

Responsive Design Essentials

Responsive Design cover	1
Responsive Web & Mobile.	1
Goal of Responsive Design	
Most Popular Mobile & Tablet Resolutions 2015	2
The Universal Page	
Responsive Design Starts with HTML & CSS	
Technologies Necessary for Responsive Design	
Responsive Considerations & Guidelines	4
Do	
Don't	
Your Responsive Canvas The Browser Window	
Setting Up Constraints	
A Dao of Web Design	6
Fixed 960 Pixel Grid System (before Responsive)	7
12-Column Grid	
16-Column Grid	
12-Column Layout.	7
Content Could Span Several Columns	
16-Column Layout	7
Content Could Span Several Columns	
Planning Responsive Wireframes	8
Responsive Sketch Sheets	
Electronic Wireframe & Planning Tools	
HTML5.	9
The Need for New HTML Elements	
WHATWG New Standards Body.	10
Workshop on Web Applications and Compound Documents	
HTML5 a Living Standard	
See What CSS3 & HTML5 Can Do	11
Cascading Style Sheets (CSS).	14
CSS Box Model	
Three Methods for Applying CSS.	15
Inline	
Embedded	
External	
CSS Vocabulary	17

Types of CSS Selectors	18
Element	
ID	
Class	
3-Digit Hexadecimal Values	19
CSS3	20
Rounded CSS Corners	
Rounded Corners for Multiple Browsers	21
CSS Border-radius Prefix	
Rounded Corners for IE9	
Media Query Intro	22
Flexible or Fluid Grid = The Formula	23
Flexible Margins or Flexible Padding	
CSS Box Model Review	
Reset CSS	24
The Viewport	25
Layout Viewport	26
Mobile Viewport or Visual Viewport	26
Viewport Control	27
Viewport Meta Tag	
Recommended Viewport Meta Tag	
Viewport Width & Height Attributes	
iPhone Widths	
Viewport CSS	
Viewport Width (vw) in CSS3	
Is a Pixel a Pixel?	30
Hardware Pixels (or Device Pixels)	
Reference Pixels (or CSS Pixels)	
Screen Density	
Device-Pixel-Ratio	
Window Size	
Break Points.	32
Layouts for Each Breakpoint.	32
Framework for Specific Needs	32
Custom Framework Concerns	
Relative Type Sizes.	33

Proportional Type Sizes	
Web Typography	
Flexible Images34
3 Factors in Determining Responsive Images	
Flexible Image Concerns by Device	
Raster Images on HiDPI or Retina Screens	
Responsive Background Images	
Responsive Image Options37
Browser Prefetching or Link Prefetching40
The Future of Responsive Images or <picture>	
Support for SRCSET	
Support for <picture>	
Picturefill Responsive Image Polyfill	
Responsive Forms43
Tips for Forms	
Required Reading44
Resource Web Sites44
CSS Resources44
HTML5 Resources45
HTML5 Forms	
Browser Testing of HTML45
Responsive Website Examples46
Responsive Design Resources46
Responsive Testing.47
Responsive Images.47

Responsive Web & Mobile

Responsive Design is a design method initially identified by **Ethan Marcotte** in a book titled *“Responsive Web Design.”* Responsive Design is all about flexibility. Most people have heard that websites today should be responsive, but very few have a handle on what that exactly means. In this class you will get the full picture of how responsive pages & sites are created and what you need in order to build a Responsive layout.

Goal of Responsive Design

The goal of Responsive Design is to craft websites so they provide an optimal viewing experience across a wide range of devices from desktop computer to mobile and devices. If a site is built as responsive, it offers easy reading & navigation with a minimum amount of resizing, panning, zooming and scrolling on the device that is viewing the content.

“Smartphone usage is up 394 percent, and tablet usage is up a whopping 1,721 percent as these platforms now combine to account for 60 percent of digital media time spent.” (from 2010 – 2014)

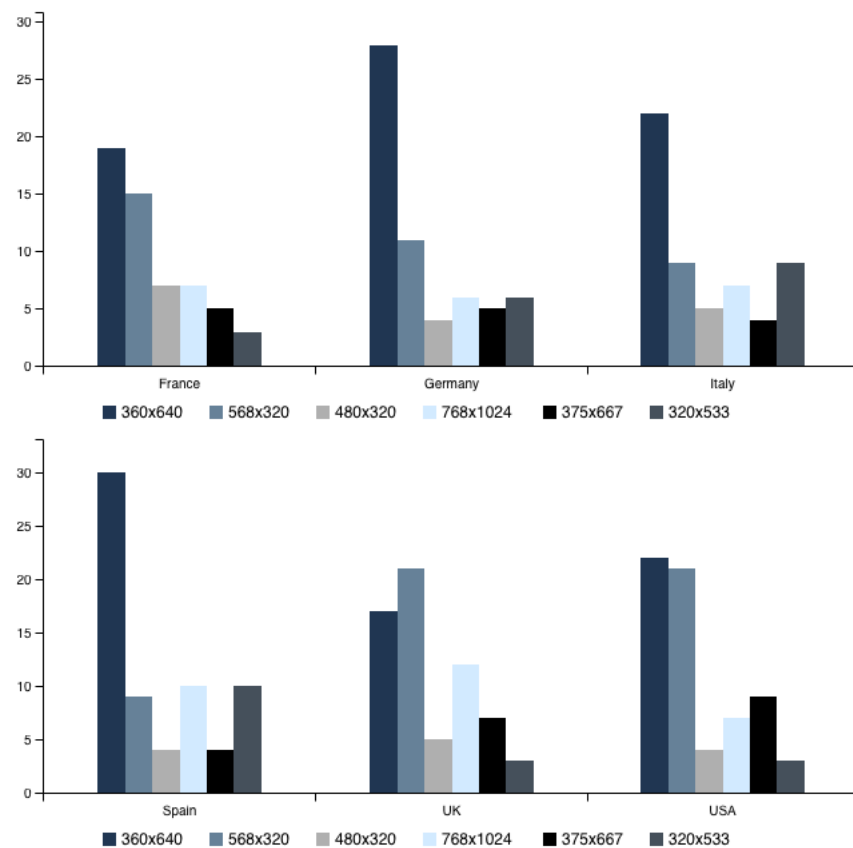
Source: comscore.com/Insights/Blog/Mobile-Internet-Usage-Skyrockets-in-Past-4-Years-to-Overtake-Desktop-as-Most-Used-Digital-Platform

(April 13, 2015)

Most Popular Mobile & Tablet Resolutions 2015

**in virtual (CSS) pixels. Source: DeviceAtlas*

<https://deviceatlas.com/blog/most-popular-smartphone-screen-resolutions-2015>



The Universal Page

Web content today must be adaptable and accessible. A “**universal page**” would be accessible, work well and look fantastic regardless of browser, platform or screen that the reader chooses. Web pages must adapt to the needs of the viewer, not simply bend to the “*design vision*” of the author. Consider that someone looking at your page may be visually impaired and need larger fonts, they may be blind and have a screen reading application delivering the content, they may be on a busy train on their way to work in the morning and viewing important content on the small screen of their smartphone.

Responsive Design Starts with HTML & CSS

A responsive layout starts with clean, semantic HTML. This means that your HTML tags follow the guidelines of the W3C (*World Wide Web Consortium*), your IDs and Classes are properly identified and you aren't writing any non-standard code. HTML5 includes a few new features that help websites become more responsive, such as the new form elements (*which include over a dozen new input types*).

Technologies Necessary for Responsive Design

Much of the technologies necessary to build responsive sites existed when **Ethan Marcotte** coined the phrase Responsive Web Design:

- Fluid Grids or Flexible, Grid-based Layouts (*may involve CSS3 Transitions & Transforms*)
- Flexible Images and Media (*CSS-based Graphics*)
- Media Queries (*a module from the CSS3 specification that may be combined with JavaScript to control resource loading, adaptive media and control device-specific functions*)

Responsive Considerations & Guidelines

Think about what your page should do, not what it looks like. Let the design come from what the end user needs, not from the idea of what YOU want your page to look like. Cascading Style Sheets (CSS) should be used to “suggest” the appearance of the page. Older browsers that don’t support modern CSS will display plainer content, but the content should still show.

Do

- Let form follow function
- Separate content from it’s appearance
- If HTML offers an appropriate element, like `<p>` or `<h1>` use it, where it doesn’t create a **CSS class**
- Be sure every element of your page has an assigned style
- Use color thoughtfully and with intent
- Set your font sizes in ems, rems or percentages
- Design for both Landscape and Portrait orientations (*and tell users which works best, prompt them to switch orientation if it works better for the content being viewed*)
- Keep menus short (*consider how to present your menu with the fewest items possible*)
- Make it easy to get back to the Home page (*your company logo should always be a path back to the home page*)
- Keep your page in a single window (*opening multiple pages on a smartphone or tablet can be problematic, keep your content in 1 window*)

Don’t

- Get a design idea stuck in your head then force the programmers to make it “work”
- Use any HTML for presentation (no ``, `` or `<i>` tags)
- Assign **ABSOLUTE** units, like **pixels** or **points**
- Offer “full site” links, this implies that the mobile or responsive site is somehow limited (*instead use terms like “desktop” instead of “full”*)
- Use long, “desktop-like” navigation

Note: The default resolution for most Mac monitors (not including Retina) is 72 ppi. The Windows default resolution is 96 ppi, meaning 72 pixels = 1 inch on the Mac, but the scale will be different on Windows.

Keeping this in mind, 12 point type on a Mac will print at 12 point. 12 point type on Windows at 96 ppi, will actually print at 16 points...

Your Responsive Canvas | The Browser Window

The browser window, on any device or platform, is truly your Responsive Canvas. Using your favorite HTML authoring application or image editor (for mock-ups) we begin the design process, keeping our limited canvas in mind. Once your content is viewed online you are at the mercy of the end-user's device:

- Screen size
- Resolution
- Font settings
- Zoom level
- and more...

Setting Up Constraints

In order to control the user's experience, we setup constraints (or specify everything we can for the page):

- Default background color
- Font sizes
- Text color
- Header, Footer and Navigation areas
- Maximum of 960 pixel width for the window (*many designers today are using 1180 as the new max size*)

A Dao of Web Design

An article written by John Alsopp on April 7, 2000 is still a great (*and necessary*) read today:

<http://alistapart.com/article/dao>

This article gives insight into what was coming for the web. It lays the foundation of “*rituals*” which are repeated patterns borrowed from an old medium and applied to a new one: like Radio to Television and Print to the Web. We adopt habits and techniques that worked well for the old, not considering if they work well for the new. Linking the web to the human experience or “harmony” of design (*or to pull in the Dao, a way of living, of being for web developers*). “*Make pages which are adaptable.*”

Well established hierarchies are not easily uprooted;
Closely held beliefs are not easily released;
So ritual enthralls generation after generation.

— *Tao Te Ching; 38 Ritual*

“The control which designers know in the print medium, and often desire in the web medium, is simply a function of the limitation of the printed page. We should embrace the fact that the web doesn’t have the same constraints, and design for this flexibility. But first, we must “accept the ebb and flow of things.”

— *John Alsopp | A Dao of Web Design*

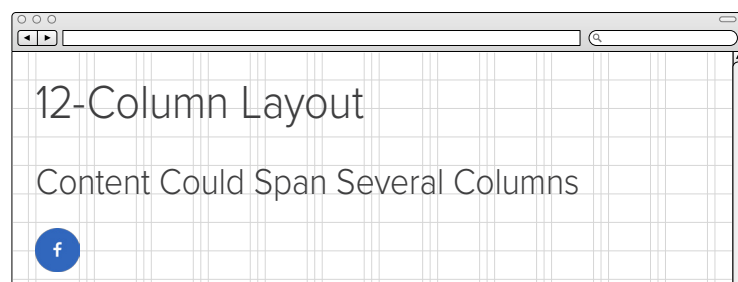
Fixed 960 Pixel Grid System (before Responsive)

The 960 Grid System is a standard dimension for web developers to work within. This grid system is used when designing pages for the desktop, based on a maximum width of 960 pixels. Keeping 960 px in mind, the page is often laid out in 12 or 16 columns, which can be used separately or together. For more detail see: <http://960.gs>

The goal of the 960 px grid system is for rapid prototyping and to avoid **left and right scroll bars** or **clipped content** on end user's browser windows. There are a lot of design layouts, and re-usable CSS files for the 960 pixel grid available online.

12-Column Grid

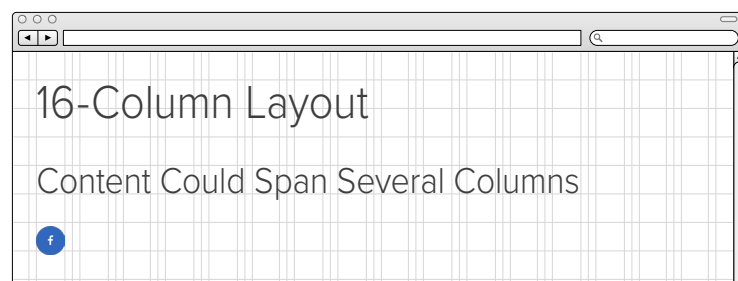
The 12-column grid is divided into portions that are 60 pixels wide.



Source: <http://github.com/nathansmith/960-Grid-System>

16-Column Grid

The 16-column grid is made up of portions that are 40 pixels wide. Each column has 10 pixels of margin on the left and right, which create 20 pixel wide gutters between columns.



Source: <http://github.com/nathansmith/960-Grid-System>

Planning Responsive | Wireframes

To start I recommend creating 3 or 4 versions of your mock-up, to fit small, medium, and large screen sizes (for smartphones, tablets, and desktop). Designing with code and testing in the browser is more difficult and time consuming. However, if you create flexible layouts at the thinner and wider ends of the design, you can easily resize the browser to figure out exactly where a layout breaks.

Responsive Sketch Sheets

Here are a few resources for wireframing.

Responsive Web Design Sketch Sheets (*PDF files with different blank layouts*)

<http://jeremypalford.com/arch-journal/responsive-web-design-sketch-sheets/>

Zurb Responsive Sketchsheets (*PDF files with different blank layouts*)

<http://zurb.com/playground/responsive-sketchsheets>

APP Sketchbook (*a physical notebook*)

<http://appsketchbook.com/products/responsive-design-sketchbook>

Electronic Wireframe & Planning Tools

This area is ever evolving, there are many tools to choose from here. I have listed a few:

- NEW Adobe Project Comet: <http://adobe.com/ProjectComet>
- Webflow: <https://webflow.com>
- Balsamiq: <https://balsamiq.com>
- Adobe Comp (App): <http://www.adobe.com/products/comp.html>
- Sketch (App): <https://www.sketchapp.com>

HTML5



HTML5 is the latest iteration of HTML, but it is so much more than that. HTML5 builds on the W3C's (*World Wide Web Consortium*) open web platform. HTML5 attempts to foster development with the full potential of the web.

HTML5 supports a richer (and new) set of tags, microdata and microformats. RDFa (Resource Description Framework in Attributes) technology is also part of HTML5. RDFa provides a set of markup attributes that has machine-readable hints. Here is a real world example: If I see the website zappos.com and I click the "Like" button on Facebook™ my news feed can change to display more catered information, based on the knowledge that I like Zappos products. This means that we have an evolving, data-driven web experience.

The Need for New HTML Elements

When HTML5 was being developed, they researched the most commonly used ID and Class names (in CSS). Some of the most popular results were:

- Header
- Footer
- Nav
- Section
- Article
- Figure
- Audio
- Video
- Embed
- Time
- Progress

For this reason these new HTML elements are built-into HTML5. Think of them as an addition to the standard `<h1>` and `<p>` tags.

WHATWG | New Standards Body

The **WHATWG** (*Web Hypertext Application Technology Working Group*) <http://whatwg.org> was established in 2004 as a community of developers interested in evolving the web. The WhatWG was started by Mozilla, Opera and Apple and became the driving force behind the development of HTML5. Their **main** focus is HTML, DOM (*Document Object Model*), and URLs including an API (*Application Program Interface*) for URLs. There are other goals of the WhatWG and increasingly implementers and interested WhatWG community members are communicating on **GitHub**. Making the process open to anyone, helping the community collaborate and contribute to the new web standards.

Workshop on Web Applications and Compound Documents

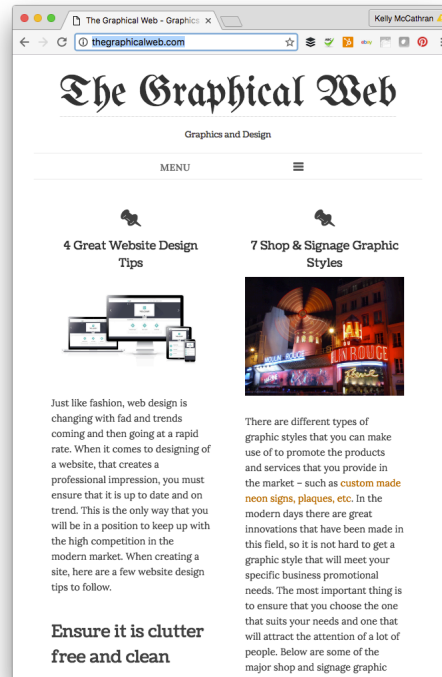
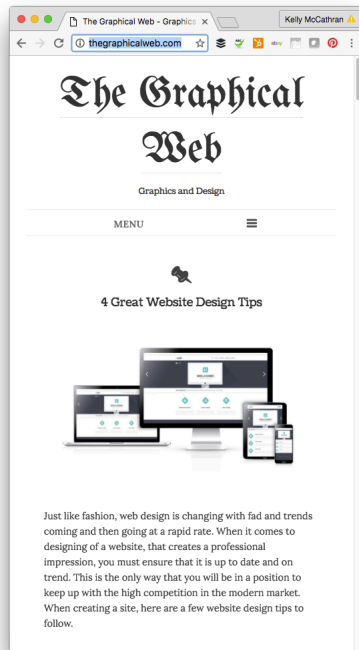
The impetus behind **WhatWG** started in **mid-2004** at a **W3C Workshop** at on Web Applications and Compound Documents held at **Adobe** in San Jose. The W3C had so far failed to create a language for web applications, XHTML became standard in 1999, but still didn't address this. Mozilla and Opera jointly proposed a vote to see if the W3C should extend HTML and the DOM to support web applications. The vote was defeated: **8 YES** and **14 NO**.

HTML5 a Living Standard

HTML is a living standard, the WhatWG has abandoned the HTML5 specification in favor of HTML. Just HTML, that evolves and grows as necessary. When we will stop referring to HTML by the version number is anyone's guess.

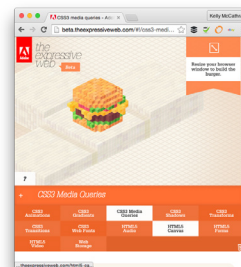
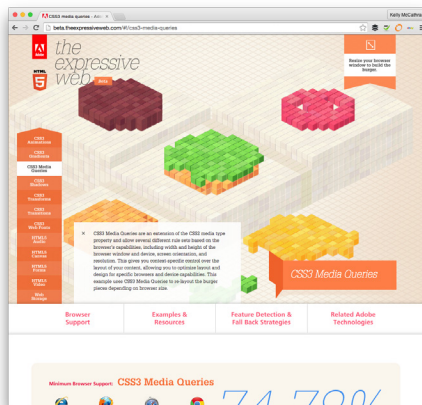
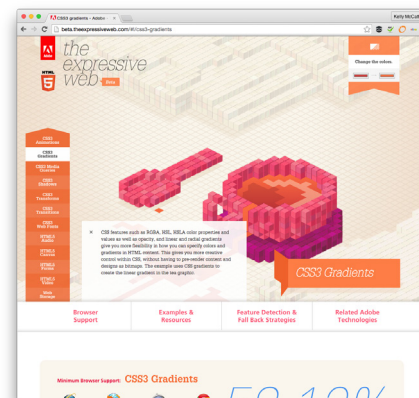
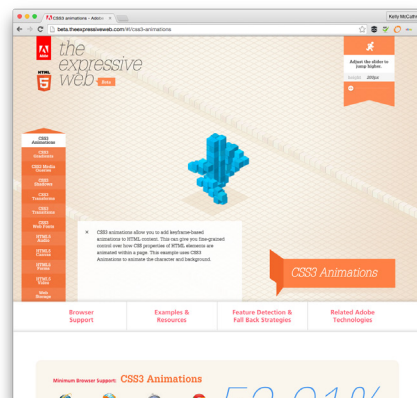
See What CSS3 & HTML5 Can Do

<http://thegraphicalweb.com>



<http://theexpressiveweb.com> (site no longer active)

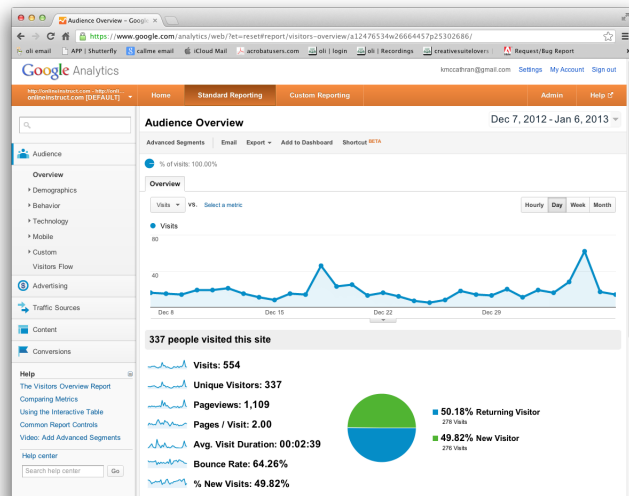
- CSS3 Animations
- CSS3 Gradients
- CSS3 Media Queries
- CSS3 Shadows
- CSS3 Transforms
- CSS3 Transitions
- CSS3 Web Fonts
- HTML5 Audio
- HTML5 Canvas
- HTML5 Forms
- HTML5 Video
- Web Storage



HTML5 Rocks | <http://html5rocks.com>



Google Analytics



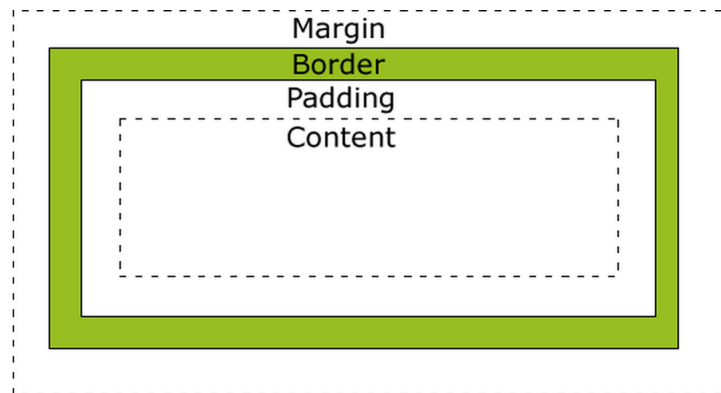
Cascading Style Sheets (CSS)

Cascading Style Sheets give us a better method than HTML for specifying how content should be presented on a web page. The main reason to use CSS is to separate **Structure** and **Presentation**. CSS can control many things, including: fonts, colors, background images and page layout. Another important reason to use CSS, is to have custom stylesheets for different devices. You could use one stylesheet for **Print**, one for **On-screen** and a third for **Mobile** devices. As you make a change to the CSS, when using External styles, those changes **Cascade** down to all the pages referencing that CSS document.

One important reminder, as with HTML, different browsers interpret Cascading Style Sheets differently. It is essential to test your pages on as many browsers and platforms as possible. Currently these are the most popular browsers, at a minimum you should keep 2 or 3 on your hard drive:

- Internet Explorer
- Firefox & Mozilla
- Netscape
- Safari
- Opera
- Mobile browsers (Safari for iPhone)

CSS Box Model



Three Methods for Applying CSS

There are 3 methods for applying Cascading Style Sheets to your web pages: Inline, Embedded or External.

Inline

Inline Cascading Style Sheets occur directly in the HTML and are supported by every tag. They are easy to understand, but not very practical since you'd have to keep repeating common style changes throughout the HTML.

Example:

```
<body>
  <h1 style="color: #669966; font:Verdana, Geneva, sans-serif;">Welcome to CSS</h1>
  <p style="font:Verdana, Geneva, sans-serif;">This is the first paragraph.</p>
  <p style="font:Verdana, Geneva, sans-serif; font-size: .7em; text-align:center">
    Copyright 2008 Kelly McCathran</p>
</body>
```

Embedded

Embedded Cascading Style Sheets specify all style information in the header of the HTML, using a style tag. This gives the page "rules".

Example:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>CSS Practice</title>
<style type="text/css">
  <!--
    h1 {
      font-family:Verdana, Geneva, sans-serif;
      color:#669966;
    }
  -->
</style>
</head>
```

Note: Typically for Embedded CSS, older browsers need HTML comments wrapping the CSS. This way if it can't interpret the styles, the code won't be displayed.

External

External Cascading Style Sheets link to a separate CSS document; thereby allowing web browsers to only have to download the style sheet once and use it multiple times.

Example:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>CSS Practice</title>
<link rel="stylesheet"
      type="text/css" href="CSS_practice.css"/>
</head>
```

CSS Document “CSS_practice.css”

```
@charset "UTF-8";
/* CSS Document */

h1 {
    font-family: Verdana, Geneva, sans-serif;
    color: #669966;
}
p {
    font-family: Verdana, Geneva, sans-serif;
    font-size: 1em;
}
```

Note: You can also have multiple external CSS docs referenced in the same HTML file.

CSS Vocabulary

Although Dreamweaver aids in the building of CSS styles, every good web designer should be able to “*talk the talk*”.

Opening CSS Closing
HTML Tag Property HTML Tag

`<h1 style="color: #669966;">Welcome to CSS</h1>`

HTML CSS Value
Attribute

CSS **Properties** are followed by colon and the value or values are listed after.

CSS **Values** are listed after properties. To list more than one value a comma is used. Semicolons are used to end a line (*called a declaration*). Also, the semicolon is equivalent to a return in UNIX.

CSS **Selectors** have 3 categories: Element, ID, or Class (*discussed later in this chapter*).

CSS **Rules** apply to a selector or style being reformatted with properties and values in the header of the HTML. A stylesheet is a group of rules.

Example:

```
property:value;
color:#696;
```

Note: Stylesheet Rules are applied from the top-down. Rules that occur later in the Stylesheet override or update earlier rules.

Types of CSS Selectors

Element

CSS Rules can be applied to an **HTML Element** (*tag*) such as the `<p>` or `<h1>` tags and **Selectors** are used to identify that rule. These are my favorite way to define the formatting for large portions of text and they redefine all occurrences of that element on the page. In the CSS you identify the Element and re-define it's style settings.

Example:

```
h1 {  
    font-family:Verdana, Geneva, sans-serif;  
    color:#669966;  
}
```

ID

The 2nd type of Selector is **ID Selector**, as mentioned earlier. With ID selectors you are identifying unique areas of the page that are named within any tag. IDs can only call to one element on a page.

IDs aren't limited to DIV & SPAN, but they are good examples of how IDs can be used. SPAN tags are more finite than DIV, they can specify a formatting change on a single character. The DIV tag defines a **division/section** in a document. Standard DIV attributes include: id, class, title, style, dir, lang, xml:lang

Example:

```
<div id="copyright">Copyright 2008 - Kelly McCathran</div>
```

Example:

```
<style type="text/css">  
    #copyright {  
        font-family:Verdana, Geneva, sans-serif;  
        font-size:.7em;  
        text-align:center;  
        color:#999;  
    }  
</style>  
<div id="copyright">Copyright 2008 - Kelly McCathran</div>
```

Class

The 3rd type of Selector is a **Class selector** which can be applied to any text in the HTML, regardless of the tags used to format the text. All Class selectors start with a period (.) and can be used over and over again in the same page.

Example:

```
<style type="text/css">
.quote {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-style: italic;
    color: #666;
}
-->
</style>

<p class="quote">If you can't lead by example, at least be a horrible warning.</p>
```

Note: HTML Elements can have both Class & ID properties assigned to them:

- *Class settings override default Element properties*
- *ID settings override Class and default Element properties*

3-Digit Hexadecimal Values

The three-digit hexadecimal color value can be listed in the form #RGB, where RGB is a three-digit number that can be expanded to define the six-digit color. In this usage, each digit is repeated once.

Example:

#RGB maps to the color #RRGGBB

#696 maps to the color #669966

CSS3

CSS3 along with HTML5 provides a more powerful way to adapt print publications to a rich digital format, far beyond the original capabilities of the first generation ePub (for eReading devices).

Rounded CSS Corners

A popular request when styling a page is to add CSS rounded corners or “*the CSS corner*.” CSS corners that aren’t square are in high demand for essential elements in web page design and app design. The elements that absolutely need rounded corners are commonly the “*chrome*” of a web application or page. Chrome is not only the name of Google’s browser, but it is also a generic term referring to outer parts of a window or app. Items that can be classed as chrome are: scroll bars, buttons, visually attractive tabs and dialog boxes. Rounding the corners give a more visually-polished look and feel to your UI (*user interface*) and enhance the overall UX (*user experience*).

Until CSS3, rounded corners had to be created by using tables and tiny rounded images that were placed in each corner. While this worked in EVERY browser, it was cumbersome to write the code (to say the least). From this design need many alternate methods sprung up, blog posts claiming “this is the way to do it”, dedicated JavaScript libraries and many JQuery plug-ins.

Today we use the **border-radius** property in CSS3—Curve Radii to accomplish rounded corners. Example:

```
p {  
    border-radius: 20 px;  
    background: #696;  
    margin-left: 80;  
    margin-right: 80;  
}
```


Rounded Corners for Multiple Browsers

To compensate for the difference among browsers, many people create a CSS Class with selectors for multiple browsers.

Example

```
.rounded-corners {
  -webkit-border-radius: 30px;
  -moz-border-radius: 30px;
  -o-border-radius: 30px;
  border-radius: 30px;
}
```

CSS Border-radius Prefix

-webkit- is for Chrome & Safari

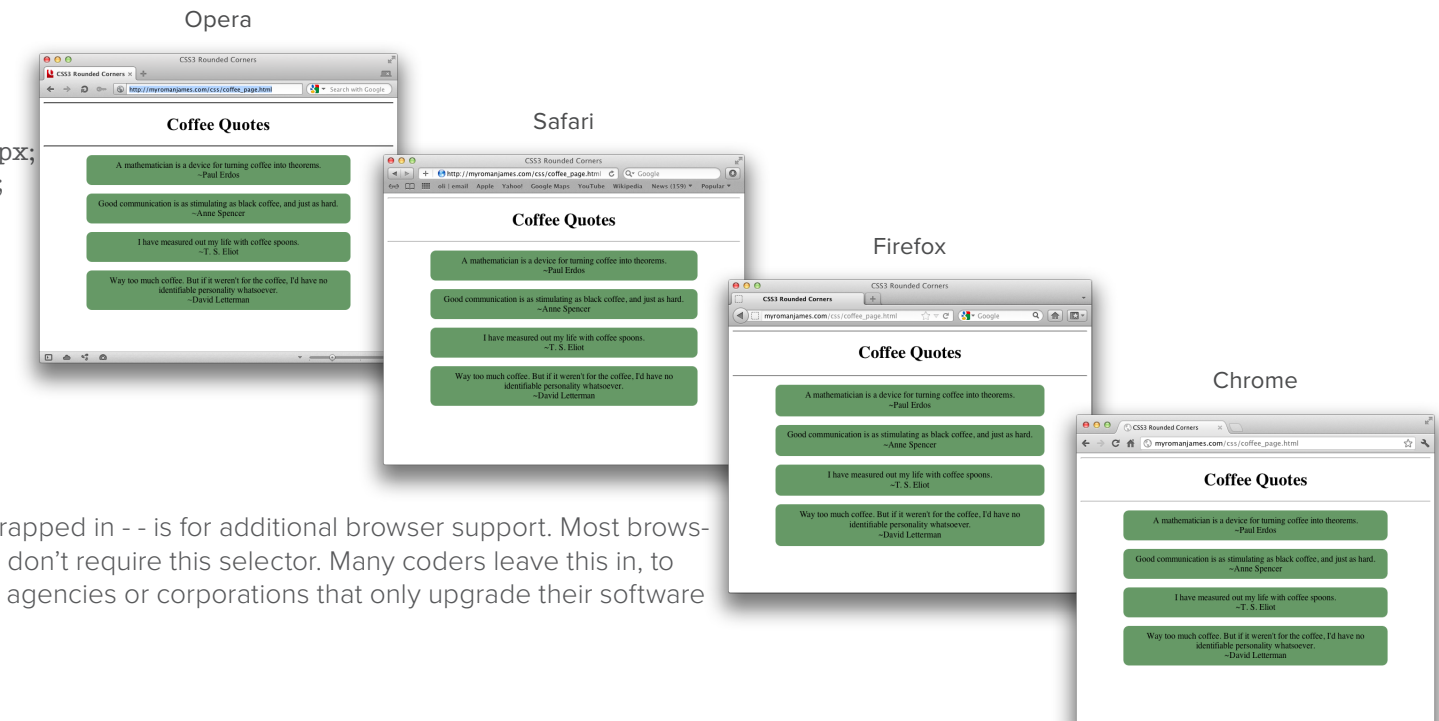
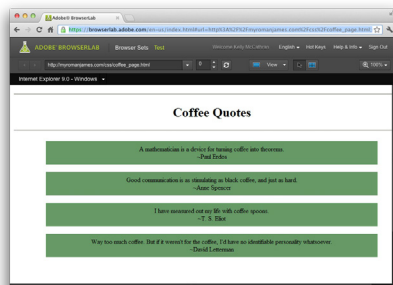
-moz- is for Firefox

-o- is for Opera

In the example above the prefix wrapped in - - is for additional browser support. Most browsers released or updated after 2011 don't require this selector. Many coders leave this in, to compensate for large government agencies or corporations that only upgrade their software every 5 years.

Rounded Corners for IE9

IE9 in Adobe® BrowserLab



IE8 and older simply do not support border-radius. IE9 is reported to, it doesn't work with the above CSS. There are many recommendations for extra code to get this to work (adding to the CSS, HTML, using a plug-in, or writing JavaScript). None of which I believe is worth the time here.

Media Query Intro

In order to deliver ideal content to every screen you need to serve different code to different devices, this is accomplished using **CSS3 Media queries**. A media query allows you to define special rules that apply to devices at certain **width break points**. For example, if the width is larger than x, then use this CSS. Media queries contain a media type and one or more expressions. Keywords such as “and” “not” or “only” help identify exactly when styles should be applied.

CSS for Media All

```
div.sidebar {  
    width: 600px;  
    color: #fff;  
    background: #333;  
    height: auto;  
}  
  
@media all and (max-width: 800px) {  
    /*styles assigned when width is smaller than 800px*/  
  
    div.sidebar {  
        width: 300px;  
    }  
}
```

HTML for DIV

```
<div class="sidebar">  
<h1>Sidebar</h1>  
</div>
```

In the above code, the sidebar's default width is 600 pixels. If the width is lower than 800 pixels, the sidebar becomes 300 pixels.

Flexible or Fluid Grid = The Formula

It is important to think of your page content like water, that can pour into many different screen sizes. A flexible grid can ensure that your content works, as intended, on any screen that views it. Creating separate fluid layouts for each breakpoint would be a lot of work, to avoid that we will use 1 flexible grid. A **grid** is typically defined as a set number of **columns** with **gutters** on either side. The columns and gutters are typically specified using **CSS classes**.

If the context size of most websites is 960 pixels wide, we can use this in our formula to get relative percent instead of fixed pixels:

target ÷ **context** = **result**

target = current viewport | **context** = original goal

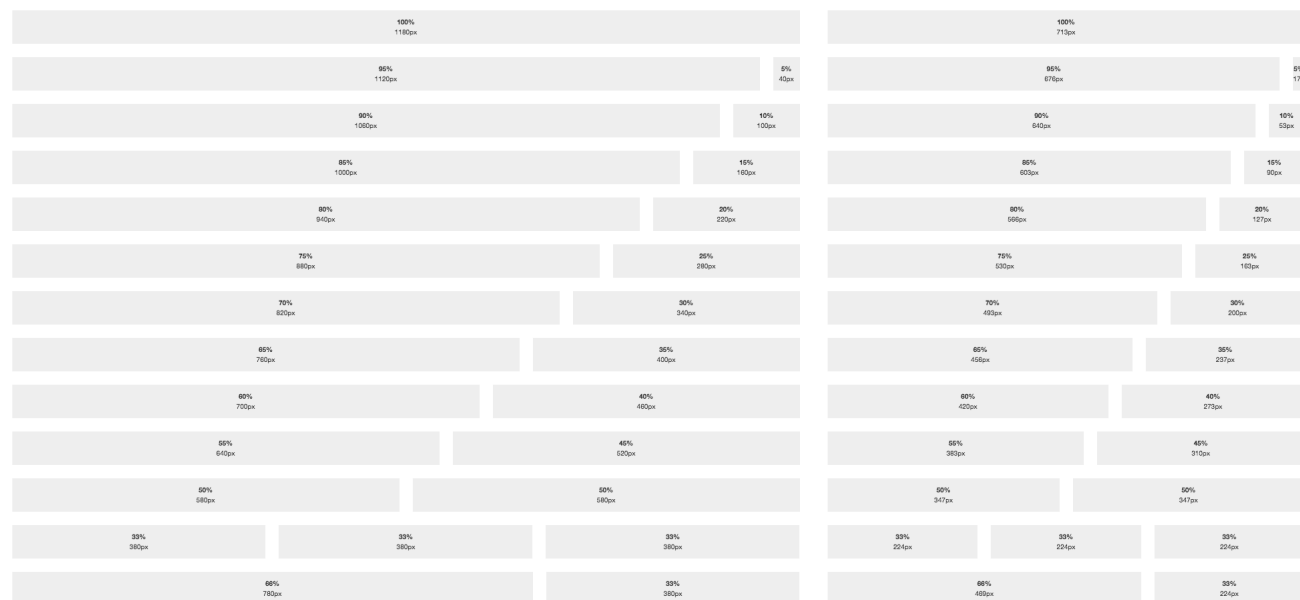
Responsive Demo

<http://unsemantic.com/demo-responsive>

To start we'll need a container for the entire page, using 90% as an example, the container will expand and contract with the viewport. If we center that container on the page, the left and right margins will be at 5% on each side.

Now, if the target width of a the **main section** of your page is 900 pixels and the context (*or default goal*) is 960 px we can use the formula above, plugging in these numbers:

$900 \div 960 = 0.9375$ | Convert that to percent: **93.75%** to use in our CSS.



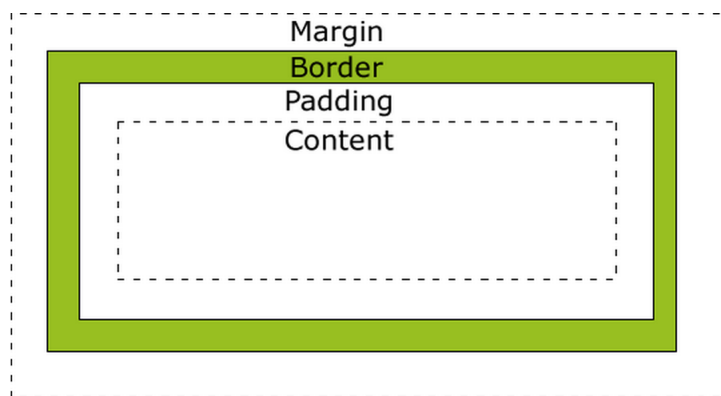
Flexible Margins or Flexible Padding

The **context**, when setting flexible margins or padding is slightly different for each.

1. When setting **flexible margins**, your **context** is the width of the **element's container**.
2. When setting **flexible padding**, your context is the width of the **element itself**.

CSS Box Model Review

Think back to the CSS Box Model when calculating the padding, in relation to the box (*or content*) itself.



Reset CSS

All browsers have presentation defaults for paragraph, headings and other content, but no two-browser families have the same defaults, for this reason we remove (or reset) built-in settings from the CSS. Eric Meyer has done extensive research and put together a public domain Reset CSS file.

Why reset browser css: <http://meyerweb.com/eric/thoughts/2007/04/18/reset-reasoning>

Get the Reset CSS from meyerweb.com: <http://meyerweb.com/eric/tools/css/reset>

The Viewport

The viewport meta tag was introduced by Apple (*in Safari Mobile*), and later adopted and developed by others. Mobile browsers display pages in a virtual “*window*” which is technically the **viewport**. This viewport is usually wider than the actual screen, so the device doesn’t need to scale every page to fit in a tiny window. Creating a viewport that is wider than the actual screen allows users to **pan** and “**pinch-zoom**” to see different areas of a page.



Technically the viewport constrains the `<html>` element (*the parent block of your web page or site*). Keeping that logic in mind, the width of your `<html>` element, on mobile, is restricted by the width of the viewport. Ideally, the `<html>` element takes 100% of the width of the viewport. The viewport is not an HTML construct, so you can’t manipulate it using CSS

The viewport meta tag allows web developers to control the size and scale of the viewport.

```
<meta name="viewport" content="width=device-width">
```

Layout Viewport

The layout viewport is considerably wider than the visual viewport. Think of the layout viewport as the area that is zoomed out. It does not change size or shape. The layout viewport is what allows mobile users to pan, and pinch zoom.

Mobile Viewport or Visual Viewport

The **mobile** or **visual viewport** is a smaller **frame** that you are seeing through to the **layout** viewport. You can zoom in and out of this “frame,” change orientation, and move around — but the layout viewport never changes.

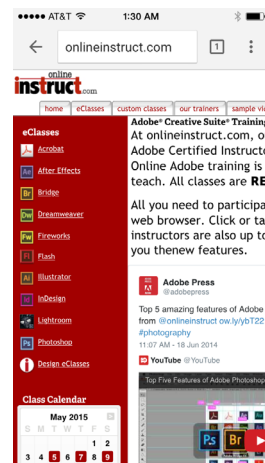
When you load a web page on a mobile browser, it will typically assume you’re viewing a desktop site, and that you want to see all of it, not just the top left corner. If a non-responsive site is encountered on a smaller device, often the viewport will automatically scales the site to fit the tiny screen.

The default viewport size for iOS is 980 pixels, if you don’t use the `meta name=“viewport”` tag, attribute, and value.

Static Site



Pinch Zoom Viewport



Viewport Control

Viewport Meta Tag

If your mobile design is intentionally built to 320 px wide you can specify the viewport width:

```
<meta name="viewport" content="width=320">
```

To match your layout width to exact size of the device, use **width=device-width**. If you use the code below to set the **width** to **device-width** (or 100% of the device) the browser **automatically** sets the **initial scale** to **100%**.

```
<meta name="viewport" content="width=device-width">
```

To be certain that your layout will be displayed as you intended, you can set the **initial-scale=1** (or zoom level to 1:1). This will ensure when the page is opened, your layout will display properly at **1:1 scale** whether rotated to portrait or landscape. Specifically on the iPhone, no re-zooming will happen.

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

If you'd like to **prevent** any **zooming** by the **user** set the **maximum-scale=1**. (This may not be recommended, since you have no way of knowing if the viewer can read everything as well as possible at a 1 to 1 ratio):

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

Recommended Viewport Meta Tag

If you don't want to prevent the user from scaling (which is a kindness for the visually impaired), leave off the **maximum-scale=1**. This is the common, or recommended, tag:

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

Viewport Width & Height Attributes

width — The width of the **virtual viewport**.

device-width — The **physical width** of the device's screen.

height — The **height of the virtual viewport**.

device-height — The **physical height** of the device's screen.

iPhone Widths

In calculating widths, we should consider the iPhone first, since it is by far the most popular brand of smart-phone. The iPhone 1 – 5 have a maximum width of **320 points** in the portrait orientation. The iPhone 6 has a width of **375 points** and the 6 Plus **414 points** (*hit is a higher pixel density display*). Yes, I am quoting **points** here, **not pixels**. Drawings begin as points, and points are referred to for a mathematical coordinate space, then devices render points into pixels. When the points are rasterized, they are multiplied by a scale factor, to get their coordinates on the screen. Higher scale factors produce more on-screen detail. The **iPhone 2 & 3** had a **scale factor** of **1**, with a resulting resolution of **320 x 480**. The **iPhone 4** had a slightly smaller screen (*in physical dimensions*) than the **iPhone 5**, but both had a **scale factor** of **2**, resulting in a resolution of **640 x 960** for the iPhone 4 and **640 x 1136** for the iPhone 5. The iPhone 6 Plus gives us a higher **scale factor** of **3**, that we will also have to consider when designing responsive. There are a ton of other **“non-Apple”** devices out there, and they have a wide variety of viewport widths.

Source: <http://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions>

Viewport CSS

Another way to control the viewport is with the `@viewport` CSS rule, which is newer than the viewport meta tag. It is important to place the `@viewport` CSS rule before any media queries. Most designers make it the 1st rule or near one of their top-level styles.

```
@viewport {  
    width: 480 px;  
    zoom: 1;  
}
```

Viewport Width (vw) in CSS3

CSS3 added vw (viewport width) and vh (viewport height). 10 vw = 10/100 of the current viewport width or 10% of the width. 20 vh = 20/100 of the current viewport height or 20% of the height.

You might wonder why we need another way to determine width when we could just specify `div { width: 50%;}`, well the `body` width does not include **margin**. `Body` height is **dependent** on the amount of **content** on the page (*not the dimensions of the browser window*). Finally, **percentage width** of `body` can not be applied to **font size**. For example, a font size of 20% sizes the font relative to defaults, not the dimensions of the viewport.

Is a Pixel a Pixel?

Before digging into this topic, I recommend reading “A tale of two viewports — part one”

<http://www.quirksmode.org/mobile/viewports.html>

Hardware Pixels (or Device Pixels)

The values of `screen.width` and `screen.height` are the total width and height of a users display. These are measured in device pixels (*or hardware pixels*), and are fixed. Screen resolution is a feature of the monitor and not the browser. This is not as important to responsive design (*when it comes to the desktop*) as Window size. Web developers are not interested in the device width (*for the most part*); it is the width of the browser window that is the most crucial piece of information.

Reference Pixels (or CSS Pixels)

The pixel isn't as fixed as we once thought it was. The proliferation of devices with varying ppi resolutions has made the pixel less exact. CSS Pixels (or Reference pixels) are not the same as device pixels, especially when viewing distance, retina and HiDPI displays are used, but we can predict their size to create responsive content. Reference pixels are defined by the W3C as:

“The reference pixel is the visual angle of one pixel on a device with a pixel density of 96dpi and a distance from the reader of an arm's length. For a nominal arm's length of 28 inches, the visual angle is therefore about 0.0213 degrees. For reading at arm's length, 1px thus corresponds to about 0.26 mm (1/96 inch).”

Source: <http://www.w3.org/TR/CSS2/syndata.html#length-units>

Using `device-pixel-ratio` media query can identify devices with scaled pixels. For example: the iPhone 4 has a `device-pixel-ratio` of 2, so it measures pixels as two-times the size of a hardware pixel. Many Android devices have a `device-pixel-ratio` of 1.5, which will scale objects one-and-a-half times larger than the hardware pixel.

Screen Density

Hardware Pixels are the smallest point a screen can display. Higher density displays can show roughly **double the pixels** in the same area. If you don't accommodate for these newer screen densities, the website & content could display as half the size.

Reference Pixels (*or CSS Pixels*) is a unit of measure that establishes an optical standard for the length of a pixel, and is totally **independent** of **hardware pixels**. The commonly accepted value is $\frac{1}{96}$ of an inch.

Device-Pixel-Ratio

Each **HiDPI** or **Retina** devices will have a device-pixel-ratio value, which is a scaling factor applied to reference pixels, so that they map more closely to hardware pixels. **Apple's iOS** devices typically have a device-pixel-ratio of 2, which means the screen density is exactly double.

Android, however, uses four basic device-pixel ratios:

- Low Density of 0.75 when lower than 120 pixels
- Medium Density of 1, for screens up to 160 pixels
- High Density of 1.5, for screens up to 240 pixels
- Extra High Density of 2 for screens up to 320 pixels

Window Size

Any responsive page will need to know the inner dimensions of the browser window, meaning, is the page maximized, do they have bookmarks along the top or favorites on the right side. To find these values you can use `window.innerWidth` and `window.innerHeight` or `document.documentElement.clientWidth` & `document.documentElement.clientHeight` (*the case for using one over the other can be a bit technical, JavaScripters will understand the differences*). Keep in mind that measured width and height do include the scrollbars, which are technically part of the window. Scrollbars average between 17 – 20 pixels in width and height. To be safe, you should typically deduct 20 pixels for scrollbars.

Break Points

When moving from a fixed width layout to it is necessary to establish break points for key viewport sizes. These break points will form the basis for our CSS3 Media Queries that “trigger” CSS style rules on the screen at a specific viewport width.

- 320 px for common smartphones including iPhone in portrait orientation
- 480 px and below for iPhones in landscape orientation and many of HTC devices
- 768 px – 980 px for iPad and other tablets
- 980 px and up for desktop monitors
- 1200 px and up for larger desktop monitors

Layouts for Each Breakpoint

Simple Design, Few Changes = Layouts for Each Breakpoint. If your design needs are simple and un-likely to change often, you may want to build layouts for each break point. Those layouts can be tailored to fit nicely on several different screen sizes.

Framework for Specific Needs

Complex (or Deep) Designs, Many Variations = Framework for Specific Needs. If your design has more complex elements and a wide range of page types, you may need to craft a framework for your site’s specific needs. The framework consists of each varying layout’s design, at the different break points. These frameworks may be just a **layout.css** applied differently at different break points, or site wide (*template-based*) content that includes: **HTML**, **JavaScript**, **CSS** and **Web Fonts**.

Custom Framework Concerns

If you select a template-based framework, you may encounter the following issues:

- There may be a lot of custom, non-semantic class names (*messy HTML & CSS*).
- You may need specific container or clearing elements, which add code to your pages.
- The templates are often large in size and sites rarely use ALL the code that is included, leaving you with bloated code.

Relative Type Sizes

When given a target size for type (*ideal*), that needs to be made responsive, you use the same formula that we used for fluid grids: In most cases 1 em = 16 pixels below is a real-world example of the formula:

target ÷ **context** = **result**

A **target** of **24 px** divided by **16 px** (1 em, or 100% of the default font size, ultimately the **context**) = **result** of **1.5 em**.

Proportional Type Sizes

Unlike responsive images that scale vertically as the width of their content column adjusts, text set in pixels does not automatically scale as the window re-sizes, it wraps. That can cause a number of layout issues and often make lines of text that are either: too short or too long to read easily. If you have text that you would like to make responsive (**and proportional to the original size**) you can apply the same formula we used above. This time; however, the **context** will be **24 pixels**. If our goal **at 100%** is **11 pixels**, the formula would be:

target of **11 px** divided by **24 px** (the goal size of our **context**) = **result** of **0.458333333 em**.

“Design is the fundamental soul of a human-made creation that ends up expressing itself in successive outer layers of the product or service.”

— Steve Jobs, 2005

Web Typography

A More Modern Scale for Web Typography | Article

<http://typecast.com/blog/a-more-modern-scale-for-web-typography>

Flexible Images

Images created and sized to look best on desktop screens are not ideal for smartphones that may have a maximum width of 320 or 240 pixels. The opposite is also true, if you enlarged images optimized for mobile, you would see giant pixelated photos.

Using fewer **raster** (or bitmap) images can benefit your responsive layout. Replacing raster images with **SVG** (*Scalable Vector Graphics*) or using **newer CSS**, such as CSS Transform, can make your pages more responsive. A drawback to SVG is that they can be larger than raster images and they need to be redrawn with each screen change. A concern in using newer CSS is that it may not be universally supported.

Icon fonts are another way to create scalable graphics (*that are in fact a real text, or a web font*).

3 Factors in Determining Responsive Images

There are 3 primary considerations when preparing for responsive images:

- Screen size (*dimensions in pixels for width & height*)
- Screen density (*72, 96 or higher ppi*)
- Bandwidth (*if you serve higher ppi images to higher density displays, you wouldn't want to do that over slower connections*)

Flexible Image Concerns by Device

Smartphones



- Small screen size
- Low bandwidth
- High **Latency** (*long delays incurred in processing network data*)
- More likely to be HiDPI or Retina

Tablets



- Medium screen size
- May be high or low bandwidth
- Possibility of HiDPI or Retina

Desktop / Laptop



- Often larger screen size & resolution
- Could be low or high bandwidth (*more often high*)
- Possibility of HiDPI or Retina

Raster Images on HiDPI or Retina Screens

For raster images in responsive layouts, you have a few options for high-density screens. The basic formula is this: if you need an image to be 100 pixels x 100 pixels, you create the image at the scaling factor that's appropriate for the targeted screen. For example, an **iOS** device with a **Retina display** that has a **scaling factor of 2** your image should be 200 pixels x 200 pixels. It's important to note here that PPI does not matter; it's only about the physical amount of pixels. When targeting the high-density screen, you use the 200 x 200 pixel image and resize it to fit the 100 x 100 pixel desired size.

This means that the end user will have to download **larger images** and not all screens have an easy scaling factor of 2 (*an example is the 4 different screen densities of Android devices*).

Responsive Background Images

Background images of different sizes can be swapped in or out using the @media and device-pixel-ratio queries. Support for device-pixel-ratio isn't available across all platforms and may require vendor prefixes.

```
@media screen and (device-pixel-ratio: 2) {  
  .highres {  
    background: url (image_highres.jpg); background-size: 50%;  
  }  
}
```


Responsive Image Options

1. **Replace fixed dimension images with percentages** (or relative unit values) — Instead of specifying a fixed width & height in the HTML, you can use width 100% or the CSS property of max-width: 100%.

```

```

When doing this, you should specify a **maximum-width** property to avoid scaling the image beyond its ideal dimension. To do this, you could place the image in a **flexible container** and use **max-width = 100%** property to constrain the image to the size of the container (*this trick works with video also*).

```

```

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

Technical Note: Internet Explorer doesn't support max-width.

This method also makes it necessary for you to serve the largest image that your layout needs, and applies scaling down to fit your grids. This means that mobile devices (often on cell phone networks) will be downloading larger images than necessary).

Img Tag Images

Swapping context-based images, using the `` tag requires **JavaScript** (see option #3 for more details).

Note:
Width in **HTML** specifies the **width** of the **element**. This is different than **max-width** in **CSS**, which specifies the width "not to exceed."

2. **Load the image as a background** — If the image doesn't affect the structure of your page, you could load the image as a background, specifying different background images for each break point.

```
@media (max-width: 480 px) {  
    div.swapImg {  
        background: url ("phoneImage.jpg")  
    }  
}  
  
@media (min-width: 481 px) and (max-width: 480 px) {  
    div.swapImg {  
        background: url ("tabletImage.jpg")  
    }  
}  
  
@media (min-width: 779 px) {  
    div.swapImg {  
        background: url ("desktopImage.jpg")  
    }  
}
```

This method only works with images that don't need to be in the actual HTML. Also, if your page or content needs to be accessible (*have the ability to be read out loud for the visually impaired*), this solution won't work.

Technical Note: *On older Android Devices (running 2.x or older), ALL CSS background images are downloaded, not just the one required for that media query. This is fixed for newer Android devices.*

3. **Use JavaScript and/or server-side scripting** — There are many elegant & robust solutions that use JavaScript or server-side scripting to serve the proper image, at the proper size for the device that is viewing the page. One free solution is **Picture Fill** by the **Scott Jehl**: <https://github.com/scottjehl/picturefill>

There are many solutions out there and they work on the same basic principle:

- Have a script parse the page for context
- Replace the smaller images with larger images (*where appropriate*)

Drawbacks to this method are the complexity, slow load times and often; the replacement images are requested after the original images have already been downloaded.

Browser Prefetching or Link Prefetching

There is another caveat to the aforementioned responsive image solutions, many modern browsers have a mechanism that **pre-loads** or “*prefetches*” **documents** or **assets** that the user might need in the near future. Based on hints from the web page, after a page has loaded, a browser may silently prefetch specified documents and store them in the cache. The “hints” that the browser looks for are the `<link>` tag or **HTTP Link:** header with a relation type of either next or prefetch. **Meta** tags with `http-equiv=“Link”` are also browser hints to prefetch. When the browser finds these hints, it queues up each unique request **when the browser is idle**.

Prefetching is part of the scope of **HTML5** and is working towards standardization for all browsers supporting newer HTML. Even HTML 4.01 allows for new link relation types and does not violate any web standards.

Technical note: *Currently, `<a>` anchor tags are not prefetched. Secure content (`https`) can not be prefetched.*

The Future of Responsive Images | or <picture>

The WhatWG (see the *HTML* section for more details on [whatwg.org](https://html.spec.whatwg.org) or check <https://html.spec.whatwg.org>) has a new standard of **srcset** (like the attribute) can list several images to be used in one tag for **device-pixel ratio-based** images. Example:

```

```

Technical Note: In the above example 2x and 1.5x represent the pixel depth ratio.

A competing solution, proposed by the W3C's **Responsive Images Community Group** is the <picture> element. The <picture> element (tag) works as a parent for a range of source element and an tag. **Picture** also works with a <source> tag and **srcset** attribute and can contain **media** queries and **alt** text.

Support for SRCSET

Caniuse.com tracks many newer HTML features, see which browsers currently support this attribute: caniuse.com/srcset. Currently, the newest versions of these browsers support **srcset**:

- Chrome
- Firefox
- Safari (*not fully supported, sizes using the w descriptor don't work*)
- iOS Safari (*not fully supported, sizes using the w descriptor don't work*)
- Opera
- Android 5.x
- Chrome for Android
- IE Edge is also **plans** to support **srcset**

Support for <picture>

Caniuse.com is tracking <picture>: caniuse.com/picture. The newest versions of these browsers support **picture**:

- Chrome
- Firefox
- Opera
- Android 44
- Chrome for Android

Note: IE Edge & Safari on desktop & iOS do not support <picture>.

Picturefill Responsive Image Polyfill

The picture element is a newer W3C standard that looked promising to web developers, offering the hope of delivering an appropriate image to every device, depending on conditions like screen size, viewport size, screen resolution and more. Picturefill is a JavaScript file that technically works as a **polyfill** on the **<picture>** tag or element.

Article | **Picturefill 2.0: Responsive Images And The Perfect Polyfill**

<http://www.smashingmagazine.com/2014/05/12/picturefill-2-0-responsive-images-and-the-perfect-polyfill>

Responsive Forms

Forms have always been a bit of a “*pickle*” for web developers. No matter how wide you make your form field width, the size can change slightly from browser to browser. When you open a form on a smaller device, you throw another wrench into the mix. There are some best-practices that will help make them more responsive. There is a large movement to “*mobile-first*” forms.

Tips for Forms

There are several considerations to help you plan responsive forms:

- Structure your forms to prepare for responsive (eliminate tables).
- Use structural elements like list and field sets, for better form mark-up.
- Consider how the form layout needs to change from one device to another.
- Labels next to form fields are fine for larger screens, but when you scale down, the labels may need to go on top of the input field.
- Since users tap with their finger to fill out forms, be sure to include enough distance in your form field (*at least 20 px*), so they don't tap the wrong area.
- Wrap the `<label>` tag around the `<input>` tag, this makes both the input and the label responsive to clicking or touch.
- Use newer HTML5 attributes such as email, search and url. These supply the users with common characters such as @ for email fields and .com for url.
- Use placeholder text for feedback to the user.

Required Reading

A Dao of Web Design

by John Allsopp

<http://alistapart.com/article/dao>

Resource Web Sites

The Graphical Web

<http://thegraphicalweb.com>

The Expressive Web

<http://theexpressiveweb.com>

WhatWG | Standards body for HTML5 and More

<http://whatwg.org>

HTML a Living Standard | WhatWG

<https://html.spec.whatwg.org>

CSS Resources

World Wide Web Consortium, Cascading Style Sheets home page:

<http://www.w3.org/Style/CSS/>

Tutorial for Cascading Style Sheets:

<http://www.w3schools.com/css>

See what CSS Can do:

<http://CSSzengarden.com>

HTML5 Resources

List of New Elements in HTML5

<http://www.w3.org/TR/html5-diff/#new-elements>

Adobe HTML5, CSS3 and JavaScript Archive

<http://www.adobe.com/devnet/html5.html>

Free HTML5 Templates | HTML5 UP

<http://html5up.net>

HTML5 Forms

A FORM OF MADNESS

<http://diveintohtml5.info/forms.html>

Browser Testing of HTML

There are many sites that can check HTML and CSS to validate your code, be sure to test against a site that is platform independent and not Microsoft-centric. Here is my shortlist:

W3C Markup Validation Service

<http://validator.w3.org>

HTML5Test

<http://html5test.com>

Adobe Edge Inspect

<http://creative.adobe.com/products/inspect>

On the Windows platform you can typically only run one version of Internet Explorer, here is a website with instructions on how to install multiple versions of IE:

http://tredosoft.com/Multiple_IE

Responsive Website Examples

The Boston Globe

<http://thebostonglobe.com>

RocketLawyer

<https://www.rocketlawyer.com>

Responsive Web Design Site by Ethan Marcotte

<http://responsivewebdesign.com>

Responsive Demo from unsemantic.com

<http://unsemantic.com/demo-responsive>

Responsive Design Resources

Responsive Web Design I by ETHAN MARCOTTE May 25, 2010

<http://alistapart.com/article/responsive-web-design>

RESPONSIVE DESIGN.is

<https://responsivedesign.is>

Deliver Faster Responsive Web Design Sites | Akamai PDF

Can you optimize website performance for Responsive Web Design sites?

<https://content.akamai.com/PG1127-HowToDeliverFast.html>

Google Developers | Web Fundamentals

<https://developers.google.com/web/fundamentals>

A Tale of Two Viewports Part 1 (Desktop)

<http://www.quirksmode.org/mobile/viewports.html>

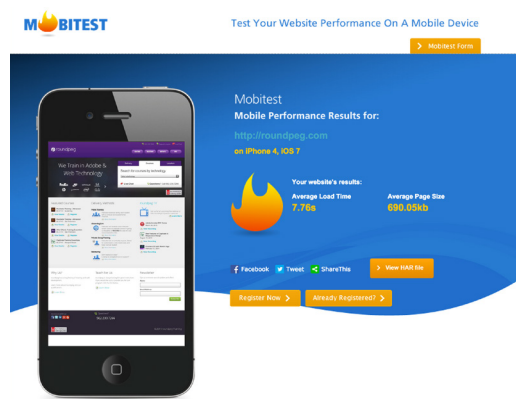
A Tale of Two Viewports Part 2 (Mobile)

<http://www.quirksmode.org/mobile/viewports2.html>

Responsive Testing

Free Mobile Web Performance Measurement Tool | Akamai

<http://mobitest.akamai.com>



Responsive Images

Picture Fill by the Scott Jehl:

<https://github.com/scottjehl/picturefill>

Responsive Images Community Group

<http://responsiveimages.org>