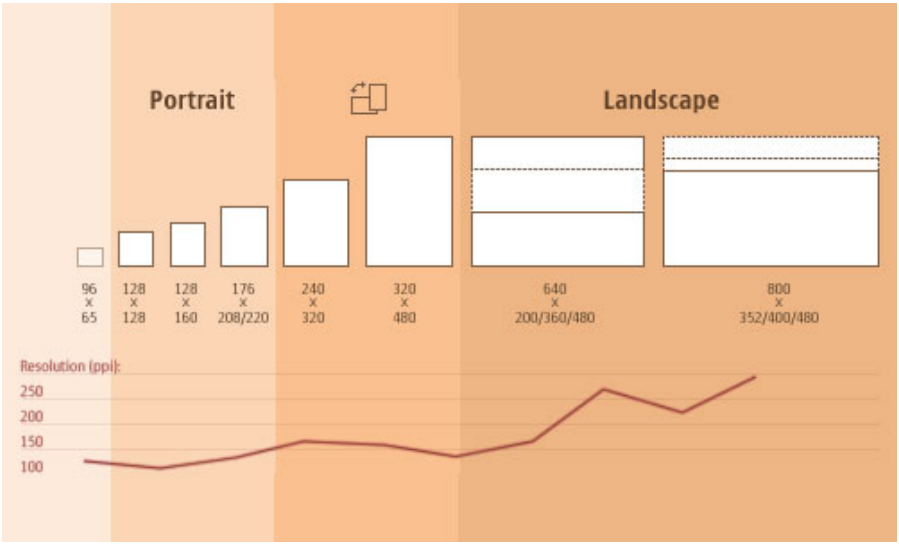
Almost every new client these days wants a mobile version of their website. It's practically essential after all: one design for the iPhone, another for the iPad — and all screen resolutions must be compatible, too.

In the field of Web design and development, we're quickly getting to the point of being unable to keep up with the endless new resolutions and devices. For many websites, creating a website version for each resolution and new device would be impossible, or at least impractical. Should we just suffer the consequences of losing visitors from one device, for the benefit of gaining visitors from another? Or is there another option?

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. In other words, the website should have the technology to automatically respond to the user's preferences. This would eliminate the need for a different design and development phase for each new gadget on the market.

Adjusting Screen Resolution

It is possible to group devices into major categories, design for each of them, and make each design as flexible as necessary. But that can be overwhelming, and who knows what the usage figures will be in five years? Besides, according to the statistics based on about 400 devices sold between 2005 and 2008, we have the whole range of various devices:



It's obvious that we can't keep creating custom solutions for each one. So, how do we deal with the situation?

PART OF THE SOLUTION: FLEXIBLE EVERYTHING

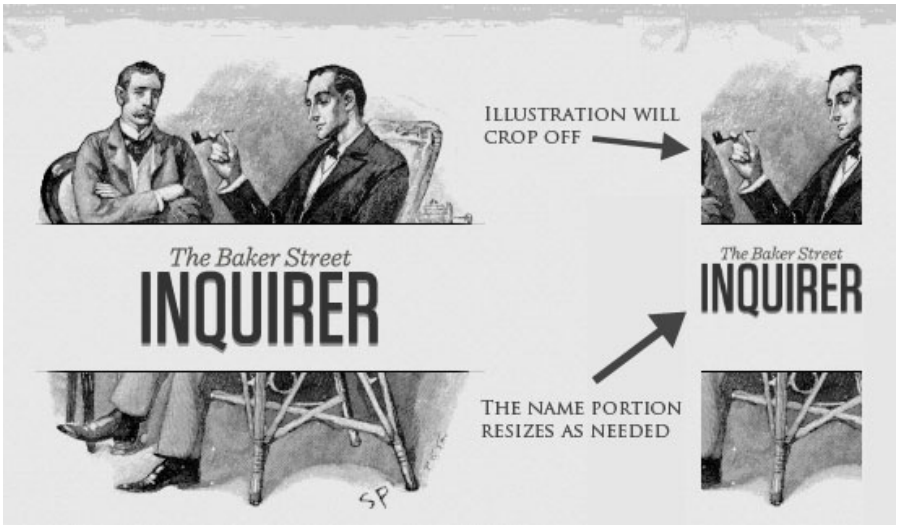
While it's not a complete fix, the solution gives us far more options.

In Ethan Marcotte's article, he created a sample Web design that features this better flexible layout:

The entire design is a lovely mix of fluid grids, fluid images and smart mark-up where needed. Creating fluid grids is fairly common practice, and there are a number of techniques for creating fluid images.

For more information on creating fluid websites, be sure to look at the book "Flexible Web Design: Creating Liquid and Elastic Layouts with CSS" by Zoe Mickley Gillenwater, and download the sample chapter "Creating Flexible Images."

While from a technical perspective this is all easily possible, it's not just about plugging these features in and being done. Look at the logo in this design, for example:



If resized too small, the image would appear to be of low quality, but keeping the name of the website visible and not cropping it off was important. So, the image is divided into two: one (of the illustration) set as a background, to be cropped and to maintain its size, and the other (of the name) resized proportionally.

```
<h1 id="logo">
   <a href="#">
      <img src="site/logo.png"
           alt="The Baker Street Inquirer"/>
   </a>
</h1>
```

Above, the h1 element holds the illustration as a background, and the image is aligned according to the container's background (the heading).

Flexible Images

One major problem that needs to be solved with responsive Web design is working with images. There are a number of techniques to resize images proportionately, and many are easily done. The most popular option is to use CSS's *max-width* for an easy fix.

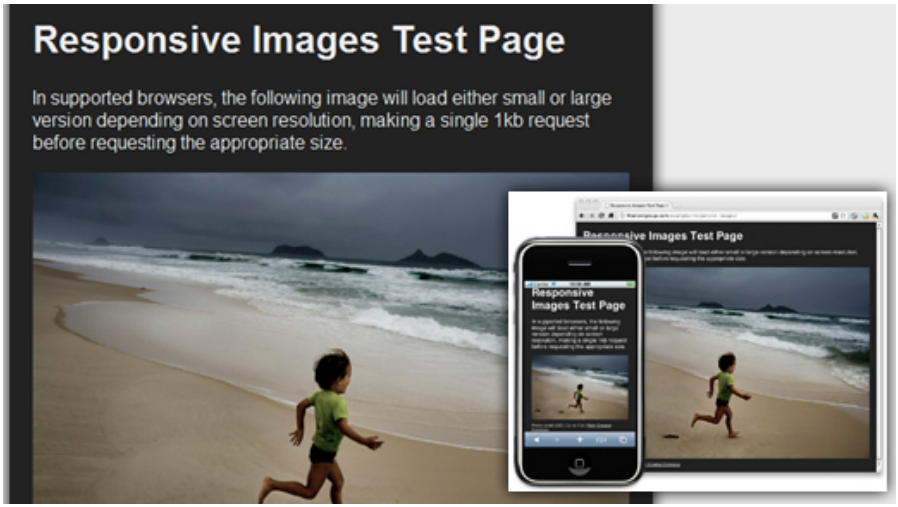
```
img {max-width: 100%;}
```

The maximum width of the image is set to 100% of the screen or browser width, so when that 100% becomes narrower, so does the image. Note that *max-width* is not supported in IE, so use *width: 100%*.

While the above is a great quick fix and good start to responsive images, image resolution and download times should be the primary considerations.

ANOTHER WAY:
RESPONSIVE IMAGES

This technique, presented by the Filament Group, takes this issue into consideration and not only resizes images proportionately, but shrinks image resolution on smaller devices, so very large images don't waste space unnecessarily on small screens.



This technique requires a few files, all of which are available on Github. First, a JavaScript file (rwd-images.js), the .htaccess file and an image file (rwd.gif). Then, we can use just a bit of HTML to reference both the larger and smaller resolution images: first, the small image, with an .r prefix to clarify that it should be responsive, and then a reference to the bigger image using *data-fullsrc*.

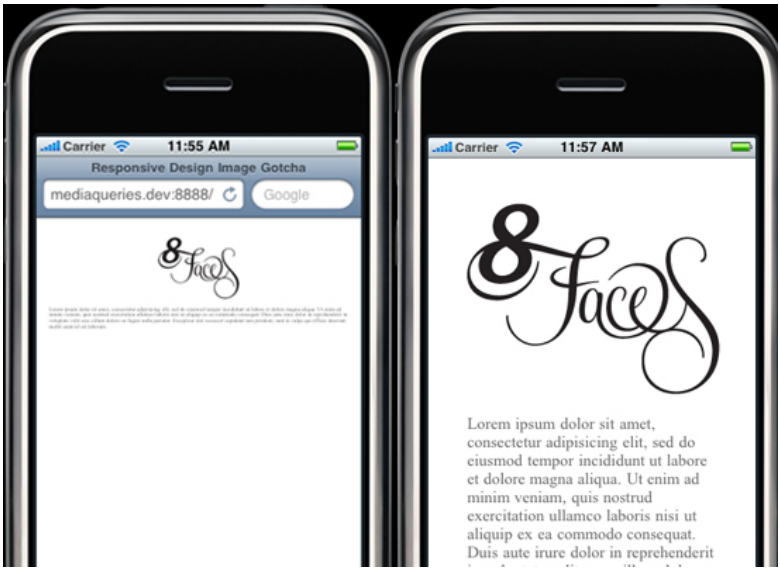
```
<img src="smallRes.jpg" data-fullsrc="largeRes.jpg">
```

The data-fullsrc is a custom HTML5 attribute, defined in the files linked to above. For any screen that is wider than 480 pixels, the larger-resolution image (largeRes.jpg) will load; smaller screens wouldn't need to load the bigger image, and so the smaller image (smallRes.jpg) will load.

Interesting iPhone Feature

One nice thing about the iPhone and iPod Touch is that Web designs automatically rescale to fit the tiny screen. A full-

sized design, unless specified otherwise, would just shrink proportionally for the tiny browser, with no need for scrolling or a mobile version. But image or text is too small:



To fix this problem, use the meta tag:

```
<meta name="viewport"
      content="width=device-width; initial-scale=1.0">
```

If the *initial-scale* equals 1, image width is the same as the device's width.

Custom Layout Structure

For extreme size changes, we may want to change the layout altogether, either through a separate style sheet or, more efficiently, through a CSS media query. This does not have to be troublesome; most of the styles can remain the same, while specific style sheets can change.

For example, we could have one main style sheet (which would also be the default) that would define all of the main structural elements, such as *#wrapper, #content, #sidebar, #nav*, along with colors, backgrounds and typography. If a style sheet made the layout too narrow, short, wide or tall, we could then detect that and switch to a new style sheet.

style.css (default):

```
/* Default styles that will carry to the child style
   sheet */
html, body {
    background... font... color...
}
h1,
h2,
h3{}
p,
blockquote,
pre,
code,
ol,
ul {}
/* Structural elements */
#wrapper {
    width: 80%;
    margin: 0 auto;
    background: #fff;
    padding: 20px;
}

#content {
    width: 54%;
    float: left;
    margin-right: 3%;
}
```

```
#sidebar-left {
    width: 20%;
    float: left;
    margin-right: 3%;
}

#sidebar-right {
    width: 20%;
    float: left;
}
```

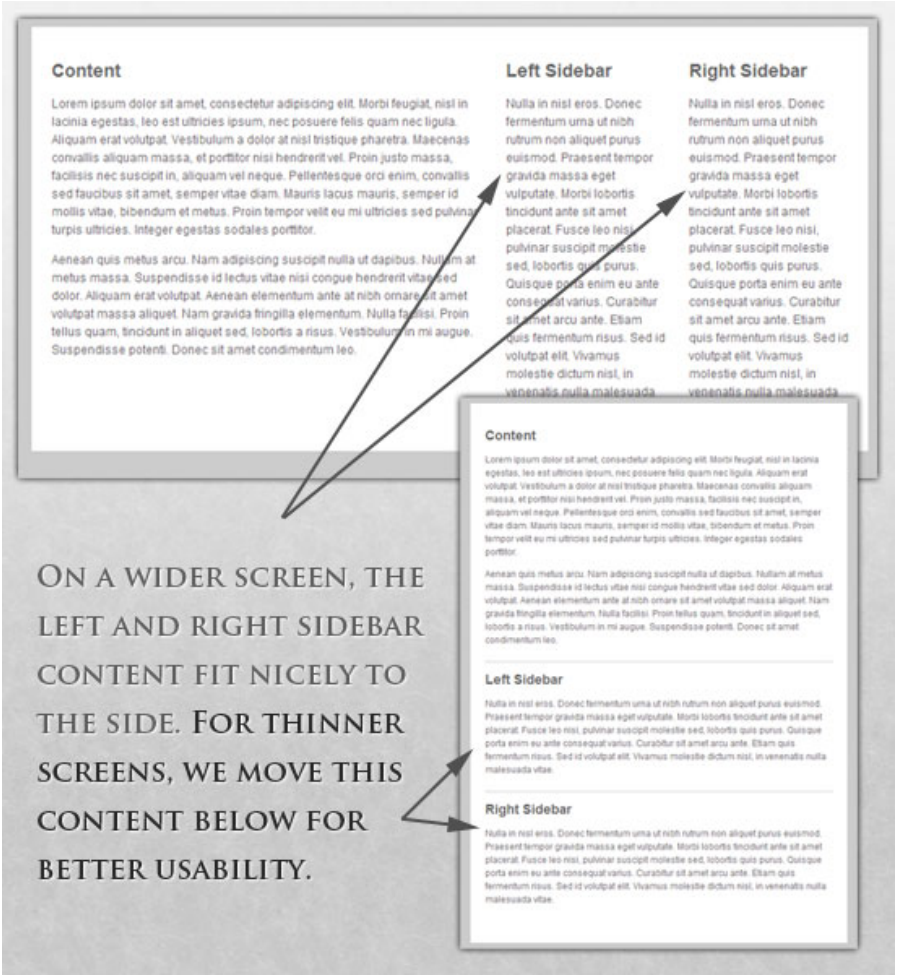
mobile.css (child):

```
#wrapper {
    width: 90%;
}

#content {
    width: 100%;
}

#sidebar-left {
    width: 100%;
    clear: both;
/* Additional styling for our new layout */
    border-top: 1px solid #ccc;
    margin-top: 20px;
}

#sidebar-right {
    width: 100%;
    clear: both;
/* Additional styling for our new layout */
    border-top: 1px solid #ccc;
    margin-top: 20px;
}
```



CSS3 MEDIA QUERIES

Let's see how we can use CSS3 media queries to create responsive design. The min-width property sets a minimum browser or screen width that a certain set of styles (or separate style sheet) would apply to. If anything is below this limit, the style sheet link or styles will be ignored. The max-width property does just the opposite.

Example:

```
@media screen and (min-width: 600px) {
    .hereIsMyClass {
        width: 30%;
        float:
        right;
    }
}
```

This media query will run only if the min-width is 600 pixels or wider.

```
@media screen and (max-width: 600px) {
    .aClassforSmallScreens {
        clear: both;
        font-size: 1.3em;
    }
}
```

In this case class (aClassforSmallscreens) will work only when screen width is 600 pixels or narrower.

While the above min-width and max-width can apply to screens and browser windows, we might only need to work with the device width. The min-device-width and max-device-width media query properties are great for this:

```
@media screen and (max-device-width: 480px) {
    .classForiPhoneDisplay {
        font-size: 1.2em;
    }
}

@media screen and (min-device-width: 768px) {
```

```
    .minimumiPadWidth {
        clear: both;
        margin-bottom: 2px solid #ccc;
    }
}
```

For the iPad specifically, there is also a media query property called orientation. The value can be either landscape (horizontal orientation) or portrait (vertical orientation).

```
@media screen and (orientation: landscape) {
    .iPadLandscape {
        width: 30%;
        float: right;
    }
}

@media screen and (orientation: portrait) {
    .iPadPortrait {
        clear: both;
    }
}
```

There are also many media queries that make sense when combined, for example:

```
@media screen and (min-width: 800px)
and (max-width: 1200px) {
    .classForaMediumScreen {
        background: #cc0000;
        width: 30%;
        float: right;
    }
}
```

The above code in this media query applies only to screen and browser widths between 800 and 1200 pixels.

Some designers would also prefer to link to a separate style sheet for certain media queries, this could be done as follows:

```
<link rel="stylesheet" media="screen
      and (max-width: 600px)" href="small.css"/>
<link rel="stylesheet" media="screen
      and (min-width: 600px)" href="large.css"/>
<link rel="stylesheet" media="print"
      href="print.css"/>
```

JavaScript

If your browser does not support CSS3 media queries, you can change styles using jQuery as follows:

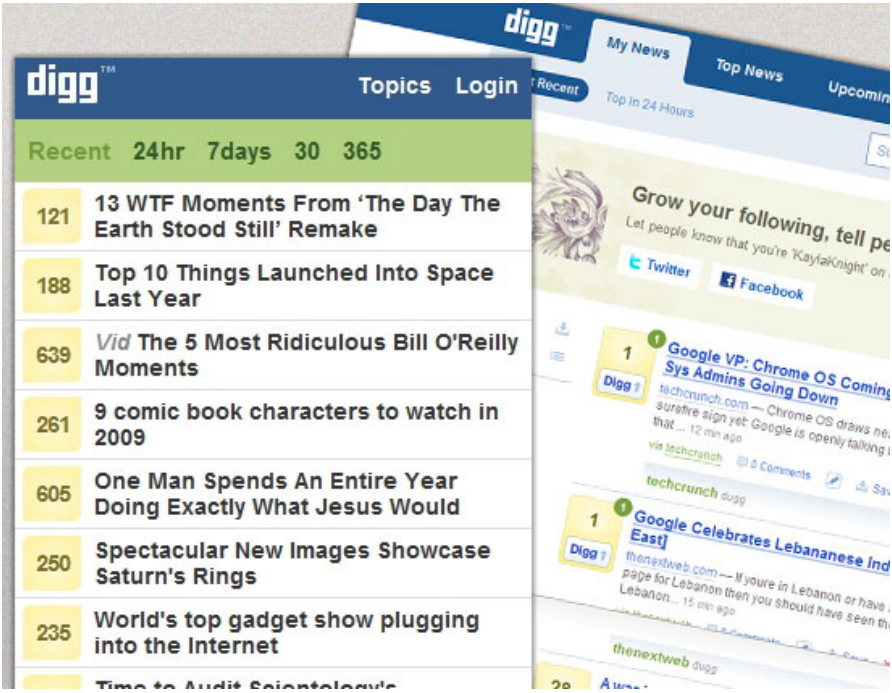
```
<script type="text/javascript" src="http://ajax.
googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js">
</script>
<script type="text/javascript">
$(document).ready(function(){
$(window).bind("resize", resizeWindow);

function resizeWindow(e){
    var newWindowWidth = $(window).width();
    // If width width is below 600px,
    // switch to the mobile stylesheet
    if(newWindowWidth < 600){
        $("link[rel=stylesheet]").
          attr({href : "mobile.css"});
    }
    else if(newWindowWidth > 600){
    // Else if width is above 600px, switch
    // to the large stylesheet
```

```
    $("link[rel=stylesheet]").
      attr({href : "style.css"});
    }
}
});
</script>
```

Showing or Hiding Content

It is possible to shrink things proportionally and rearrange elements as necessary to make everything fit (reasonably well) as a screen gets smaller. We have best practices for mobile environments: simpler navigation, more focused content, lists or rows instead of multiple columns.

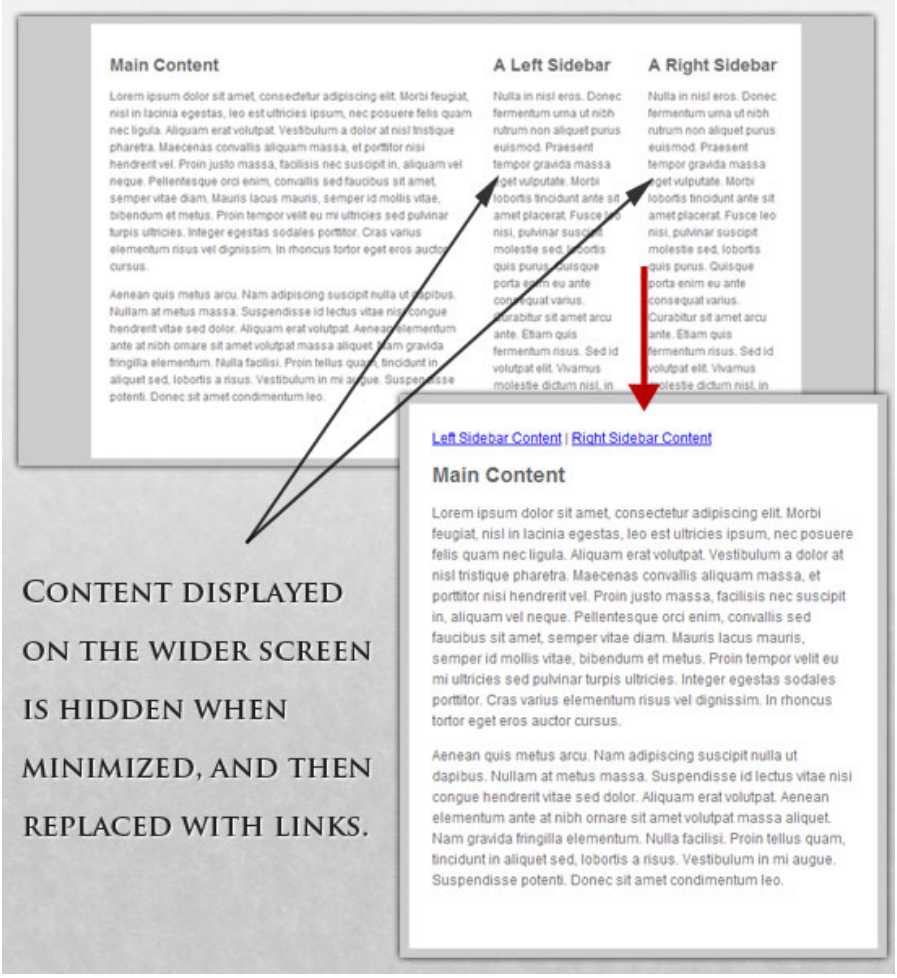


Fortunately, CSS has been allowing us to show and hide content with ease!

```
display: none;
```

display: none is used for objects that need to be hidden.

Example:



Here is our mark-up:

```
<p class="sidebar-nav">
    <a href="#">Left Sidebar Content</a> |
    <a href="#">Right Sidebar Content</a>
</p>

<div id="content">
    <h2>Main Content</h2>
</div>

<div id="sidebar-left">
    <h2>A Left Sidebar</h2>
</div>

<div id="sidebar-right">
    <h2>A Right Sidebar</h2>
</div>
```

In our default style sheet below, we have hidden the links to the sidebar content. Because our screen is large enough, we can display this content on page load.

style.css (default)

```
#content {
    width: 54%;
    float: left;
    margin-right: 3%;
}

#sidebar-left {
    width: 20%;
    float: left;
    margin-right: 3%;
}
```

```
#sidebar-right {
    width: 20%;
    float: left;
}

.sidebar-nav {
    display: none;
}
```

Now, we hide the sidebars and show the links:

mobile.css (simpler)

```
#content {
    width: 100%;
}

#sidebar-left {
    display: none;
}

#sidebar-right {
    display: none;
}

.sidebar-nav
{
    display: inline;
}
```

With the ability to easily show and hide content, rearrange layout elements and automatically resize images, form elements and more, a design can be transformed to fit a huge variety of screen sizes and device types. As the screen gets smaller, rearrange elements to fit mobile guidelines; for exam-

ple, use a script or alternate style sheet to increase white space or to replace image navigation sources on mobile devices for better usability.

Creating Website with Layout

Software: PC with installed Microsoft Visual Studio.

Review, aim and objective of the lesson

Creating a website. Improvement of new pages for the AQUATIC website.

Having learned material of this lesson, the student will be able to:

Lay out a website and add new pages to the already existing website.

Summary

- The first step after receiving requirements document is to create a layout and slice all required elements.
- Images should be sliced so they are of minimum size and used to the maximum in layout.
- Then you need to lay out the website.
- After creating a page, you should create all other pages and alter them in the appropriate way.
- Navigation should be created via the Menu element where the complete website model should be implemented using links to other pages.
- The final step is cross-browser testing. Test your website in the most used browsers: Mozilla Firefox, Google Chrome, Opera, and Internet Explorer.

Quick Check

1. What stages of website creation have you learned today?
2. What is website layout?
3. What is requirements document?
4. What cross-browser problems have you faced?

Recommended Resources

http://html.com/.

LESS and Responsive Design

Software: PC with installed Visual Studio 2010 or later or Notepad++.

Review, aim and objective of the lesson

This lesson considers the concept of responsive design and ways to create applications adapted not only to desktop browsers, but also to mobile devices. Also, this lesson reviews a method of creating styles using the LESS library.

Having learned material of this lesson, the student will be able to:

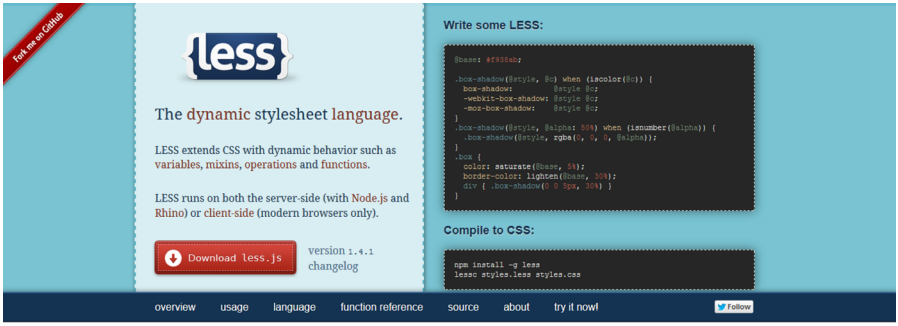
5. Use various responsive design techniques for different devices and browsers.
6. Develop CSS for correct displaying on various mobile devices.
7. Change object sizes relative to the browser window sizes.
8. Use LESS.
9. Use Skeleton.

Table of Contents

3. Responsive web design techniques.
4. Media Queries.
5. CSS Skeleton framework.

Summary

- LESS extends CSS with dynamic behavior such as variables, mixins, operations, and functions. http://lesscss.com/.



Variables

```
// LESS
@color: #4D926F;
#header {
    color: @color;
}
h2 {
    color: @color;
}

/* Compiled CSS */
#header {
    color: #4D926F;
}
h2 {
    color: #4D926F;
}
```

- Mixin

```
// LESS
.rounded-corners (@radius: 5px) {
    -webkit-border-radius: @radius;
    -moz-border-radius: @radius;
    -ms-border-radius: @radius;
    -o-border-radius: @radius;
    border-radius: @radius;
}

#header {
    .rounded-corners;
}

#footer {
    .rounded-corners(10px);
}

/* Compiled CSS */
#header {
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    -ms-border-radius: 5px;
    -o-border-radius: 5px;
    border-radius: 5px;
}

#footer {
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    -ms-border-radius: 10px;
    -o-border-radius: 10px;
    border-radius: 10px;
}
```

- Nested rules

```
// LESS
#header {
    h1 {
        font-size: 26px;
        font-weight: bold;
    }
    p { font-size: 12px;
        a { text-decoration: none;
            &:hover { border-width: 1px }
        }
    }
}

/* Compiled CSS */
#header h1 {
    font-size: 26px;
    font-weight: bold;
}

#header p {
    font-size: 12px;
}

#header p a {
    text-decoration: none;
}

#header p a:hover {
    border-width: 1px;
}
```

- Base operations

```
// LESS
@the-border: 1px;
@base-color: #111;
@red: #842210;

#header {
    color: (@base-color * 3);
    border-left: @the-border;
    border-right: (@the-border * 2);
}

#footer {
    color: (@base-color + #003300);
    border-color: desaturate(@red, 10%);
}

/* Compiled CSS */
#header {
    color: #333;
    border-left: 1px;
    border-right: 2px;
}

#footer {
    color: #114411;
    border-color: #7d2717;
}
```

Responsive web design

- Responsive web design is a template responding to various screen sizes, beginning with large PC screens and ending with tiny phones.
  ▷ Changing image sizes in response to screen resolution.

- ▹ Low-speed image maintenance on mobile devices.
- ▹ Simplifying page elements to use on mobile devices.
- ▹ Hiding unimportant elements on small screens.
- ▹ Providing bigger in size and easily clickable links and buttons for mobile users.
- ▹ Detecting and respond to mobile functions, such as device geolocation and orientation.
- ▹ The key to create responsive templates is the CSS media queries standard. Using media queries, you can create multiple CSS rule sets, which are applied only when the visible browser area is within certain width or height or within the range of the set width/height.

- `min-width: width`
  Applied if the window width is larger or equal to the template width.
- `max-width: width`
  Applied if the window width is less or equal to the template width.
- `min-device-width: width`
  Applied if the screen width of the device is larger or equal to the template. width.
- `max-device-width: width`
  Applied if the screen width of the device is less or equal to the template width.
- width is a browser window width. Width is, as a rule, less than screen width in browsers of regular PC. Nevertheless, width is usually larger than screen width on mobile browsers, as the majority of mobile browsers use virtual window,

and it is larger than screen size, which allows the user to zoom in and out images, as well as to move around the window area, dragging images. For example, Mobile Safari uses virtual window, its width equals 980px, although IOS device screen width, as a rule, equals from 320 to 768px (vertically). You will find out more about windows later.

- `device-width` is device screen width.
- The viewport meta tag. You can manage window size of your mobile browser by adding this tag to the head area.
- `initial-scale = 1`.
  This guarantees that the page will be fully enlarged so that the window width coincides with the device screen width in vertical mode: for example, 320px on iPhone and 768px on iPad.
- `maximum-scale = 1`.
  This prevents zooming in the page more than at the ratio 1:1, even when switching to the horizontal mode. In other words, it makes iPhone zoom in the window width up to 480px in the horizontal mode. If maximum scale is disabled, the window width is 320px, in both vertical and horizontal modes. Sometimes it is exactly what you need, but in case of the responsive template, this can lead to an overstretched content in the horizontal mode.
- Three parts of the responsive design:
  - ▹ Responsive modular grid
  - ▹ Responsive fluid images
  - ▹ Using media queries
  - ▹ Responsive modular grid. Essentially, it means that grid, which is traditionally measured in pixels, should

be considered as a percentage of the overall page width. Real calculated width of every column of a responsive web site changes every time the window size is changed and it cannot be the same on different devices.

- Fluid Grid. The formula to calculate proportion on a percentage basis: target / context = result. For example, we have a psd layout of 1000px in width. It has two blocks: one is on the left, 270px in width, which is 27%, the other is 730px, i.e. 73%.
- Fluid Images adjust their size to the parent block. The main idea is in unobvious property application { max-width: 100% }. Image with img { max.width: 100% }will never go beyond its parent block.
- If a parent block is less than img size, the image decreases proportionally. This principle is applicable to img, embed, object, and video.
- Mobile first is a technique when a site is laid out for devices of fewer possibilities, and then other possibilities are added using media queries.

Quick Check

- What responsive layout technologies do you know?
- How can you make an image fluid?
- What Media Queries attributes are there?
- Name three main parts of responsive design.

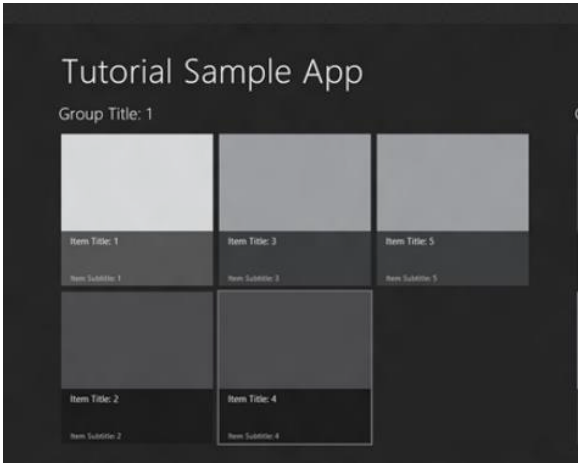
Additional Task

Using the LESS libraries, rewrite the main.css file in the 001_CSS_Problem example.

Individual Work

Task 1

Using Skeleton, lay out the page according to the following layout. When opening the page in mobile browser, all blocks should be aligned vertically.



Task 2

Redo styles of the page laid out in the previous task so that style used LESS.

Recommended Resources

LESS
http://lesscss.org.

Skeleton
http://www.getskeleton.com/.

Unit 11.
Creating pages with HTML5 and CSS3