Responsive HTML5 website

Creating the head – Doctype and Meta Tags

Defining the HTML5 doctype and creating our head section with the required scripts and CSS files.

<!DOCTYPE HTML>

In our head section let´s set the charset to UTF-8 in HTML5.

```
<meta charset="UTF-8">
```

As we would like to create a responsive design which should work on all kind of devices and screen resolutions we need to add the viewport meta tag which defines how the website should be displayed on a device. We set the width to device-width and the initial scale to 1.0. It means the page content will be displayed 1:1, an image with a size of 320px on a screen with 320px width would fill out the whole screen width.

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

Note: There are different opinions about using initial-scale and width=device-width. But It's that initial-scale=1.0 fits the viewport to the dimensions of the device because the size of the viewport fits the dimensions of the device regardless of its orientation. We recommend to use initial-scale alone, not associated width=device-width.

Creating the head – CSS files

We will use four different CSS files

The first stylesheet is called reset.css. What it does is resetting the styling of all HTML elements so that we can start to build our own styling from scratch without having to worry about cross-browser differences.

I have used Eric Meyer's "**Reset CSS**" 2.0, as given below or you can find here.

```
/**
 * Eric Meyer's Reset CSS v2.0 (http://meyerweb.com/eric/tools/css/reset/)
 * http://cssreset.com
 */
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
```

```
time, mark, audio, video {
	margin: 0;
	padding: 0;
	border: 0;
	font-size: 100%;
	font: inherit;
	vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
	display: block;
}
body {
	line-height: 1;
}
ol, ul {
	list-style: none;
}
blockquote, q {
	quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
	content: '';
	content: none;
}
table {
	border-collapse: collapse;
	border-spacing: 0;
}
```

The second stylesheet is called style.css and contains all our styling.

You can also divide this CSS file into two parts. First one is the basic layout styling and second for rest of the stylesheet.

If you look at the preview of our website and click on one of the small images in the main content section you will see that a larger version of the image will show up on top of the page. The script we use to achieve this is called **Lightbox2** and is an easy way to overlay images on top of the current page. For this tutorial you will need to download Lightbox2 and include the CSS file "lightbox.css".

The last stylesheet (**Google WebFonts**) will let us use the fonts Open Sans and Baumans.

To get fonts visit Google Web Fonts.

Note: Using multiple CSS and Javascript files the results in longer page loading time. You could combine all CSS and Javascript files into a single one.

Creating the head – Javascript / jQuery files

We are using HTML5 and CSS3 features we need several scripts to make the features visible in all browsers. The first script we will be using is **Modernizr.js**, a feature detection library for HTML5 and CSS3. It let´s you easily detect whether a browser supports a specific feature or not. Modernizr.js also comes with html5shiv which enables HTML5 elements in Internet Explorer. To download Modernizr.js click here and be sure "html5shiv" and "Modernizr.load" are checked.

The second script we need is **Respond.js**, a script which enables responsive design in Internet Explorer and other browsers which don´t support CSS Media Queries. Click here to download it from GitHub.

To be able to use i.e. lightbox.js we also need to include the jQuery library. The best way to do this is by letting Google host your jQuery file and use a fallback in case the Google jQuery file fails to load.

For Lightbox2 we need to include the lightbox.js located inside the js folder in the lightbox directory.

In this tutorial we will be using CSS3 features which require the use of different prefixes to work in all browsers. In order not to have to manually define the different prefixes each time we are using a CSS3 feature we will include the script **Prefix Free**, which automatically creates the required prefixes and let´s us write the unprefixed CSS properties. You can download the script from here.

The last script we will need is for the responsive Image Slider **SlideJS**. You can download the Plug-In from here and include the "jquery.slides.min.js" into your "js" folder.

Your file should now look like this. Note that in the course of this tutorial we will add further code to our head section.

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<title>HTML5 responsive website tutorial</title>

<link rel="stylesheet" href="reset.css" type="text/css" media="screen"
/>
<link rel="stylesheet" href="style.css" type="text/css" media="screen"
/>
<link href="lightbox/css/lightbox.css" rel="stylesheet" />
<link href='http://fonts.googleapis.com/css?family=Open+Sans|Baumans'
rel='stylesheet' type='text/css'/>

<script src="js/modernizr.js"></script>
<script src="js/respond.min.js"></script>


<!-- include extern jQuery file but fall back to local file if extern
one fails to load !-->
<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
<script type="text/javascript">window.jQuery || document.write('<script
type="text\/javascript"
src="js\/1.7.2.jquery.min"><\/script>')</script>
```

```
<script src="lightbox/js/lightbox.js"></script>
<script src="js/prefixfree.min.js"></script>
<script src="js/jquery.slides.min.js"></script></head>
```

Creating the body – Introduction

HTML5 offers several new elements which help us create more structured and accessible web pages. One of the main advantages of HTML5 is the possibility to structurize your web page and tell screen readers or search engine bots what is on a page which helps them to read only the parts they want.

Before we start coding let´s first have a look at the most important new sectioning elements in HTML5.

The new section element defines a part of a website with related content. It should not be used as a generic container element or for pure styling purposes. In that case rather use a simple div.

An article defines an independent piece of content which should be able to stand alone and still make sense.

A header defines a header for a document or a section.

A footer is used for defining the footer of a document or a section.

A nav defines a set of navigation links which should be the main navigation of your website.

An aside defines a section with secondary content. If an aside is placed within an article the content should be related to that specific article. If it is placed outside an article the content should be related to the site as whole.

Each of the sectioning elements in HTML5 should almost in any case contain a heading. By giving each section a specific heading you are creating a new section in the HTML5 outline. This is important as a proper outline does not only improve accessibility but is also good for SEO. To create a correct HTML5 outline you can use the Outliner Tool. In case you see an "Untitled Section" which corresponds to a section element in your HTML code you might want to specify a heading for the section. For further information on the document outlining algorithm be sure to check out Derek Johnson´s blog post.

If you have a look at the website we will be creating you will probably think: "Didn´t he just say that we need a heading for each sectioning element? I don´t see any heading for the header, the navigation, the footer or some of the sections.". Obviously displaying a heading for each of the sectioning elements does not really make any sense.

On the other hand we need to define headings for every section to create a proper outline. Not using sections in our document at all would somehow destroy the whole purpose of a HTML5 structured document.

A solution for this is to add headings for each section into your HTML code and then hide them with CSS. You should not hide them with display:none though as this would also remove them from the outline. Rather use the CSS property clip to hide the headings which is also the best way not to get banned by search engines for hiding content.

Creating the body – Building the website structure

Let´s start creating the HTML5 elements on the basis of our layout overview.

The first thing we need is a header. The header will include our navigation and our logo. Inside the header we put the main heading h1 and a paragraph. The main heading could be your website´s name or logo and the paragraph a motto or phrase describing the website. Depending on whether the subheading contains the main keywords and is an important part of the site you might in some cases also use a h2 instead. Apart from the heading and the paragraph let´s also put our navigation into the header. Inside the navigation section we put an unordered list with all the navigation links as list items. Your header code should now look like this.

```
<header>
<h1>Lingulo HTML5</h1>
<p>A responsive website tutorial</p>
<nav>
<ul>
<li><a href="javascript:void(0)">Home</a></li>
<li><a href="javascript:void(0)">Porfolio</a></li>
<li><a href="javascript:void(0)">About</a></li>
<li><a href="javascript:void(0)">Contact</a></li></ul></nav></header>
```

On the website structure image the next area below the header is called “Slider-Images”. As you can see when you look at the preview of our website this area will contain a large jQuery image slider which changes the image every 6 seconds. There are plenty of jQuery Sliders out there but in this tutorial we will use the responsive slideshow plug-in SlideJS which you should download now.

As you can see in the example files of SlideJS the Slider needs a new div called “slides” which contains the images you want to be slided. As we are going to show content boxes on top of the images we will wrap the div “slides” into a new section “container” and add the content for each image into a separate article which we call “slider_content1″, “slider_content2″, “slider_content3″… We are using IDs instead of classes here because we want to address each article with jQuery later on and need to be sure that each one appears only once on the page. The reason why we used articles for the first three sections and then a simple div for the slides is that the articles contain our independent content while the div only contains our three slider images and mainly has a styling purpose.

```
<section class="container">
<h2 class="hidden">Lorem Ipsum</h2>
<article id="slider_content1"></article>
<article id="slider_content2"></article>
<article id="slider_content3"></article>
<div id="slides">
<img src="img/slide1.jpg" alt="Some alt text">
<img src="img/slide2.jpg" alt="Some alt text">
<img src="img/slide3.jpg" alt="Some alt text"></div></section>
```

Now let´s add the slider content into our markup. “slider_content1″ will be the content which is shown when the image “slide1.jpg” is displayed, “slider_content2″ for “slide2.jpg” and so on. Inside these articles let´s add a heading, some teaser text and a read-more link. Note that we also add a heading with a class “hidden” into the section as we want to create a proper HTML5 outline. If you have problems deciding which heading to use on your website my recommendation is to try to use a heading structure with elements of the appropriate rank for the section’s nesting level. Read this article if you would like to know

more about proper use of headings in sections. You´re whole code for the slider should now look like this (I added some blind text into headings and paragraphs).

```
<section class="container">
<h2 class="hidden">Lorem Ipsum</h2>
<article id="slider_content1">
<h3>Lorem ipsum dolor?</h3>
<p>Weit hinten, hinter den Wortbergen, fern der Länder Vokalien und
Konsonantien leben die Blindtexte. Abgeschieden wohnen Sie in
Buchstabhausen an der Küste des Semantik.</p>
<a class="button" href="some_page.html">Mehr lesen</a></article>
<article id="slider_content2">
<h3>Nulla consequat</h3>
<p>Ein kleines Bächlein namens Duden fließt durch ihren Ort und
versorgt sie mit den nötigen Regelialien. Es ist ein paradiesmatisches
Land, in dem einem gebratene Satzteile in den Mund fliegen.</p>
<a class="button" href="some_page.html">Mehr lesen</a></article>
<article id="slider_content3">
<h3>Lorem ipsum</h3>
<p>Nicht einmal von der allmächtigen Interpunktion werden die
Blindtexte beherrscht – ein geradezu unorthographisches Leben.</p>
<a class="button" href="some_page.html">Mehr lesen</a></article>
<div id="slides">
<img src="img/slide1.jpg" alt="Some alt text">
<img src="img/slide2.jpg" alt="Some alt text">
<img src="img/slide3.jpg" alt="Some alt text"></div></section>
```

Now that we have the HTML structure of the image slider let´s move on to the next element on our structure image. As you can see the next thing to do would be the spacer. I called it spacer although it is not actually a spacer, it is just some orange part below the image slider with a short sentence and a search field. For the spacer we use a new section with the ID “spacer”. The reason I used an ID instead of a class for all the sections on the website is that I would like to be able to directly link to a specific part of the website. Inside the spacer we put a short sentence into a new paragraph and another div with the class “search”, which contains our search form.

```
<section id="spacer">
<h2 class="hidden">Dolor sit amet</h2>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit?</p>
<div class="search">
<form action="#">
<input type="text" name="sitesearch" value="Enter a word..."/>
<input type="submit" name="start_search" class="button"
value="Search"/></form></div></section>
```

You are doing great, we already have almost half of the whole HTML5 code we need. Take a look at the structure image again. You can see that from now on we have almost only “articles”, which are wrapped inside containers.

Let´s continue with the next section which contains three articles with a heading and an icon next to it. Also add a heading h3 and a paragraph p with some text to the articles. The br class=”clear” is used to clear the floating of the three articles which we will add into our CSS file in the next part of this tutorial.

```
<section id="boxcontent">
<h2 class="hidden">Adipiscing</h2>
<article>
<img src="img/some_icon.png" alt="alt text"/>
<h3>Lorem ipsum</h3>
<p>
Eines Tages aber beschloß eine kleine Zeile Blindtext, ihr Name war
Lorem Ipsum, hinaus zu gehen.</p></article>
<article>
<img src="img/some_icon.png" alt="alt text"/>
<h3>Consectetuer</h3>
<p>
Der große Oxmox riet ihr davon ab, da es dort wimmele von bösen Kommata,
wilden Fragezeichen...</p></article>
<article>
<img src="img/some_icon.png" alt="alt text"/>
<h3>Dolor sit amet</h3>
<p>
Es packte seine sieben Versalien, schob sich sein Initial in den Gürtel
und machte sich auf den Weg.</p></article>
<br class="clear"/></section>
```

Now let´s create our next section again with a h2 heading and four articles in it. Inside the articles we put a hidden heading h3 and a new HTML5 element called figure. The figure element is used in conjunction with the figcaption element to mark up images or diagrams (among others) and should be an independent part which can be moved away from the main flow of the document. Create a figure element and add an image and a figcaption to it.

If you move your mouse over the images in the website preview you will see that they will become dark and there will be a magnifier icon on top. When clicking on an image Lightbox2 will open the image up in a larger version. We have already included the Lightbox2 script into our head part. To enable image resizing we now need to create an anchor around our image and give it the attribute rel=”lightbox”. The href attribute should point to the large image version.

The effect with the magnifier icon on mouse hover is created purely with CSS3 features. The HTML element used to create the half transparent black layer is a span element wrapped inside the anchor. We will style the span in the second part of this tutorial to make it invisible by default and only show it on mouse hover.

Your code for the section “four_columns” should look like this.

```
<section id="four_columns">
<h2>
Lorem ipsum</h2>
<article class="img-item">
<h3 class="hidden">Dolor sit amet</h3>
<figure>
<a href="img/example-slide-1.jpg" rel="lightbox" title="Some Title">
<span class="thumb-screen"></span>
<img src="img/example-slide-1sml.jpg" alt="Some alt text"/></a>
<figcaption>
<strong>
```

```
Photo by: Some Name</strong>
Als es die ersten Hügel des Kursivgebirges erklommen
hatte</figcaption></figure></article>
<article class="img-item">
<h3 class="hidden">Sit Amet</h3>
<figure>
<a href="img/example-slide-2.jpg" rel="lightbox" title="Some Title">
<span class="thumb-screen"></span>
<img src="img/example-slide-2sml.jpg" alt="Some alt text"/></a>
<figcaption>
<strong>
Photo by: Some Name</strong>
warf es einen letzten Blick zurück auf die Skyline seiner Heimatstadt
Buchstabhausen</figcaption></figure></article>
<article class="img-item">
<h3 class="hidden">Dolor Sit</h3>
<figure>
<a href="img/example-slide-3.jpg" rel="lightbox" title="Some Title">
<span class="thumb-screen"></span>
<img src="img/example-slide-3sml.jpg" alt="Some alt text"/></a>
<figcaption>
<strong>
Photo by: Some Name</strong>
die Headline von Alphabetdorf und die Subline seiner eigenen
Straße.</figcaption></figure></article>
<article class="img-item">
<h3 class="hidden">Lorem Ipsum</h3>
<figure>
<a href="img/example-slide-4.jpg" rel="lightbox" title="Some Title">
<span class="thumb-screen"></span>
<img src="img/example-slide-4sml.jpg" alt="Some alt text"/></a>
<figcaption>
<strong>
Photo by: Some Name</strong>
Wehmütig lief ihm eine rhetorische Frage über die
Wange.</figcaption></figure></article>
<br class="clear"/></section>
```

Next thing we need is another section with a hidden heading, an article and another section in it. The second section contains a hidden heading and two articles which each contain also a hidden heading. Give the outer section an ID, for example “text_columns” and each of the columns a class, for example “column1″ and “column2″. We could of course also not use classes and address the article and section directly. The reason I like to use classes though is that it makes it easier in case you want to change the HTML code later on without having to change the CSS code.
Inside the first column put a heading h3 and a paragraph p with some text. Into the second column we will put two articles or divs depending on whether the content is relevant and independent or not. Inside each of these two wrappers add a paragraph p and an image. The images need a class “rocket” and “clock” and will be animated later on in the second part of this tutorial.

```
<section id="text_columns">
<h2 class="hidden">Blindtext</h2>
<article class="column1">
```

```
<h3>
Dolor sit amet</h3>
<p>Unterwegs traf es eine Copy. Die Copy warnte das Blindtextchen, da,
wo sie herkäme wäre sie zigmal umgeschrieben worden und alles, was von
ihrem Ursprung noch übrig wäre, sei das Wort "und" und das
Blindtextchen solle umkehren und wieder in sein eigenes, sicheres Land
zurückkehren.</p>
<p>Doch alles Gutzureden konnte es nicht überzeugen und so dauerte es
nicht lange, bis ihm ein paar heimtückische Werbetexter auflauerten, es
mit Longe und Parole betrunken machten und es dann in ihre Agentur
schleppten, wo sie es für ihre Projekte wieder und wieder
mißbrauchten.</p></article>
<section class="column2">
<h3 class="hidden">Lorem Impsum</h3>
<article class="row">
<h4 class="hidden">Dolor sit</h4>
<img src="img/icon.png" width="80" class="rocket" alt="Some alt text"/>
<p>Und wenn es nicht umgeschrieben wurde, dann benutzen Sie es
immernoch.</p></article>
<article class="row">
<h4 class="hidden">Ipsum</h4>
<img src="img/icon.png" width="80" class="clock" alt="Some alt text"/>
<p>Doch alles Gutzureden konnte es nicht überzeugen und so dauerte es
nicht.</p></article></section></section>
```

We have already reached the footer of our website. Into the footer we are going to put not only a hidden heading and a section with three articles for the upper three columns of the footer but also another section with the ID “copyright” which we are going to position at the bottom of our footer later in the second part of this tutorial. Into the “copyright” section we add a div with the class “wrapper” which will make sure the content is centered and does not exceed the maximum width (we are going to talk about that in the second part more detailed). The reason we are using a div here instead of an article is that in this case all we need is a container for styling purposes, there is no actual related and independent content inside. Into the “wrapper” div we put a copyright note as blank text and another div with the class “social” which contains all the social links.

The section which we just created at the beginning of the footer again gets a hidden heading and contains three articles with the class “column”. To the second and third article we need to assign another class so we can later on target each article separately in CSS. Add the class “midlist” to the second article and the class “rightlist” to the third one. To do so simply add a blank space after “column” and put the other class behind (<article class="column midlist">). Into the articles we need to add some content, into the first one let´s just put a heading and a text. Into the one in the middle and the one in the right also put a heading and an unordered list with a link inside each of the list items. Each link should be wrapped around an image and a span element. The span contains our text that we want to display. We are using a span to be able to position the text in any way we want and display it as a block element (more about that later).

```
<footer>
<h2 class="hidden">Our footer</h2>
<section id="copyright">
<h3 class="hidden">Copyright notice</h3>
<div class="wrapper">
<div class="social">
<a href="javascript:void(0)"><img src="img/Social-Networks-Google-plus-
```

```
icon.png" alt="google plus" width="25"/></a>
<a href="javascript:void(0)"><img src="img/Logos-Tumblr-icon.png"
alt="tumblr" width="25"/></a>
<a href="javascript:void(0)"><img src="img/Logos-Youtube-icon.png"
alt="youtube" width="25"/></a>
<a href="javascript:void(0)"><img src="img/Social-Networks-Bebo-
icon.png" alt="bebo" width="25"/></a>
<a href="javascript:void(0)"><img src="img/Social-Networks-Xing-
icon.png" alt="xing" width="25"/></a></div>
&copy; Copyright 2012 by <a href="http://www.lingulo.com">Lingulo</a>.
All Rights Reserved.</div></section>
<section class="wrapper">
<h3 class="hidden">Footer content</h3>
<article class="column">
<h4>Lorem ipsum</h4>
Unterwegs traf es eine Copy. Die Copy warnte das Blindtextchen, da, wo
sie herkäme wäre sie zigmal umgeschrieben worden und alles, was von
ihrem Ursprung noch übrig wäre, sei das Wort "und" und das
Blindtextchen solle umkehren und wieder in sein.</article>
<article class="column midlist">
<h4>Consectetuer</h4>
<ul>
<li><a href="javascript:void(0)">Die Copy warnte das
Blindtextchen</a></li>
<li><a href="javascript:void(0)">Unterwegs traf es eine Copy</a></li>
<li><a href="javascript:void(0)">Doch alles Gutzureden konnte</a></li>
<li><a href="javascript:void(0)">Als es die ersten
Hügel</a></li></ul></article>
<article class="column rightlist">
<h4>Dolor sit amet</h4>
<ul>
<li><a href="javascript:void(0)"><img src="img/example-slide-1sml.jpg"
width="80" alt="some alt text"/><span>Unterwegs traf es eine Copy. Die
Copy warnte.</span></a></li>
<li><a href="javascript:void(0)"><img src="img/example-slide-2sml.jpg"
width="80" alt="some alt text"/><span>Doch alles Gutzureden konnte es
nicht.</span></a></li>
<li><a href="javascript:void(0)"><img src="img/example-slide-3sml.jpg"
width="80" alt="some alt text"/><span>Und es dann in ihre Agentur
schleppten.</span></a></li></ul></article></section></footer>
```

Creating the styling – Defining the global rules

When creating the stylesheet it is important to always use comments for the different sections of the CSS file. The first section is the global or general section where we will put all stylings which are used on your whole website and mostly remain unchanged. The first element we are gonna style is the body element. As you can see on our website preview there is an orange stripe on top of the page. This stripe is created by a 5px thick top border defined in our body element rule. Let´s also define our website background color and our general font family and font color.

```
body
{border-top: 5px solid #e56038;
background: #ebe8de;
font-family: 'Open Sans', sans-serif;
color: #333333;}
```

Next element we want to style is our input fields. As we would like the input fields to look the same on any further page of our website we will put the input styling into our global section of the CSS file. In our website example we only have two different kinds of input fields: a text input field and a submit button. Due to that reason we will create a general input rule which will cover the styling for all the different kinds of input fields and a second rule only for our submit button.

If you look at the CSS code for our input fields you will probably note that we are using CSS3 features without defining prefixes for the different browsers (-webkit, -moz etc.). As already mentioned in the last part of this tutorial we use a script called **Prefix Free** which automatically rewrites the CSS rule depending on the browser used by the visitor. That way we can use CSS3 properties by simply adding the general rule (e.g. just box-shadow instead of -moz-box-shadow, -webkit-box-shadow and box-shadow).

```
input
{font-family: 'Open Sans', sans-serif;
font-size:16px;
padding: 7px;
outline: 0;
border:0;
width:250px;
background: #EBE8DE;
border-radius:5px;}

input[type=submit]
{width:auto;
padding: 5px 18px;
line-height:25px;
text-shadow:none;
cursor:pointer;
box-shadow: none;
background: #333333;
color: #fff;}
```

Next let´s add rules for our paragraphs and anchors. As we have removed all styling by using the reset.css we now need to define our margin (or padding) and line-height manually for the paragraphs. The anchors should inherit their font color from the parent element and change it by fading it from one color to another. To do so we use the CSS3 feature transition (for further information on transitions please read my blog post). Since we have removed all predefined CSS styling we also need to define our strong elements. In our case they should just be displayed in bold.

```
p
{margin: 5px 0;
```

```
line-height: 25px;}

a
{text-decoration: none;
color: inherit;
transition: color .5s ease;}

strong
{font-weight: bold;}
```

We have three rules left in our general section of the CSS file. The first one is the rule for our figcaption element. As mentioned in the previous part of the tutorial a figcaption is useful in conjunction with the figure element to mark up images or diagrams. In our website the figure element is used for our four little images. To emphasize the heading of our caption a bit more and to add the line between caption title and caption text we will now add a CSS rule for our strong element inside the figcaption.

```
figcaption
{line-height: 25px;
font-size:14px;
width:200px;}

figcaption strong
{border-bottom: 1px solid #D6D0C1;
padding-bottom:10px;
margin: 10px 0;
display:block;}
```

Creating the styling – Defining the section rules

Now that we have finished creating the general CSS rules let´s start defining the rules for our different sections of the website from top to bottom. To make our website responsive we will set the width property to auto and define a max-width of e.g. 900px for each section on our website. What this does is leaving the sections at 900px width when the browser window is wider than 900px. However when the browser width gets smaller than 900px the width of our sections will adapt to the browser´s width.

As we would like the sections to be horizontally centered we will set the margin-right and margin-left property to auto (see an example here). Centering elements like that is however not supported in Internet Explorer 5. As a solution you could set the body element´s text-align property to center. I personally won´t even bother as IE 5 is almost not used anymore at all.

Let´s start now creating the rules for our different sections of our website. The first section to style is our header. On a real website with several pages you might have multiple header elements so you would need to give the main header a class or an ID to create specific rules only for that element. In our case though since we only use one header element on our entire website we will just add a rule for all header elements and a rule for all paragraphs and h1 headings inside a header.

```
header
{
position:relative;
width:auto;
max-width:900px;
```

```
margin: 0 auto 20px auto;}


header h1
{margin: 35px 0 0 0;
font-size: 55px;
color: #e56038;
font-family: 'Baumans', cursive;}


header p
{font-family: 'Open Sans', sans-serif;
font-size: 16px;
color: #4A463B;
margin-left:132px;}
```

For our main navigation we used the new HTML5 nav element. Since we have added the CSS property position: relative to the header element we can now use position: absolute to position the navigation at the right bottom corner of our header.

For our unordered list which contains all the navigation links as list items we will remove the default list style (the little dot next to each list item). Our list items should not be positioned one below the other but next to eachother. To do so we will make each list item float and add a padding to create a space between each of them. For the anchors within our list items we will define a font type and color and set text-transform to uppercase to make all letters uppercase. Also add a transition of 0.25 seconds and define a color in the hover state. This way the links will fade from one color to another when hovering them.

```
nav ul
{list-style:none;}


nav ul li
{display:block;
float:left;
padding:3px 15px;}


nav ul li a
{font-family: 'Open Sans', sans-serif;
text-transform: uppercase;
transition: all .25s ease;}


nav ul li a:hover
{color: #E56038;}
```

The next section to style is our slider-image section (don´t forget to write the section as a new comment into your CSS file to make it more clear). Just like in the example file of the Slides script we set the container width to auto to make the slider images resize depending on the browser´s width. The div “slides” needs to be hidden with CSS as it will be made visible by the Slides script. Also add a border to the div with the class “slidesjs-container” to prevent the slideshow from flashing on load. Our slider content is positioned absolutely within our slider container and gets a z-index higher than the one from our container to make it overlay the slider images. As we want to show only slider_content1 at the beginning when loading the site we will set display to none for all slider content boxes except of slider_content1.

```
#slider_content1, #slider_content2, #slider_content3
{line-height: 25px;
font-family: 'Open Sans', sans-serif;
width:350px;
position:absolute;
top:15%;
left:15%;
display:none;
z-index:11;}

#slider_content1
{display:block;}

.container
{width:auto;
margin: 0 auto;
position:relative;}
```

Now that we have styled the wrapper for our slider content let´s add a styling to the headings and paragraphs inside our slider_content1, slider_content2 and slider_content3. To create the effect of two rounded borders simply add 0 to the borders you don´t want to be rounded into our border-radius property. The order for the values inside that property are top-left, top-right, bottom-right, bottom-left.

For our property display we choose inline-block which displays our headings inline but will preserve the capability to style them just like block elements (setting width and height, margin, padding and so on). The advantage over just using block instead in our case is that the dark background of the heading will be as wide as the heading text itself and not as the surrounding element.

For the paragraphs we will define the same padding as for the heading to make it look in a line and set a bottom margin to create a space between the paragraph and the “Read more” button. You can also set a small border-radius to make the borders of the paragraphs a bit round.

```
#slider_content1 h3, #slider_content2 h3, #slider_content3 h3
{color:#EBE8DE;
font-size:25px;
font-weight:bold;
margin-bottom:10px;
background-color:#333333;
padding:10px 15px;
border-radius: 15px 0 15px 0;
display:inline-block;}


#slider_content1 p, #slider_content2 p, #slider_content3 p
{margin:0 0 30px 0;
color:#4a463b;
background-color:#EBE8DE;
padding:10px 15px;
border-radius: 5px;}
```

Now that we have finished the CSS part of our slideshow let´s move on to the next section, our orange "spacer". We set the width to auto to make it fill the whole width of the surrounding element (which is our body element).

To position the question text which is wrapped inside a paragraph we can´t just define a fixed procentual value for our left property. If we did so the question would move to different positions depending on the browser´s width and would not stay fixed at a certain position. The solution is to set the paragraph´s position to absolute and the left property to 50% which positions the paragraph in the middle of our spacer. Then add a negative margin-left to move it a bit more to the left side. This way the paragraph will always stay at the same position regardless of the browser´s width. To position our search form we do the same thing. Note that we don´t need to style our form elements as we have already added a general styling in the global/general part of our CSS file.

```
#spacer
{width:auto;
height:70px;
background-color:#e56038;
position:relative;
font-family: 'Open Sans', sans-serif;
color:#fff;
font-size:18px;}

#spacer p
{margin-top:22px;
width:auto;
position:absolute;
left:50%;
margin-left:-450px;}

#spacer .search
{margin-top:15px;
width:auto;
position:absolute;
right:50%;
margin-right:-480px;}
```

Our next section consists of three articles each with an image, a heading and a paragraph. Again we need to set the width of our "boxcontent" to auto to make the width respond to the browser´s width and set a max-width to limit the maximum width. To make the section centered we again use an auto margin to the left and to the right. To create a little space to the top and bottom we set a padding. As we want the articles to be positioned next to each other let´s make them float, set a fixed width and create a little space between them by setting a margin-right. Inside the articles we let the icon image float to the left in order for the text to be placed directly next to the icon and not below it. By floating the icon the heading will now be placed next to the icon image but the text which is placed in a paragraph will only stay next to the image as long as the image is still there. As soon as the text exceeds the height of the image it will continue at the left border of our article below the icon. To fix this we need to define a margin-left for the paragraph to make it vertically in a line with the heading.

```
#boxcontent
{width:auto;
max-width:900px;
margin:0 auto;
```

```
padding:70px 0 45px 0;}

#boxcontent article
{float:left;
width:250px;
margin-right:45px;
font-size:14px;}

#boxcontent article h3
{font-family: 'Open Sans', sans-serif;
font-size:20px;
margin-bottom:10px;
margin-left:75px;}

#boxcontent article img
{float:left;}

#boxcontent article p
{line-height:25px;
font-family: 'Open Sans', sans-serif;
margin-left:75px;}
```

For the next two sections we will combine two rules in order not to have to write it two times. For both the section “four_columns” as well as the section “text_columns” we set the font, the line height, the width etc. as well as define the styling of the h2 and h3 headings.

```
#four_columns, #text_columns
{line-height:25px;
font-family: 'Open Sans', sans-serif;
width:auto;
max-width:900px;
margin:0 auto;}

#four_columns h2, #text_columns h3
{font-size:20px;
border-bottom: 1px solid #D6D0C1;
padding: 20px 0;
margin-bottom: 20px;}
```

Just like in the last section we also let the articles within “four_columns” float to position them next to each other. To create the effect when hovering the thumbnail images we will first set display to block for the anchors inside our articles. We want them to be displayed as block elements as they should adapt its own size to the size of the elements inside of it and fill the whole height and width of our thumbnail image. Also set the position to relative in order to be able to position the span element “thumb_screen” absolutely inside of it. This span element will be displayed on top of our image and shows the little zoom icon and a dark background on mouse-hover. We need to position it absolutely in order to make it lay on top of the image. With a relative position the span element would be displayed above the image. To create the half transparent background we will set the background color to black and add the zoom icon as background image (vertically and horizontally centered). Then we set the opacity to 0 to hide the span element by default. To let the span element fade from invisible to visible we will add the CSS3 feature transition (more about it in my other blog post) for the property opacity. In the hover-state of the span element we set the opacity to 0.5 to make it 50% transparent on mouse-hover.

```
#four_columns .img-item
{float:left;
margin-right:25px;}

#four_columns .img-item a
{position:relative;
display:block;}

.thumb-screen
{display:block;
position:absolute;
top:0;
left:0;
width:100%;
height:113px;
background: #000 url(img/zoom.png) center center no-repeat;
z-index:99;
opacity: 0;
transition: opacity .5s ease;}

.thumb-screen:hover
{opacity:0.5;}
```

The next section consists of two columns, one article and one section which should float next to each other. Each of the two columns should have half the width of the website width of 900px. To move the second column a bit more down to be vertically centered with the first column we define a margin. The section "column2″ contains two articles which should each have a relative position in order to be able to absolutely position the rocket and clock icons.

```
#text_columns article.column1, #text_columns .column2
{margin: 70px 0;
font-size:14px;
float:left;
width:450px;}

#text_columns .column2
{margin: 120px 0;}

.row
{position:relative;
margin: 40px 0 0 50px;
float:right;
width:350px;}
```

As a little gimmick we will now create the CSS animation which starts when hovering the rocket and clock icon. Whether or not this is a useful feature depends on your site and what you would like to achieve with the animation. Note that CSS animations will not be supported by every browser and might in some cases lead to unexpected behavior (for example might the rocket in some browsers disappear on mouse-hover).

When hovering the rocket we want it to fly to the top right and fade out. So let´s set the opacity to 0 on mouse-hover and define a transition for the opacity. We want it to fade out within 0.4 seconds after a delay of 0.2 seconds. To do so add the delay as last value into our transition property. Now that the rocket will

fade out on mouse-hover let´s add the path animation. In order not to have to set all the keyframes manually we will use the tool Stylie to create a path from the bottom left corner to the right top corner. Simply copy the CSS code inside the class “stylie” generated by the tool into our rocket:hover rule. Then create a new section “keyframes” at the end of your CSS file where you add the keyframe rule (the part starting with “@keyframes stylie-transform-keyframes”). You might need to adapt the translateX or translateY property to match the starting point of your rocket with the rocket icon (if you don´t change the values manually your rocket might start not exactly from where the rocket image was before hovering it). Just play around with the values to make it look somehow realistic. Also be sure to set the value for animation-iteration-count to 1 so that the animation will occur only once on mouse-hover and the rocket doesn´t fly infinetely.

For the clock icon we want the animation to look just like a ringing alarm clock. To achieve this we will again use Stylie to create a horizontal path and then set the animation-duration to about 100ms. Also set the animation-iteration-count to infinite to make the animation repeat endlessly. Please note that we need to change the name of our keyframe rule as the name “stylie-transform-keyframes” has already been used for our rocket keyframes.

```
.rocket:hover
{opacity:0;
transition: opacity 0.4s ease 0.2s;
animation-name: stylie-transform-keyframes;
animation-duration: 700ms;
animation-delay: 0ms;
animation-fill-mode: forwards;
animation-timing-function: linear;
animation-iteration-count: 1;
transform-origin: 0 0;}

.clock:hover
{animation-name: stylie-transform2-keyframes;
animation-duration: 100ms;
animation-delay: 0ms;
animation-fill-mode: forwards;
animation-timing-function: linear;
animation-iteration-count: infinite;
transform-origin: 0 0;}


/* KEYFRAMES */

@keyframes stylie-transform-keyframes
{0%
{transform:translateX(30px) translateY(46px) rotate(0deg) translate(-
50%, -50%);
animation-timing-function: cubic-bezier(.25,.25,.75,.75);}
100%
{transform:translateX(260px) translateY(-150px) rotate(0deg)
translate(-50%, -50%);}}

@keyframes stylie-transform2-keyframes
{0%
{transform:translateX(40px) translateY(40px) rotate(0deg) translate(-
```

```
50%, -50%);
animation-timing-function: cubic-bezier(.25,.25,.75,.75);}
100%
{transform:translateX(50px) translateY(40px) rotate(0deg) translate(-
50%, -50%);}}
```

We have reached the footer of our website. As we want it to fill the whole width of the body element and have a fixed height we set the width to auto and define a height value. Also be sure to add position:relative again in order to be able to position the copyright section at the bottom of our footer.

Inside our footer we have another section with a class "wrapper" which should have an auto width and a max-width of 900px just like the other sections in our website. This will make sure that the footer content will change its width depending on the browser´s width.

The three columns inside the footer should be next to each other so we will float them and set a different font color in order for the text to be visible on the dark background. The column in the middle (second column) consists of several list items with a little arrow icon. Each of the list items has a padding and a border-bottom. To move the list items a bit down and create a symmetric look we use margin-bottom. For the arrow icon we set a background image and position it left and 6px from the top. As we have defined a padding the text will not overlap the icon.

```
footer
{position:relative;
clear:both;
width:auto;
height:350px;
background:#333333;}

footer .wrapper
{line-height:25px;
margin: 0 auto;
padding-top:30px;
width:auto;
max-width:900px;
font-size:14px;}

footer .wrapper .column
{font-family: 'Open Sans', sans-serif;
color:#ababab;
float:left;
width:280px;
margin-right:20px;}

footer .wrapper .column.midlist ul li
{width:auto;
padding:0 0 10px 25px;
margin-bottom:10px;
border-bottom: 1px solid #444444;
background:url(img/arrowright2.png) left 6px no-repeat;}

footer .wrapper .column.midlist ul li a:hover
{color:#fff;}
```

In the right column we will set display to block for the span elements inside the list items. We want the text inside our span element to be displayed as block element in order to create a margin between the little image and our text. We float our image in order to position it next to the text. Since our span element containing the text is displayed as a block element we need to define a margin-left starting from the left border of our ul element to create a space between the image and the text. For our images we will define a border and a transition for our border which will make sure the border color fades from one color to another on mouse-hover.

For our h4 headings inside the footer we set a font color which looks good on the dark background and define a bottom border just like we did for the previous headings.

```
footer .wrapper .column.rightlist ul li
{width:auto;
margin-bottom:15px;}

footer .wrapper .column.rightlist ul li a span
{margin-left:95px;
display:block;}

footer .wrapper .column.rightlist ul li a img
{transition: border .25s ease;
float:left;
border:3px solid #444444;}

footer .wrapper .column.rightlist ul li a img:hover
{border-color: #5e5e5e;}

footer .wrapper .column h4
{font-size: 16px;
color: #fff;
border-bottom: 1px solid #444444;
padding: 0 0 10px 0;
margin-bottom: 10px;}
```

Now that we have styled the footer let´s position our copyright notice at the very bottom of our footer. If you have a look at the HTML code you will see that we have a div with the ID “copyright” which we will position absolutely at the bottom of our footer and give it a slightly darker background color. The width of this div should be similar to the width of our footer so we will set the width to 100%.

Inside our copyright section we have a div with the class “wrapper”. We do not have to set the width or the max-width for it as we have already defined a rule footer .wrapper which will apply. However we need to set the font color and font type, define a padding and set the position to relative to be able to position our social button links absolutely within the wrapper. The div with our social links should be positioned at the right corner and 25px from the top of our wrapper.

If you have a look at the HTML code you will see that our social buttons are simply made by wrapping an anchor around an image. As the links are positioned next to each other we need to make them float just like we did several times before. To achieve the hover effect we will simply set the opacity to about 30% and then to 70% or even more on mouse-hover. To make it fade smoothly from 30% to 70% opacity we define a transition.

```
#copyright
{background: #1D1D1D;
height:70px;
position:absolute;
bottom:0;
left:0;
width:100%;}

#copyright .wrapper
{font-family: 'Open Sans', sans-serif;
padding-top:25px;
color: #5e5e5e;
font-size:14px;
position:relative;}


#copyright .wrapper .social
{position:absolute;
right:0;
top:25px;}

#copyright .wrapper .social a
{transition: opacity .25s ease;
opacity: 0.3;
margin-left: 12px;
display:block;
float:left;}

#copyright .wrapper .social a:hover
{opacity: 0.7;}

#copyright .wrapper a
{color: #ABABAB;}

#copyright .wrapper a:hover
{color: #fff;}
```

The next section in our CSS file is called “MISC” and will contain different rules which you don´t know where to put them else. For our website we need to define two rules inside this section. In the HTML code you will find several br with a class “clear”. These are necessary to clear the floating which we used to horizontally position elements next to each other. Simply try not setting the class and you will see why we need it.

The second rule we still need to define is for our hidden headings. As mentioned in the previous part of this tutorial we need to define a heading for each section in our document to create a proper HTML5 outline. Since displaying these headings does not always make sense we will hide all elements which have the class “hidden” by using the CSS property clip.

```
.clear
{clear:both;}

.hidden
```

```
{position:absolute;
clip: rect(1px 1px 1px 1px); /* IE6 & 7 */
clip: rect(1px, 1px, 1px, 1px);}
```

Creating the styling – Making it responsive

We have finished styling our website and it should look just like in the website preview. The only difference is that the website we just created is not yet entirely responsive. We have already set our width to auto and defined a max-width for our sections. However this will only change the width of our sections but the content inside these sections might be wrong positioned or overlapping. We will now use CSS Media Queries to specify certain rules for different browser window sizes. I will not cover the topic media queries in detail, if you want to know more about this topic check out the web developer guide. In our website we will only use the max-width attribute for our media queries which defines the maximum width for the rule to take effect. So if you define a rule with a max-width of 500px then this rule will take effect as long as the viewport width is smaller than 500px or exactly 500px.

To make our website responsive we need to first of all think about what content is relevant and should be displayed on small displays like mobile phones or on tablets. When creating a website you need to already have in mind how it should look on a mobile phone at the very beginning of your development phase.

Start reducing the width of your browser window until you detect that some parts are wrong positioned or just don´t look good. In that case the easiest way to determine the current width of the website is to use your browser´s web development tool to get the width of the body element. For that width we then need to define a media query to somehow change the layout of our website and make it look good again. However note that the width defined in our media queries can vary depending on the browser you are using. In Firefox the vertical scrollbar gets included in the viewport width which means that if you define a media query with a max-width of 1000px then in Firefox the media query rules will occur at about 985px. The 15px wide scrollbar is subtracted from the max-width. To read more about that topic check out this blog post. The solution for this problem in our case would be to just use a buffer and simply set our max-width a bit higher than necessary.

On our website the first thing you will notice when changing the width of your browser window to about 1215px is that the “Read more” button in our slider is likely to overlap the spacer. How can we get rid of this problem? One way is to completely remove the button (set display to none) and put the “Read more” behind our paragraph as plain text. That way we will already save some space. To do so we still need to add some HTML code to our file. Add an anchor behind each of the paragraphs inside our slider content boxes and give it a class “responsive_button”. Then add a rule into your CSS file to set display to none for all elements with a class “responsive_button” inside a paragraph inside a slider content box.

```
#slider_content1 p .responsive_button, #slider_content2
p .responsive_button, #slider_content3 p .responsive_button
{display:none;}
```

In our media query rule we now need to simply set display to inline to display it inline behind the paragraph text and define a font color.

However there is still a problem since the text inside our paragraphs is still overlapping the orange spacer. What we could do here to solve this is to make the slider content wider. To do so we set its width to auto to make it adapt to the parent element and give it a margin-right to create a free space between the slider

content boxes and the right edge of our slider images. Also we want the font size to be smaller which will save some space and looks a bit lighter.

```
@media (max-width: 1215px)
{#slider_content1, #slider_content2, #slider_content3
{width:auto;
margin-right:50px;}
#slider_content1 h3, #slider_content2 h3, #slider_content3 h3
{font-size:18px;}
#slider_content1 p, #slider_content2 p, #slider_content3 p
{font-size:14px;}
#slider_content1 p .responsive_button, #slider_content2
p .responsive_button, #slider_content3 p .responsive_button
{display:inline;
color:#000;}
.container .button
{display:none;}}
```

When reducing the width even more you will notice that when we get below the max-width of 900px which we have set for our website the floated articles will not be next to each other anymore. To get rid of the Firefox scrollbar problem I told you about we will add 15px and define a media query for a max-width of 915px. Since thearticles don´t fit next to each other anymore we will position them one below the other horizontally centered. To do so we need to remove the floating and set margin-right and margin-left to auto. Also we change the width to 60% to make it change its width depending on the website´s width.

We also have the problem that below 900px the orange spacer with the search form will not fit anymore. To solve this let´s change the font size of the spacer and the input field, reposition the paragraph with the question text inside and change the size of our input field by adjusting the padding values.

Another problem occurs with the section “four_columns”. Below 900px some of the articles with the class “img-item” might wrap and overlap the other articles. To solve this problem we will set a specific width to our section so that two articles wrap to the next line. Then we define a margin-top for the two wrapped articles to create a space between the two lines. To address only the third and fourth article (which are the ones in the second row) we will use the CSS3 feature nth-of-type. Note though that this will not work in IE8 or below. If you want to assure full browser support you could give a specific class to the third and fourth article instead.

For the section text_columns we also need to reposition the articles. To do so let´s remove the floating and change the width of the two columns by overwriting the value for max-width. To center the articles within the section we will set margin-left and margin-right to auto just as we did several times before. We also don´t want the two articles inside the second column (the ones with the rocket and the clock icons) to be floated right anymore and need to readjust the margin to make it look good.

The last section we need to adjust at 900px is the footer. To make it fit the smaller width of our website we will reduce the font size and the width of the three columns.

```
@media (max-width: 915px)
{#boxcontent article
{float: none;
margin: 30px auto 0 auto;
width: 60%;}
```

```
#spacer
{font-size:15px;}

#spacer .search
{margin-top:19px;
margin-right:-385px;}

#spacer p
{margin-left:-370px;}

input
{padding:4px;
font-size:14px;}

input[type="submit"]
{padding: 1px 14px;}

#four_columns
{width: 500px;}

#four_columns .img-item:nth-of-type(3), #four_columns .img-item:nth-of-
type(4)
{margin-top: 25px;}

#text_columns article.column1, #text_columns .column2
{float:none;
max-width: 500px;
margin: 50px auto 0 auto;}

.column2 .row
{float:none;
margin:0 0 40px 50px;}

footer .wrapper .column
{font-size: 12px;
width: 230px;}}
```

When reducing the width of your browser even more you will notice that at some point the slideshow is just too small and does not look good anymore. To remove it we will set the top value to a high negative value to make it disappear. We can not use display to remove it because the slideshow will not appear again if it has been removed. As the slideshow has disappeared the orange spacer will start overlapping our navigation so we need to define a fixed height for our header. We can also reposition our navigation because now that everything has been centered it might look better to center the navigation as well. To center it we first need to overwrite our top and bottom values by setting them to auto. Then we set the left attribute to 50% and define a negative margin corresponding to half of the navigation´s width.

We also got some issues concerning the spacer. The search form and the paragraph do not fit next to each other anymore so we need to position them below each other. We define a fixed height for the spacer and setmargin-right and margin-left to auto to center it. To get rid of any margin and the left value we will set the position to static. By defining a padding we create some free space above and below the paragraph. Just as we did for the paragraph we will do the same for our search form.

Because our footer does not fit anymore we will position the three columns of our footer below each other. To achieve this we first need to set the height to auto as it has been set to 350px before. Then we set a fixed width to the wrapper inside the footer and center it. The three columns inside the wrapper should not float anymore as they should be positioned below each other. We also set the width to auto in order to make it fill out the whole width of the wrapper. Since there is not a lot of space in the footer we will completely remove the social links.

```
@media (max-width: 765px)
{.container
{top: -1500px;}

header
{height:120px;}

header nav
{right: auto;
bottom: auto;
left: 50%;
top:100px;
margin-left: -184px;}

#spacer
{height:100px;}

#spacer p
{text-align:center;
position:static;
margin: 0 auto;
padding:15px 0 7px 0;}

#spacer .search
{text-align:center;
position:static;
margin: 0 auto;}

footer
{padding-bottom:70px;
height: auto;}

footer .wrapper
{width: 350px;
margin: 0 auto;}

footer .wrapper .column
{margin-top:30px;
float:none;
font-size: 14px;
width: auto;}

footer .wrapper .social
{display:none;}}
```

At about 500px we will again adapt the layout of our website. The section “four_columns” will now display the articles below each other. We do not need a fixed width anymore so we can set it to auto. As we don´t want the articles inside our “four_columns” section to be next to each other anymore we will remove the floating and define a fixed width.

For our wrapper inside the footer and the rocket and clock articles we set the width to auto in order to make it adapt its width to the parent element. Unfortunately our navigation is too wide as well so we need to think of a solution. We could of course divide it into two parts which we position below each other but I find it better to completely remove the navigation and replace it with a simple select box. To achieve this we add the HTML code for a select box into the header of our website and give it an ID “alternative_menu”. We then set display to none to hide it by default and position it absolutely somewhere below the subheading paragraph. In our media query we will then set display to block to make it appear when the navigation gets hidden.

If you look at the website again and reduce your browser´s width you will see that the website will now respond to the changing width and look good at any screen size. Obviously you can always do more to make the website look even better on different screen sizes. Just play around a bit with the values and add rules for any element you want to change at a specific screen size.

Creating the JavaScript – Coding the slider content

Now that we have finished our styling we will create the JavaScript code for our slider content boxes to appear and disappear when the slider image changes. For the Slider to work we will add the JavaScript code just like in the example file of the SlidesJS plugin and make a few adjustments. For some reason we need to define a height in the JavaScript function which is much smaller than the actual slider images. I am not yet sure why this is but simply choose a height which makes the slider look good, in our case at about 235px.

In order for our slider content boxes to appear and disappear whenever a slider images is changing we need to add two callback functions. The first callback will fire when a slide starts to change to another, the second one whenever a slide has entered and is not moving anymore. In the start callback we will add a rule to fade out all the slider content boxes because as a slide changes we don´t want any box to be visible. In the complete callback we will fade in the slider content box corresponding to the current image. So say the second image is shown then “slider_content2″ will fade in. To achieve this we can add a variable into our complete function which will always give us the current slide number.

```
$(function() {$('#slides').slidesjs({height: 235,
navigation: false,
pagination: false,
effect: { fade: {speed: 400}},callback:
{start: function(number)
{$("#slider_content1,#slider_content2,#slider_content3").fadeOut(500);},
complete: function(number)
{$("#slider_content" + number).delay(500).fadeIn(1000);}},
play:
{active: false,
auto: true,
interval: 6000,
pauseOnHover: false}});});
```

Creating the JavaScript – Coding the alternative menu

As mentioned before we will now add the jQuery code for our alternative menu, The alternative menu appears as soon as the browser size falls below 500px. When we select an item from the select box we want to be redirected to the selected page. To do so we will simply add the following code into the $(document).ready part in our script.js file.

```
$("#alternative_menu").change(function()
{$selected = $("#alternative_menu option:selected");
window.location.href = "http://www.lingulo.com/" + $selected.val();});
```

With the first line we define a change handler for our alternative menu. Inside the change handler we set a variable "$selected", which holds the selected element from our alternative menu. In the last step we set the window.location.href to your domain and add the value attribute behind the domain. Now let´s add the page we want to be redirected to into the value attribute in our HTML code. To redirect for example to the page http://www.lingulo.com/contact you would add this to your alternative menu:

```
<option value="contact" >Contact</option>
```

Our alternative menu should work now. However using only a select box as a mobile menu is not the best way. I will now show you how to include a better navigation for small display sizes.

– Coming very soon –

Additional features – Creating skiplinks

If you have lots of content on your web page then it might be useful for your visitors to offer them a skiplink. With a skiplink they can get back to the top of the page by simply clicking a button. If you scroll down our website in the preview you will see that a little button in the right bottom corner will appear. Clicking that button will scroll the page up to the top. To create this skiplink we will download the source files from this quick tip. From the "index.html" file we copy the jQuery code which basically fades in the button if we have moved more than 200px from the top and removes it again if we have moved less than 200px from the top. In the click function we define that the page will scroll up to the top within 300ms when the button has clicked.

We also need to insert our button into the HTML code. To do so we simply add an anchor in the footer of our website and add a class "go-top". You can now style the button as you like or simply copy the styling from the CSS file in the source files which we just downloaded.

Improving our site´s performance

Now that our website has been finished let´s take a look at how we can improve the page speed ecspecially on mobile devices. The first easy thing we can do to speed up the loading time is to specify width and height attributes for all our images. When the browser loads a page it will try to start displaying content before all the images have been downloaded. The browser will leave gaps for each image and fill these gaps when the images have been loaded. However if you don´t define the size of an image the browser won´t know how much space to leave for it. In the end it will have to reposition all the elements on the page in order to make the image fit. So let´s add width and height attributes to all our images except the three slider images (because their dimensions will be set by SlidesJS). Note that we don´t have to add "px" to the width and height attributes.

```
<img src="img/Buzz-Private-icon.png" width="64" height="64" alt="alt
text about the private icon"/>
```

Except from setting dimensions for our images we also have

to make sure that the images have an appropriate file size.
Another thing we can do is to use CSS Sprites. When you have a lot of different images and icons on your website you need to make sure the amount of HTTP requests is as low as possible. To achieve this you can can combine all images into one single image and position the image in CSS depending on what icon you would like to be seen. As it is quite difficult to create your own sprites there are some useful tools out there like SpriteMe which automatically detects images on your website which can be combined to a sprite. To find out more about CSS Sprites be sure to read the SpriteMe FAQ.

Another important way to achieve better website performance is to cache your pages. Each time a user visits your website the browser has to download all web files in order to display the page correctly. This occurs to any element on the page be it an image or a CSS file. As we would like to reduce the amount of HTTP requests we need to find a way to define which files should be downloaded each time the user visits your website and which files can be stored in the browser´s cache as they are not regularly updated. One way to achieve this is by adapting your .htaccess file. Before adding the following code to your file be sure your server supports the module mod_expires (most of the web servers do).

```
<IfModule mod_expires.c>
ExpiresActive On
ExpiresDefault "access plus 1 month"
ExpiresByType text/html "access plus 1 month"
ExpiresByType image/jpeg "access plus 1 year"
ExpiresByType image/gif "access plus 1 year"
ExpiresByType image/png "access plus 1 year"
ExpiresByType text/css "access plus 1 month"
ExpiresByType text/javascript "access plus 1 month 1 week"
ExpiresByType application/x-javascript "access plus 1 month"
ExpiresByType text/xml "access plus 1 seconds"
</IfModule>
```

Each line defines the duration of your browser´s cache for a certain file type. The line ExpiresDefault "access plus 1 month" defines the default caching time for all files which are not set in the .htaccess file. In this example we set a caching time of one month to our HTML code and a caching time of one year for your images. You might make changes to your HTML files every now and then so it makes sense to refresh the browser´s cache every month. Your images however are in general not changed for a very long time so we set the caching time to one year for all images.

If you are using WordPress or another CMS you should make use of Cache plugins like W3 Total Cache which will give you a lot of ways to cache your website.

The next way to reduce our website´s loading time is to combine and minify our CSS and JS files. You could either do this manually by simply adding your CSS into one file and using an online tool like CSS Minifier as well as adding your JS files into one file and compressing them with an online tool like JS Compress. Or you could use Minify which automatically combines and minifies all your JS and CSS files. If you use the WordPress plugin W3 Total Cache it will include Minify by default.

Apart from minifying JS files we can also try to load them asynchronous. What that means is that the script will be loaded after the website has been rendered. That way the page appears faster leaving the asynchronous scripts to be loaded afterwards. Obviously there are scripts which need to be loaded from the very beginning, for example the Modernizr script or jQuery. In our example we could definitely set the lightbox script to load asynchronous. To do so we add the attribute async to the scripts we want to load after the document is ready. Here one example with the lightbox script.

```
<script async src="lightbox/js/lightbox.js"></script>
```

The last and probably most important thing in order to improve page speed on mobile devices is to load different image sizes depending on the device width. Right now we have three huge slider images which will be loaded even on mobile phones (even though they aren´t even visible!). One way to load images depending on the screen size is the new picture element.

```
<picture width="300" height="300">
<source media="(min-width: 35em)" src="large.jpg">
<source media="(min-width: 16em)" src="med.jpg">
<source src="small.jpg">
<img src="small.jpg" alt="">
<p>Accessible text</p>
</picture>
```

However as of now the picture element is not supported in major browsers but there is a fix called picturefill.js that mimics the picture element using span elements. If you would like to find out more about using images on responsive websites check out Chris Coyier´s blog post.

Responsive Menu Tutorial

Responsive design has basically become a staple to any modern website. With this in mind, navigation must be ready to adjust to the various screen sizes possible. As you'd imagine, a 10 item long horizontal menu doesn't translate well to a cellphone' screen. However, with a bit of proper tweaking, we can make a proper responsive menu for use on mobile.

Considerations

Throwing in a dash of media queries isn't all that's required to make a menu responsive. You have to know what type of menu you are working with, how you want it to display, and how it behaves. Use the following check list to help guide yourself:

Count

If your menu has a fixed amount of 3 or 4 items, you can most likely avoid needing any "special" mobile adjustments. This depends on the text though, as 4 long words would most likely cause an over flow problem.

Orientation

For the most part, if you have a vertical menu, you're already set. Vertical menus are the standard format for most mobile devices, as the portrait mode lends itself to that style. In the case of a long, horizontal menu, what you'll end up doing is styling it so that at lower resolutions it basically converts into a vertical menu instead.

Tap/Hover

An important thing to keep in mind is that mobile devices don't really have a proper "hover" state. This is a desktop only concept, which needs a different approach for mobile devices. This is only a concern if your menu has submenus, as that is generally when menu's make use of the hover states to continue expanding the navigation. When thinking responsive, we need to make sure hover states can be easily convert into

"taps". This means that top-level menu items can not be links *and* contain a submenu at the same time (unless you give them some sort of specific dropdown button to go along with them. This also means you'll want each menu item to be around at least 26-40 pixels tall, to be easy tap targets for a finger.

Wrapping

Usually, a long horizontal menu will want to remain as horizontal as possible until necessary. However, once you get into tablet portrait resolutions, that may become difficult. You'll have to decide whether you're okay with the menu wrapping its items, or alternatively, you can add a class to specific menu items that you don't mind simply hiding for mobile users.

Example Menu

For this tutorial, we'll keep things simple. We'll create a responsive menu, with a variable menu item count, and a 2nd level sub menu for some items:

- Home
- Products
- About
- Contact

For this menu, we'll stick to one responsive state (phone), which would make it look like this:

- Home
- Products
- About
- Contact

Menu Markup

We'll start off with some simple markup, using 4 items, with one of them containing a submenu...

```
<ul class="egmenu">
  <li><a href="#">Home</a></li>
  <li>
    <a href="#" class="has-sub">Products</a>
    <ul>
      <li><a href="#">Computers</a></li>
      <li><a href="#">Phones</a></li>
      <li><a href="#">Tablets</a></li>
      <li><a href="#">Beepers</a></li>
    </ul>
  </li>
  <li><a href="#">About</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

Styling with Responsive States

Once you have the markup setup, we can go ahead and add some basic styles, as well as add the submenu toggling:

```
ul.egmenu {
    background: #333;
    height: 30px;
    width: 100%;
}

ul.egmenu > li {
    float: left;
    position: relative;
}

ul.egmenu ul {
    background: #444;
    display: none;
    position: absolute;
    left: 0; top: 100%;
}

ul.egmenu a {
    cursor: pointer;
    display: block;
    color: white;
    line-height: 30px;
    padding: 0 10px;
}

ul.egmenu li { list-style: none; }

ul.egmenu li:hover { background: #555;  }
ul.egmenu li:hover ul { display: block;  }
```

Now here's where the magic happens. For this example we want the menu to go into mobile mode if the browser window squeezes under the 480px mark. To do this, we add a media query, which then alters some of the previously defined selectors with some new properties!

```
@media all and (max-width: 480px) {
    ul.egmenu {  height: auto; }
    ul.egmenu > li {  float: none; width: 100%; }
    ul.egmenu a { line-height: 40px; }
    ul.egmenu ul { position: relative; }
}
```

With that block, you can probably tell, we want at 480 pixels (or less) of screen width to change a couple properties such as: making the menu height auto (instead of fixed for horizontal), take out the floats on the list items so they become vertical, make items a little taller, and put the submenu inside (instead of floating on top).

Final Touches with Javascript

Before dropping some touch-device love through Jquery, let's alter some behavior for simplicity's sake. For mobile size, we don't want to depend on the :hover state, and instead toggle it with a tap. However, notice that right now we have the last 2 :hover rule lines in our CSS, which **will** be applied to mobile, messing with the behavior we want. We can actually wrap this inside another media query except in reverse to the other we have!

```
@media all and (min-width: 481px) {
    ul.egmenu li:hover { background: #555;  }
    ul.egmenu li:hover ul { display: block;  }
}
```

Now we have a media query for everything over 480 pixels. Again, we're keeping things **very** simple here and are completely ignoring tablets (which can sometimes have desktop-like resolutions, but still work better with taps), but this will do for our example. At this point, if you were to try the menu on a mobile device, you would not be able to open the submenu (it would just go the whatever link you attached to the "Products" href). This is where we use some jQuery to add our toggle. Reminder: as per our considerations listed above, by using this technique we'll have to effectively kill the "Product" link. This can easily be replaced by adding an extra menu item like "All Products" or similar in the submenu itself, containing the same link, but we'll leave that one for you to figure out!

```
$('.has-sub').click( function(e) {
    e.preventDefault();
    $(this).parent().toggleClass('tap');
});
```

Notice how we hooked the click event (which basically translate to a tap on a touch device) to the link, rather than the parent li like we did when using :hover in CSS. This is so that we can call the .preventDefault function and avoid the link jumping to another page. However, after doing that, we do add the "tap" class to the parent li, which we now combine with this CSS:

```
@media all and (max-width: 480px) {
    ul.egmenu {  height: auto; }
    ul.egmenu > li {  float: none; width: 100%; }
    ul.egmenu a { line-height: 40px; }
    ul.egmenu ul { position: relative; }

    ul.egmenu li.tap { background: #555;  }
    ul.egmenu li.tap ul { display: block;  }
}
```

Notice how we added 2 lines to our mobile media query, very similar to the :hover's, but this time with the "tap" class. This will give us the same style, with our new behavior, and now with our JS will give us a touch-driven mobile menu!

Responsive web design

Responsive web design term is related to the concept of developing a website design in a manner that helps the lay out to get changed according to the user's computer screen resolution. More precisely, the concept allows for an advanced 4 column layout 1292 pixels wide, on a 1025 pixel width screen, that auto-simplifies into 2 columns. Also, it suitably fixes on the smartphone and computer tablet screen. This particular designing technique we call *"responsive design"*.

Almost every new client these days wants a mobile version of their website. It's practically essential after all: one design for the BlackBerry, another for the iPhone, the iPad, netbook, Kindle — and all screen resolutions must be compatible, too. In the next five years, we'll likely need to design for a number of additional inventions.

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. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities. 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.

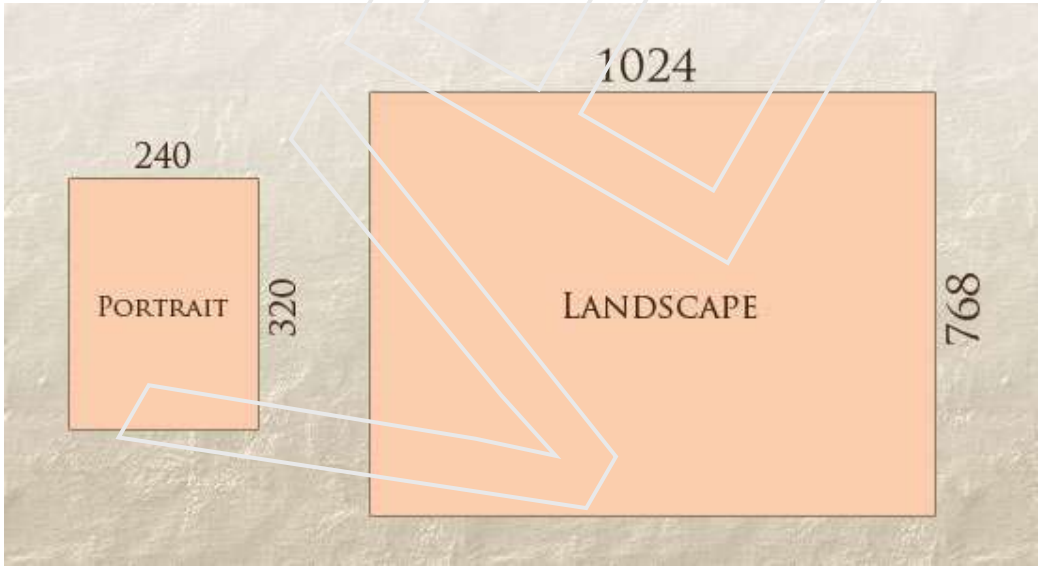
The Concept Of Responsive Web Design

Ethan Marcotte wrote an introductory article about the approach, "Responsive Web Design," for A List Apart. It stems from the notion of responsive architectural design, whereby a room or space automatically adjusts to the number and flow of people within it:

But responsive Web design is **not only about adjustable screen resolutions and automatically resizable images**, but rather about a whole new way of thinking about design. Let's talk about all of these features, plus additional ideas in the making.

Adjusting Screen Resolution

With more devices come varying screen resolutions, definitions and orientations. New devices with new screen sizes are being developed every day, and each of these devices may be able to handle variations in size, functionality and even color. Some are in landscape, others in portrait, still others even completely square. As we know from the rising popularity of the iPhone, iPad and advanced smartphones, many new devices are able to switch from portrait to landscape at the user's whim.



In addition to designing for both landscape and portrait (and enabling those orientations to possibly switch in an instant upon page load), we must consider the hundreds of different screen sizes. Yes, it is possible to group them 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, many users do not maximize their browsers, which itself leaves far too much room for variety among screen sizes.

Part of the Solution: Flexible Everything

Now we can make things more flexible. Images can be automatically adjusted, and we have workarounds so that layouts never break (although they may become squished and illegible in the process). While it's not a complete fix, the solution gives us far more options. It's perfect for devices that switch from portrait orientation to landscape in an instant or for when users switch from a large computer screen to an iPad.

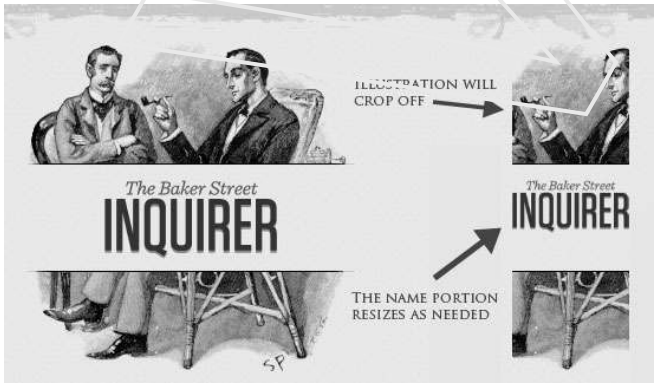
In Ethan Marcotte's article, he created a sample Web design that features this better flexible layout:

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:

- Hiding and Revealing Portions of Images
- Creating Sliding Composite Images
- Foreground Images That Scale With the Layout

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." In addition, Zoe provides the following extensive list of tutorials, resources, inspiration and best practices on creating flexible grids and layouts: "Essential Resources for Creating Liquid and Elastic Layouts."

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).

This is just one example of the kind of thinking that makes responsive Web design truly effective. But even with smart fixes like this, a layout can become too narrow or short to look right. In the logo example above (although it works), the ideal situation would be to not crop half of the illustration or to keep the logo from being so small that it becomes illegible and "floats" up.

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, noted in Ethan Marcotte's article on fluid images but first experimented with by Richard Rutter, is to use CSS's `max-width` for an easy fix.

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

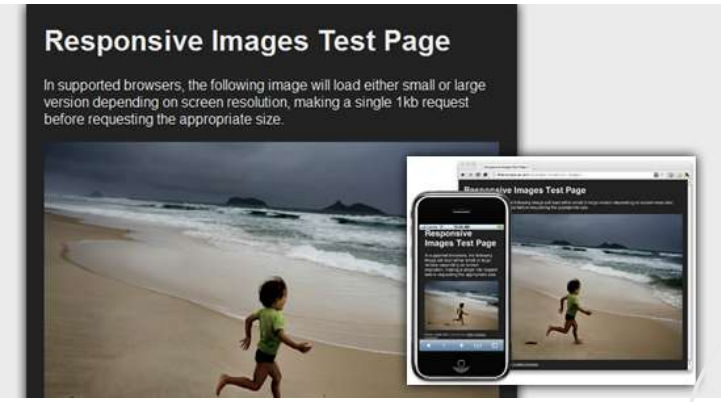
As long as no other width-based image styles override this rule, every image will load in its original size, unless the viewing area becomes narrower than the image's original width. The **maximum width** of the image is set to 100% of the screen or browser width, so when that 100% becomes narrower, so does the image. Essentially, as Jason Grigsby noted, "The idea behind fluid images is that you deliver images at the maximum size they will be used at. You don't declare the height and width in your code, but instead let the browser resize the images as needed while using CSS to guide their relative size". It's a great and simple technique to resize images beautifully.

Note that `max-width` is **not supported in IE**, but a good use of `width: 100%` would solve the problem neatly in an IE-specific style sheet. One more issue is that when an image is resized too small in some older browsers in Windows, the rendering isn't as clear as it ought to be. There is a JavaScript to fix this issue, though, found in Ethan Marcotte's article.

While the above is a great quick fix and good start to responsive images, image resolution and download times should be the primary considerations. While resizing an image for mobile devices can be very simple, if the original image size is meant for large devices, it could significantly slow download times and take up space unnecessarily.

Filament Group's 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.

The JavaScript file inserts a base element that allows the page to separate responsive images from others and redirects them as necessary. When the page loads, all files are rewritten to their original forms, and only the large or small images are loaded as necessary. With other techniques, all higher-resolution images would have had to be downloaded, even if the larger versions would never be used. Particularly for websites with a lot of images, this technique can be a great saver of bandwidth and loading time.

This technique is fully supported in modern browsers, such as **IE8+, Safari, Chrome and Opera, as well as mobile devices that use these same browsers** (iPad, iPhone, etc.). Older browsers and Firefox degrade nicely and still resize as one would expect of a responsive image, except that both resolutions are downloaded together, so the end benefit of saving space with this technique is void.

Stop iPhone Simulator Image Resizing

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. Then, the user could easily zoom in and out as necessary.

There was, however, one issue this simulator created. When responsive Web design took off, many noticed that images were still changing proportionally with the page even if they were specifically made for (or could otherwise fit) the tiny screen. This in turn scaled down text and other elements.

Because this works only with Apple's simulator, we can use an Apple-specific meta tag to fix the problem, placing it *below* the website's `<head>` section. Thanks to Think Vitamin's article on image resizing, we have the meta tag below:

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

Setting the `initial-scale` to `1` overrides the default to resize images proportionally, while leaving them as is if their width is the same as the device's width (in either portrait or landscape mode). Apple's documentation has a lot more information on the viewport meta tag.

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 inherit these styles and move elements around with floats, widths, heights and so on.

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. Default flexible widths and floats could also be defined.

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. This new child style sheet would adopt everything from the default style sheet and then just redefine the layout's structure.

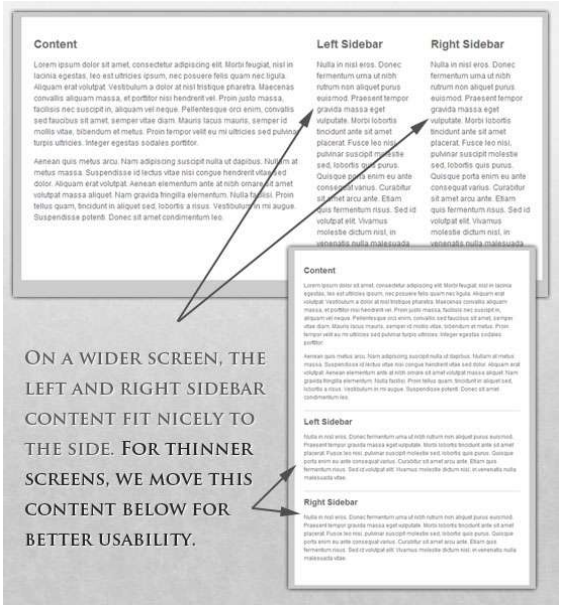
Here is the ***style.css*** **(default) content:**

```
/* 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;
}
```

Here is the ***mobile.css*** **(child)** content:

```
#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;
}
```



Media Queries

CSS3 supports all of the same media types as CSS 2.1, such as `screen`, `print` and `handheld`, but has added dozens of new media features, including `max-width`, `device-width`, `orientation` and `color`. New devices made after the release of CSS3 (such as the iPad and Android devices) will definitely support media features. So, calling a media query using CSS3 features to target these devices would work just fine, and it will be ignored if accessed by an older computer browser that does not support CSS3.

In Ethan Marcotte's article, we see an example of a media query in action:

```
<link rel="stylesheet" type="text/css"
        media="screen and (max-device-width: 480px)"
        href="shetland.css" />
```

This media query is fairly self-explanatory: if the browser displays this page on a screen (rather than print, etc.), and if the width of the screen (not necessarily the viewport) is 480 pixels or less, then load *shetland.css*.

New CSS3 features also include `orientation` (portrait vs. landscape), `device-width`, `min-device-width` and more. Look at "The Orientation Media Query" for more information on setting and restricting widths based on these media query features.

One can create multiple style sheets, as well as basic layout alterations defined to fit ranges of widths — even for landscape vs. portrait orientations. Be sure to look at the section of Ethan Marcotte's article entitled "Meet the media query" for more examples and a more thorough explanation.

Multiple media queries can also be dropped right into a single style sheet, which is the most efficient option when used:

```
/* Smartphones (portrait and landscape) ----------- */
@media only screen
and (min-device-width : 320px)
and (max-device-width : 480px) {
/* Styles */
}
```

```
/* Smartphones (landscape) ----------- */
@media only screen
and (min-width : 321px) {
/* Styles */
}

/* Smartphones (portrait) ----------- */
@media only screen
and (max-width : 320px) {
/* Styles */
}
```

The code above is from a free template for multiple media queries between popular devices by Andy Clark. See the differences between this approach and including different style sheet files in the mark-up as shown in the post "Hardboiled CSS3 Media Queries."

CSS3 Media Queries

Above are a few examples of how media queries, both from CSS 2.1 and CSS3 could work. Let's now look at some specific how-to's for using CSS3 media queries to create responsive Web designs. Many of these uses are relevant today, and all will definitely be usable in the near future.

The **min-width and max-width** properties do exactly what they suggest. 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. Anything above the maximum browser or screen width specified would not apply to the respective media query.

Note in the examples below that we're using the syntax for media queries that could be used all in one style sheet. As mentioned above, the most efficient way to use media queries is to place them all in one CSS style sheet, with the rest of the styles for the website. This way, multiple requests don't have to be made for multiple style sheets.

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

The class specified in the media query above (`hereIsMyClass`) will work only if the browser or screen width is above 600 pixels. In other words, this media query will run only if the **minimum width is 600 pixels** (therefore, 600 pixels or wider).

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

Now, with the use of `max-width`, this media query will apply only to browser or screen widths with a maximum width of 600 pixels or narrower.

While the above `min-width` and `max-width` can apply to either screen size or browser width, sometimes we'd like a media query that is relevant to device width specifically. This means that even if a browser or other viewing area is minimized to something smaller, the media query would still apply to the size of the actual device. The **min-device-width and max-device-width** media query properties are great for targeting certain devices with set dimensions, without applying the same styles to other screen sizes in a browser that mimics the device's size.

```
@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;
     }
}
```

There are also other tricks with media queries to target specific devices. Thomas Maier has written two short snippets and explanations for targeting the iPhone and iPad only:

- CSS for iPhone 4 (Retina display)
- How To: CSS for the iPad

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;
     }
}
```

Unfortunately, this property works only on the iPad. When determining the orientation for the iPhone and other devices, the use of `max-device-width` and `min-device-width` should do the trick.

There are also many media queries that **make sense when combined**. For example, the `min-width` and `max-width` media queries are combined all the time to set a style specific to a certain range.

```
@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. A good use of this technique is to show certain content or entire sidebars in a layout depending on how much horizontal space is available.

Some designers would also prefer to **link to a separate style sheet** for certain media queries, which is perfectly fine if the organizational benefits outweigh the efficiency lost. For devices that do not switch orientation or for screens whose browser width cannot be changed manually, using a separate style sheet should be fine.

You might want, for example, to place media queries all in one style sheet (as above) for devices like the iPad. Because such a device can switch from portrait to landscape in an instant, if these two media queries were placed in separate style sheets, the website would have to call each style sheet file every time the user switched orientations. Placing a media query for both the horizontal and vertical orientations of the iPad in the same style sheet file would be far more efficient.

Another example is a flexible design meant for a standard computer screen with a resizable browser. If the browser can be manually resized, placing all variable media queries in one style sheet would be best.

Nevertheless, organization can be key, and a designer may wish to define media queries in a standard HTML link tag:

```
<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

Another method that can be used is JavaScript, especially as a back-up to devices that don't support all of the CSS3 media query options. Fortunately, there is already a pre-made JavaScript library that makes older browsers (IE 5+, Firefox 1+, Safari 2) support CSS3 media queries. If you're already using these queries, just grab a copy of the library, and include it in the mark-up: *css3-mediaqueries.js*.

In addition, below is a sample jQuery snippet that detects browser width and changes the style sheet accordingly — if one prefers a more hands-on approach:

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

There are many solutions for pairing up JavaScript with CSS media queries. Remember that media queries are not an absolute answer, but rather are fantastic options for responsive Web design when it comes to pure CSS-based solutions. With the addition of JavaScript, we can accomodate far more variations. For detailed information on using JavaScript to mimic or work with media queries, look at "Combining Media Queries and JavaScript."

Showing or Hiding Content

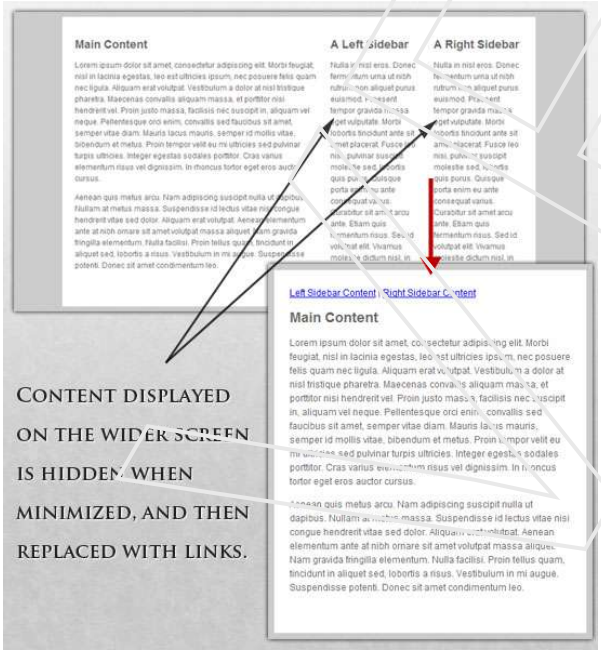
It is possible to shrink things proportionally and rearrange elements as necessary to make everything fit (reasonably well) as a screen gets smaller. It's great that that's possible, but making every piece of content from a large screen available on a smaller screen or mobile device isn't always the best answer. We have best practices for mobile environments: simpler navigation, more focused content, lists or rows instead of multiple columns.

Responsive Web design shouldn't be just about how to create a flexible layout on a wide range of platforms and screen sizes. It should also be about the user being able to pick and choose content. Fortunately, CSS has been allowing us to show and hide content with ease for years!

```
display: none;
```

Either declare `display: none` for the HTML block element that needs to be hidden in a specific style sheet or detect the browser width and do it through JavaScript. In addition to hiding content on smaller screens, we can also hide content in our default style sheet (for bigger screens) that should be available only in mobile versions or on smaller devices. For example, as we hide major pieces of content, we could replace them with navigation to that content, or with a different navigation structure altogether.

Note that we haven't used `visibility: hidden` here; this just hides the content (although it is still there), whereas the `display` property gets rid of it altogether. For smaller devices, there is no need to keep the mark-up on the page — it just takes up resources and might even cause unnecessary scrolling or break the layout.



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.

Here is the ***style.css*** **(default)** content:

```
#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 two sidebars (below) and show the links to these pieces of content. As an alternative, the links could call to JavaScript to just cancel out the `display: none` when clicked, and the sidebars could be realigned in the CSS to float below the content (or in another reasonable way).

Here is the ***mobile.css*** **(simpler)** content:

```
#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 example, use a script or alternate style sheet to increase white space or to replace image navigation sources on mobile devices for better usability (icons would be more beneficial on smaller screens).

Touchscreens vs. Cursors

Touchscreens are becoming increasingly popular. Assuming that smaller devices are more likely to be given touchscreen functionality is easy, but don't be so quick. Right now touchscreens are mainly on smaller devices, but many laptops and desktops on the market also have touchscreen capability. For example, the HP Touchsmart tm2t is a basic touchscreen laptop with traditional keyboard and mouse that can transform into a tablet.

Touchscreens obviously come with different design guidelines than purely cursor-based interaction, and the two have different capabilities as well. Fortunately, making a design work for both doesn't take a lot of effort. Touchscreens have no capability to display CSS hovers because there is no cursor; once the user touches the screen, they click. So, don't rely on CSS hovers for link definition; they should be considered an additional feature only for cursor-based devices.

Look at the article "Designing for Touchscreen" for more ideas. Many of the design suggestions in it are best for touchscreens, but they would not necessarily impair cursor-based usability either. For example, sub-navigation on the right side of the page would be more user-friendly for touchscreen users, because most people are right-handed; they would therefore not bump or brush the navigation accidentally when holding the device in their left hand. This would make no difference to cursor users, so we might as well follow the touchscreen design guideline in this instance. Many more guidelines of this kind can be drawn from touchscreen-based usability.