

This paper was originally presented at the Southwest Fox conference in Gilbert, Arizona in October, 2018. <http://www.swfox.net>



Responsive Web Design with CSS

*Rick Borup
Information Technology Associates, LLC
701 Devonshire Drive, Suite 127
Champaign, IL 61820
rborup@ita-software.com*

As the percentage of Web traffic from mobile devices steadily increases, websites that don't adjust their content for smaller screens risk becoming irrelevant. Sites originally designed for desktop browsers may be almost unreadable on mobile devices. If the site doesn't detect smaller viewports and adjust its content accordingly, the visitor might see a postage-stamp miniature of the original or content that overflows the visible area of the screen and requires scrolling. Short of redesigning and rewriting the entire website – or even creating a separate site for mobile devices – how do you make an existing site responsive to mobile devices? Using an example adapted from a real-world experience, this session shows you how to update a

website to respond to mobile devices using only CSS and a little JavaScript. No new frameworks required!

Introduction

Websites originally designed for desktop browsers typically take advantage of most or all the screen real estate available to them. Common design elements include banner images across the full width of the screen, horizontal menu bars, two- or three-column layouts, sidebars for navigation and/or additional content, fixed-width elements, and large image files. These sites look great on a full-size desktop monitor, but their appearance and usability tend to fall off rapidly when viewed on smaller handheld devices. The goal of responsive web design is to make websites look great on every device.

You will learn

- How to structure your HTML for responsive design
- What CSS is all about
- How to use CSS to control content layout
- How to use CSS to adjust the layout for different size viewports
- How to create navigation menus for mobile devices using CSS and JavaScript
- What to do about images that are too big to fit
- About tools to help you create and test responsive designs
- What Flexbox is all about and how to start using it

What is responsive web design

Responsive web design uses HTML and CSS to adjust the content and layout of a website in response to the size and orientation of the device on which it is being viewed. The size and orientation of the device define the visible area of the screen — aka the viewport — which is expressed as a height and a width in pixels.

Figure 1 is an example of a responsive web design. A visitor using a desktop browser or a large tablet in landscape mode sees the content shown in the largest image. A user with a smaller tablet in portrait mode sees the content as shown in the middle image, while a user holding a mobile phone vertically in her hand sees the content as shown in the smallest image.

In each case, note how the browser has adjusted the size and layout of the text, image, and navigation controls so they're appropriate to the device.

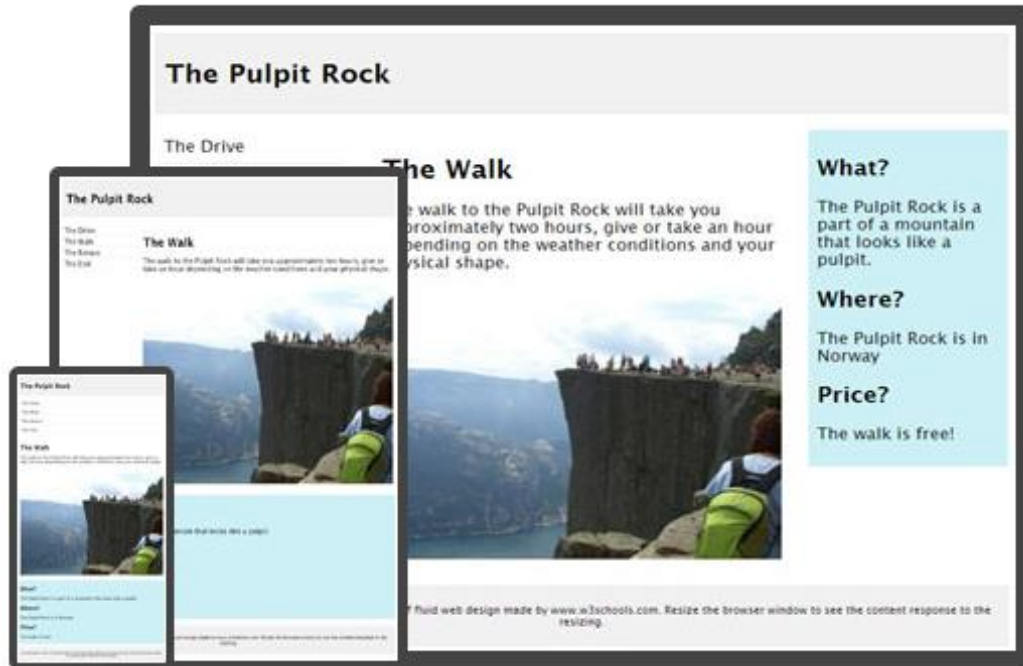


Figure 1. With responsive design, a site looks good on any size device.
(image source: https://www.w3schools.com/html/html_responsive.asp)

Why responsive design matters

Designs that use fixed-width layouts, large images, and other elements of desktop web design are not responsive by nature. This matters because on smaller devices the content either overflows the viewable area of the screen or is zoomed out by the browser so the user sees the entire page but in miniature.

Here's a very basic example from w3schools.com. **Figure 2** compares a non-responsive version of the webpage to a responsive version as rendered on an iPhone. On the non-responsive version, the image is shrunken and the text is small and difficult to read. On the responsive version, the image fills the width of the screen and the text is large enough to be more easily readable.

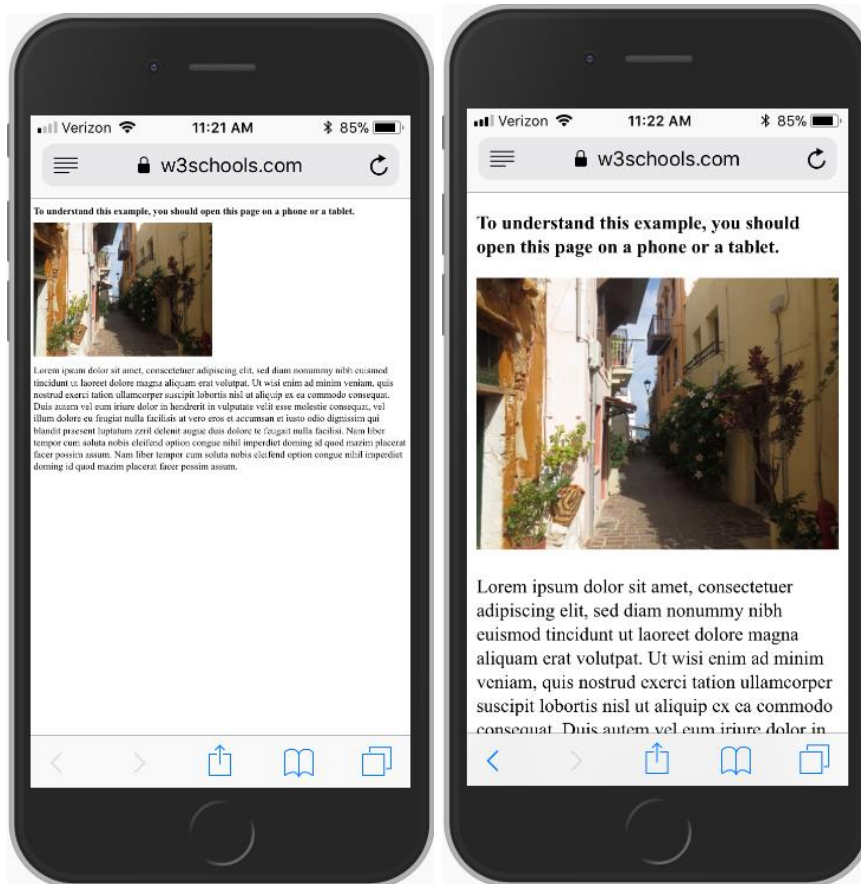


Figure 2. A non-responsive version of a webpage (left) compared to a responsive version of the same page (right).

In this example, the page was made responsive by adding a `<viewport>` meta-tag and applying a CSS style rule to the `` tag.

Listing 1: The `<viewport>` meta tag, introduced in HTML5, tells the browser to set the width of the page to the width of the device's screen.

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<style>
img {
  max-width: 100%;
  height: auto;
}
</style>
</head>
```

You can see this for yourself at www.w3schools.com/css/css_rwd_viewport.asp.

Making an existing website responsive

If you need to adapt a legacy website to work responsively on devices with smaller viewports, you have three basic choices:

- start over from scratch, using a modern framework to create a design that works well on any size device;
- create a separate companion site designed specifically for smaller screens; or
- adapt the existing design to adjust its content on the fly in response to different size devices.

All three are valid choices, but the first two may involve a lot more work.

If you decide to use a framework there are many resources to help you get started. Among them are Rick Strahl's and Doug Hennig's white papers from recent Southwest Fox conferences, which can help you learn about Angular JS and Bootstrap respectively.¹ If you're interested in Flexbox, there is a short section about it at the end of this paper plus many links in the References section. There are certainly many other choices as well.

This paper is about the third option.

Structuring HTML for CSS

This session is based on HTML5, which is the current standard and is implemented in the current versions of all major browsers. Some of the code used for responsive design in this paper requires HTML5. If you're not familiar with HTML5, a good place to start is www.w3schools.com/html/.

There are a few dos and don'ts to keep in mind when structuring HTML for responsive design. The following guidelines do not pretend to cover *all* the changes that might be required, but if you follow these three you'll be off to a good start and you'll find it much easier to implement the CSS necessary to achieve your desired results for responsive design.

Do not use the <table> tag to control layout

Of the three guidelines, this is the most important one. Before CSS, web developers commonly used HTML tables to control the layout — the width, height, and relative position — of the structural elements on a web page. HTML tables were never intended for that purpose, but were widely used that way because there wasn't much of an alternative. For all but the simplest web pages, however, this approach quickly became convoluted and difficult to maintain, with a nightmare of tables nested within tables nested with yet other tables.

If you're setting out to adapt an existing web page for responsive design and find yourself facing code that uses <table> tags to control layout, a good first step is to restructure the layout using divs and CSS layout properties instead.

¹ See the References section at the end of this paper for titles and dates.

Here's a before and after example of a simple web page with a 200 pixel-wide navigation panel on the left and a 700 pixel-wide main content area on the right.

Listing 2: This web page uses a `<table>` tag to control layout.

```
<table width=900>
  <tr>
    <td width=200>
      <p>navigation links</p>
    </td>
    <td width=700>
      <p>main content of the page</p>
    </td>
  </tr>
</table>
```

The same result can be achieved using CSS divs and floats instead of an HTML table.

Listing 3: The left-side navigation uses *float: left* while the main content uses *float: right*.

```
<div style="width: 900px">
  <div style="width: 200px; float: left;">
    <p>navigation links</p>
  </div>
  <div style="width: 700px; float: right;">
    <p>main content of the page</p>
  </div>
</div>
```

Here's an alternative way to do the same thing.

Listing 4: Both divs use *float: left* but *margin-left: 200px, auto* is applied to the main content div so it is positioned to the right of the navigation div.

```
<div style="width: 900px">
  <div style="width: 200px; float: left;">
    <p>navigation links</p>
  </div>
  <div style="width: 700px; float: left; margin-left: 200px, auto;">
    <p>main content of the page</p>
  </div>
</div>
```

Later, you'll see how to assign identifiers to divs and use CSS for even greater control over layout in responsive design.

Do not use the tag to style text

In HTML4, the tag was used to determine the color, size, and typeface of selected text. Along with several others, the tag was removed from HTML5 in favor of using CSS to control those attributes.²

The HTML4 tag was used to make a chunk of selected text red, like this:

```
<font color="red">this text is now red</font>
```

The same thing can be accomplished with an embedded CSS style.

```
<span style="color: red;">this text is now red</span>
```

The same effect again, using a CSS class named *red_text*.

```
<style>
  .red_text {
    color: red;
  }
...
<span class="red_text">this text is now red</span>
```

Do not use the
 tag to control line spacing

Before CSS, web developers commonly used the
 (line break) tag to insert spacing between lines on the web page. While the
 tag still exists in HTML5, it should no longer be used for that purpose — there's a better way to do it using CSS.

Listing 5: Two
 tags are used to insert vertical white space underneath the lead paragraph.

```
<p>lead paragraph<br><br></p>
<p>second paragraph</p>
```

Listing 6: CSS styles offer a better approach with more granular control.

```
<style>
  .lead_paragraph {
    margin-bottom: 17px;
  }
</style>
...
<p class="lead_paragraph">lead paragraph</p>
<p>second paragraph</p>
```

An introduction to CSS

CSS is all about telling the browser three things:

- where to place content on the web page,

² See “Removed Elements in HTML5” at https://www.w3schools.com/html/html5_intro.asp

- how that content should look, and
- which content to display.

In responsive design, these things depend on the type, size, and orientation of the user's device.

CSS Rules

CSS is implemented using styles, aka rules. A CSS rule consists of one or more selectors followed by a declaration block, which in turn comprises one or more declarations.

The selector(s) tells the browser what to apply the style(s) to, while the declarations define the styles to be applied. The declaration block is enclosed in curly braces. Each declaration within the declaration block consists of a property and a value separated by a colon. The structure of a CSS rule is illustrated in **Figure 3**.

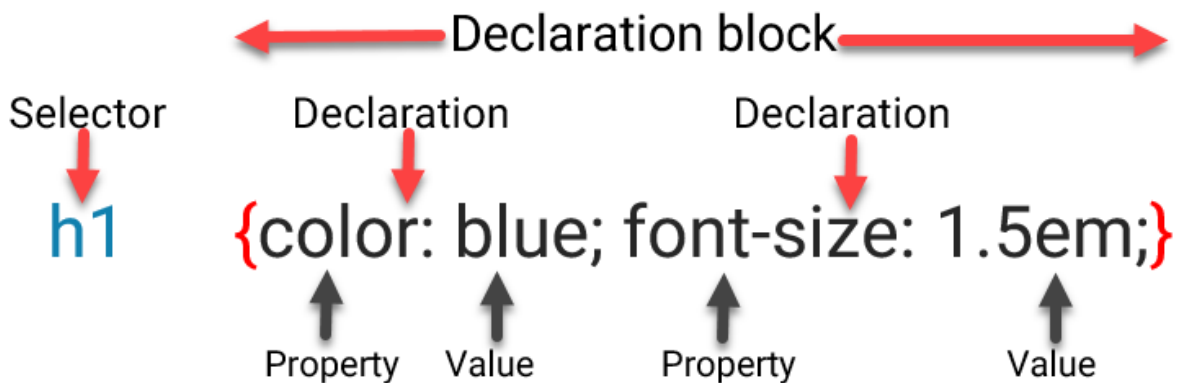


Figure 3: A CSS rule has one or more selectors followed by a declaration block with one or more declarations.

CSS Stylesheets

A CSS stylesheet is a document containing CSS rules. Stylesheets are given a .css file name extension. In stylesheets, CSS rules are typically written using the syntax style shown in **Listing 7** — a convention familiar to developers who work with other languages that use curly braces to surround blocks of code.

Listing 7: This syntax convention is commonly used in CSS stylesheets.

```
h1 {  
    color: blue;  
    font-size: 1.5em;  
}
```

A CSS stylesheet is applied to a web page by adding a link in the <head> section of the HTML document, as shown in **Listing 8**.

Listing 8: A CSS stylesheet is linked to an HTML document in its <head> element.

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="styles.css">
  </head>
```

It's often convenient for one CSS stylesheet to incorporate styles from another stylesheet. This is accomplished using the @import statement. The advantage of using @import is that the HTML page only needs a link to the one stylesheet, which in turn brings in the other.

The stylesheet in **Listing 9** shows the @import statement being used in the *styles.css* stylesheet to incorporate another stylesheet named *colors.css*. Any web page that links to *styles.css* automatically incorporates *colors.css* as well.

Listing 9: The @import statement brings in a CSS stylesheet from another file.

```
@import url('colors.css');
h1 {
  color: blue;
  font-size: 1.5em;
}
```

The order in which styles are defined in a stylesheet makes a difference because styles defined later on may override styles defined earlier. For example, in Listing 9, the *colors.css* stylesheet might style <h1> tags as red, but this is overridden by blue because the h1 style in *styles.css* is loaded after *colors.css* has been imported.

Styles do not have to reside in stylesheets — they can also be defined in the <head> element of the HTML document itself, as shown in **Listing 10**. This is convenient if there are only a few styles, but it's inefficient if there are more than a few because they would all have to be embedded in every HTML document that needs them.

Listing 10: CSS styles can be embedded in the <head> element of the HTML document.

```
<head>
  <style>
    h1 { color: blue; font-size: 1.5em; }
  </style>
</head>
```

You can also define inline styles within an HTML page. One way is to use a tag to select a block of content to which the style is applied.

```
<span style="color: blue;">This text is blue</span>
```

Another way is to embed the style within an HTML tag such as a <p> or an tag.

```
<p style="text-align:center;">some text to be centered on the page</p>
```

CSS Selectors

CSS selectors tell the browser what to apply the styles to. CSS selectors can be:

- an HTML tag (HTML element), such as `<h1>`, `<p>`, or ``
- a CSS identifier, called an ID, which starts with a hashtag symbol as in `#main` or `#sidebar`
- a CSS class, which starts with a dot as in `.small` or `.footer`
- an inline style attribute, for example within a `...` block or inside an HTML tag

In the example in Figure 3, the rule tells the browser that the `<h1>` tag should be blue text of size 1.5 em (an *em* is a unit of measure relative to the font size³).

You can assign a CSS rule to multiple selectors by separating them with commas. The code in **Listing 11** makes all `<h1>` and `<h2>` tags red.

Listing 11: This style makes the text in all `<h1>` and `<h2>` tags red.

```
h1, h2 {
  color: red;
}
```

You will frequently see descendant and child selectors defined in CSS stylesheets. A descendant selector is a selector that appears anywhere below another selector in the HTML hierarchy. Descendant selectors are defined with a space.

Listing 12: This style applies to any `<a>` tag that appears below — i.e., within — a `<div>` tag even if separated by another tag.

```
div a {color: blue;}
```

A child selector is the direct descendant of its parent. Child selectors are defined with an angle bracket.

Listing 13: This style applies only to `<a>` tags that are a direct descendant — i.e., a child — of a `<div>` tag.

```
div > a {color: blue};
```

You will also see pseudo selectors, which are defined with a colon. Pseudo selectors are frequently used to apply different styles to various states of an HTML tag, such as a link.

³ See <https://www.w3.org/Style/Examples/007/units.en.html>

Listing 14: This style makes links turn blue when the mouse hovers over them, and gray when they've already been visited.

```
a:hover {color: blue;}
a:visited {color: gray;}
```

CSS for layout

CSS can be applied to <div> tags and other structural elements of an HTML document to control their layout when viewed in a browser. The typical structural elements are an image across the top, a main content area, a navigation menu either on top of the main content or to one side, possibly other sidebar elements, and a footer.

When beginning a new website, one of the first choices is to decide on a general approach to its layout. These fall into three basic categories:

- fixed width layouts,
- fluid (aka liquid) layouts, and
- responsive design layouts.

Fixed-width layouts

In a fixed width design, the developer assigns a specific width to each part of the layout. This type of design gives the developer complete control over how the web page looks, but it is totally unresponsive to different screen sizes and looks as intended only on screens that can accommodate the full width. **Listing 15** is an example of a fixed width design.

Listing 15: This fixed-width design defines a 186px side bar to the left of the 744px main content area.

```
#sidebar {
  width: 186px;
  float: left;
}
#main {
  width: 744px;
  float: right;
}
```

Fluid layouts

Fluid design layouts, also known as liquid layouts, use a percentage instead of a fixed number of pixels to specify the width of the structural elements. Fluid designs are responsive in the sense that the browser renders the elements as a percentage of the width of their container, but this may not look good in all situations. Without additional work, the elements of a fluid layout might appear artificially wide on large screens and artificially narrow on small screens. **Listing 16** is an example of the CSS for a fluid design.

Listing 16; This fluid-design layout defines a left-hand side bar that is 20% of the width of its container and a main content area that occupies the remaining 80% to its right.

```
#sidebar {  
  width: 20%;  
  float: left;  
}  
#main {  
  width: 80%;  
  float: right;  
}
```

Some websites use a hybrid design that is part fixed width and part fluid. For example, in a two-column design the developer might specify a left-hand sidebar with a fixed width of 186px but allow the main content on the right to grow to the remaining width of the container. In a three-column design, the left-most and right-most columns might have a fixed width with the main content (in the center) allowed to grow and shrink with the width of the browser window. This type of layout is sometimes called a *liquid center* design.

Responsive layouts

Responsive web design uses CSS media queries to detect the size of the viewport and to respond by adjusting the content accordingly. On larger screens, elements can be allowed to grow without limit or they might be constrained by a maximum width. Elements that appear side by side on larger screens might be stacked vertically when viewed on smaller screens. Image sizes can be adjusted, and/or different images can be used depending on the size of the viewport. Horizontal navigation menus that work well on larger screens can be replaced on smaller screens with vertical menus that pop up when an icon is clicked or touched.

CSS media queries are implemented using the @media rule. **Listing 17** is an example of a CSS media query to suppress the sidebar element when the page is viewed on a screen that is 930px wide or less.

Listing 17: Use *display: none* to suppress showing the selected content when the conditions are met.

```
@media screen and (max-width: 930px) {  
  #sidebar {  
    display: none;  
  }  
}
```

It might surprise you to learn that fixed-width layouts and fluid design layouts can easily be transformed into responsive design. In fact, that's what this paper is all about.

The sample website that serves as the use case in this paper makes extensive use of media queries to turn what was originally a fixed-width design into a responsive design. For more information about media queries in general, see the links in the references section at the end of this paper.

There are other ways to design responsive layouts, too. One of these is flexbox, which is discussed towards the end of this paper.

The use case website – Reindeer Toys & Games

Reindeer Toys & Games⁴ is a distributor of toys and games for reindeer and the people who own them. As their business grew, the company expanded into selling reindeer food, reindeer gear (harnesses, fences, blankets, grooming supplies, etc.), as well as sleds and sleighs.

Revision 0 - the original website

Figure 5 is the home page of the original website as it appears in a desktop browser. Note the typical elements of a site designed before mobile devices were a consideration — fixed width page (950px in this case), a banner image (logo) across the top with a horizontal menu underneath, left-hand sidebar with links to complementary content, the main content area of the page to its right, and a footer at the bottom.



Figure 4: The original website was designed for desktop browsers with a viewport at least 950px wide.

⁴ Not a real company

Figure 5 shows the design template for this website, which uses CSS divs with IDs to give names to the structural elements of the layout: `#logo` is the banner image on top, `#menu` is the horizontal main menu under the banner image, `#sidebar` is the left-hand navigation panel for related links, `#main` is the main content area, and `#footer` is the footer. Although not labeled in Figure 5, the entire `<body>` tag of the HTML document is wrapped in a div named `#container`. The main page-content area in this two-column design, which comprises the `#sidebar` and `#main` divs, is wrapped in an outer div named `#body2col`.



Figure 5: The template for the original website is a two-column design that uses CSS IDs to identify the main elements of the layout.

The original website also enables online ordering, a popular feature with customers. **Figure 6** shows the shopping cart page for online ordering.

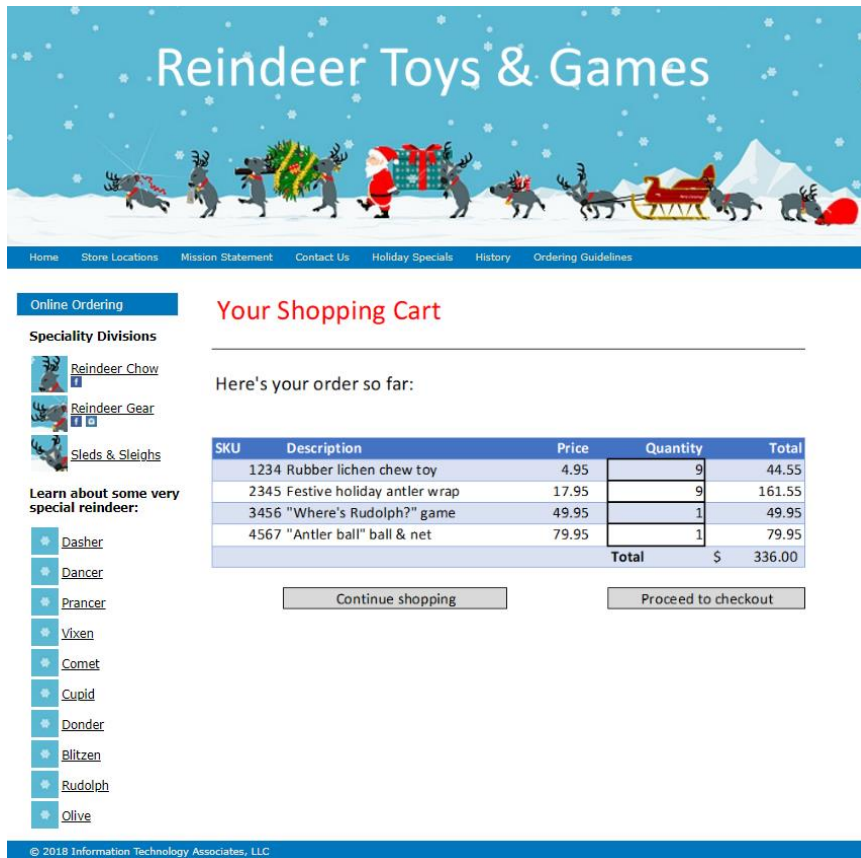


Figure 6: The online ordering page with an order in progress.

After the customer places her order, the website displays the order confirmation page as shown in **Figure 7**.

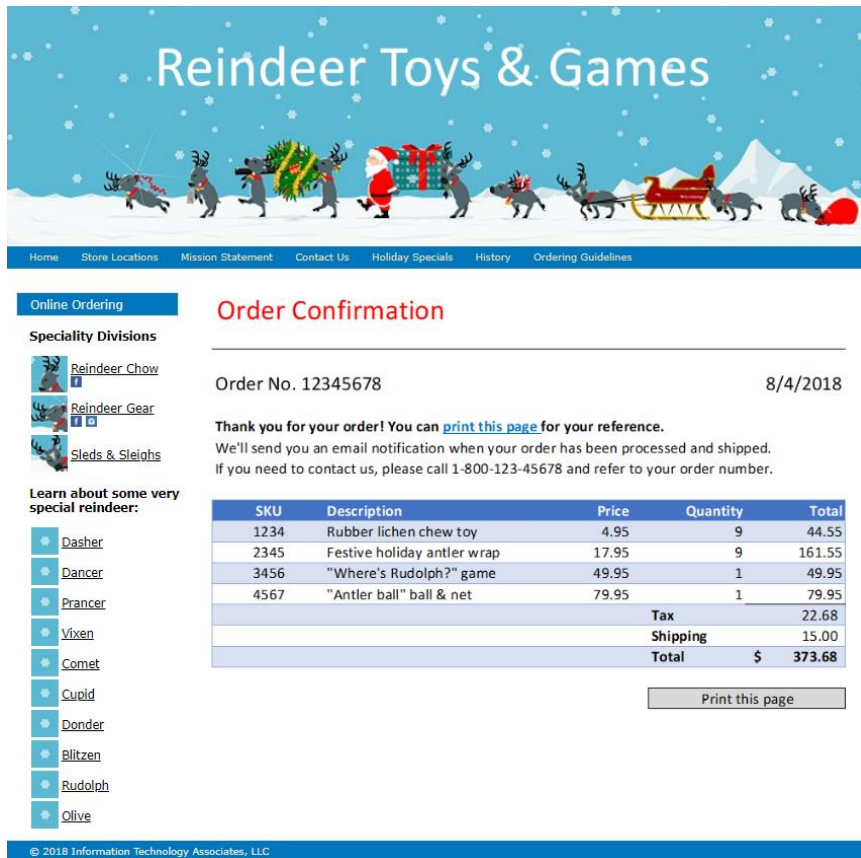


Figure 7: The order confirmation page shows the details of the purchase.

The company knew that customers like to print the confirmation page as a record of their order, so they placed a link and a button on the page for that purpose. However, complaints soon began flowing in that the printout was difficult to read because it reproduced the entire page — including the side bar and the top logo — on paper, making the order confirmation section smaller than it needed to be. What was needed was a way to restrict the printed output to only the confirmation section of the page.

CSS2 introduced the @media rule that enabled browsers to detect the type of media on which the content was being rendered - printers, handheld devices, etc. This was before the explosive proliferation of mobile devices, so the most common use of the @media rule was to implement selective content for printers. Later, CSS3 introduced media queries, which built on the concept of the @media rule to detect the capabilities of the user's device, such as screen size. Much of today's responsive web design relies on media queries, which we'll see in a subsequent iteration of the sample website.

The CSS stylesheet for the original version of the website included a media type rule to suppress everything except the order confirmation section when the page is printed. The media rule is shown in the CSS in **Listing 18**.

Listing 18: This media type rule selects the class IDs to be suppressed when the page is sent to a printer.

```
@media print {
```

```

#logo, #menu, #sidebar, #footer
  display: none;
}
#container, #body2col, #main {
  width: auto;
  float: none;
}
#body2col {
  background-image: none;
}
#sidebar, #main {
  padding: 20px;
}
}

```

Figure 8 shows how the order confirmation section now fills the entire the printed page.

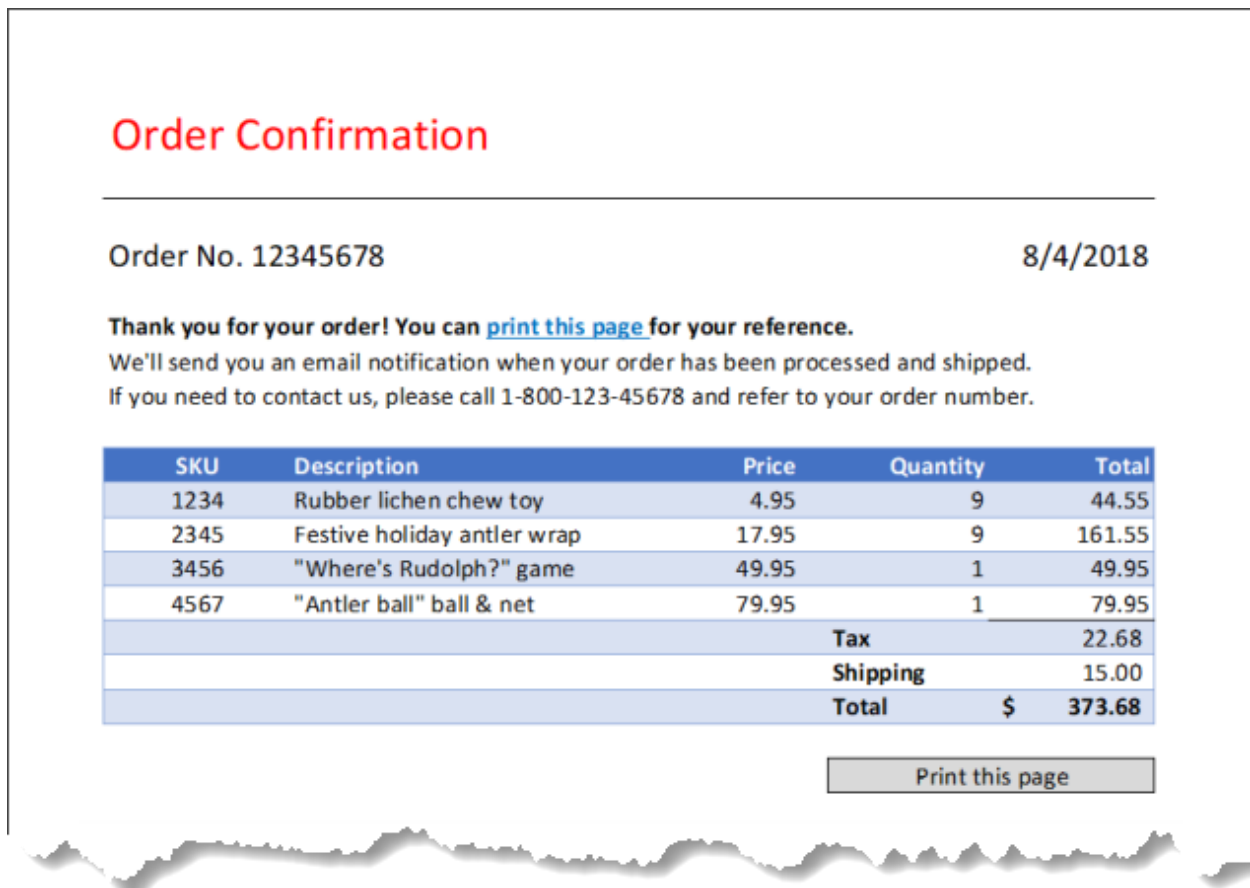


Figure 8: When printed, the order confirmation page suppresses the banner, menu, sidebar, and footer so the main content can fill the entire page.

Revision 1 – Adding a <viewport> tag

As the use of mobile devices became more ubiquitous, customers began to let the company know they wanted to be able to place online orders from their mobile devices while they were in the fields, paddocks, and barns with their reindeer instead of having to jot down

notes on what they needed and then return to a desktop computer in their home or office to place their order. This put increasing pressure on the company to make their website responsive, but the budget wasn't there to re-create the entire website from scratch and besides, the customers and the company were still happy with the original design.

The first step toward making a website responsive to mobile devices is to add a <viewport> meta tag to the HTML. Without the <viewport> meta tag, mobile browsers shrink the content so it all fits on the device's screen regardless of the screen's width.⁵

Figure 9 shows what this looks like on a simulated iPhone 6 in the Chrome browser's developer tools.

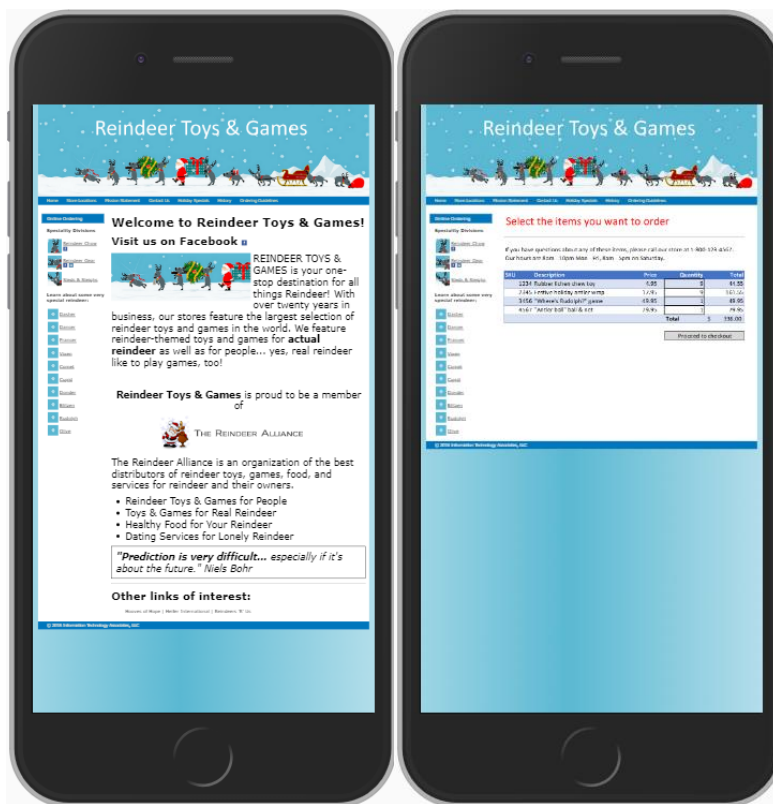


Figure 9: On a mobile phone, the original website is too small to be useful.

The <viewport> meta tag, which goes in the <head> element of the HTML document, tells the browser not to do this.

```
<head>
  <title>Reindeer Toys & Games</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  ...
</head>
```

⁵ This is sometimes referred to as zooming, but that terminology always seemed backwards to me because in this case it means zooming *out* rather than zooming *in*.

The *initial-scale=1.0* part is often omitted because it's the default.

Adding the `<viewport>` tag is a necessary step toward responsive design, but by itself it is not enough to get the desired results in all cases. In fact, when used by itself with no other changes it can actually make matters worse.

Figure 10 shows the Reindeer Toys & Games online ordering page on a simulated iPhone 6 in portrait and landscape modes with the `<viewport>` meta tag added but no other changes. The page is no longer zoomed out and rendered as a miniature, but most or all of the relevant content has overflowed off the edges of the viewport and isn't even visible.

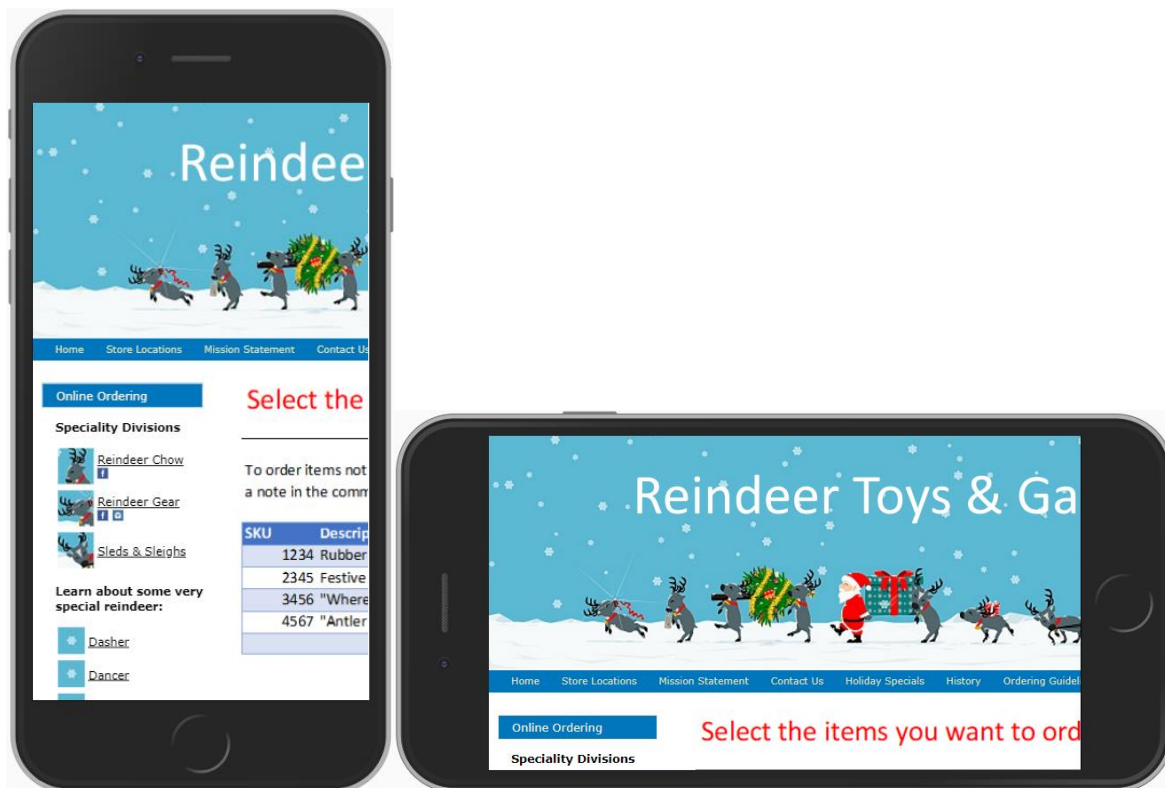


Figure 10: The online ordering page as it appears on an iPhone 6, with the `<viewport>` tag added but no other changes.

Clearly, more work is required to make this site usable on mobile devices!

Revision 2 - Using @media queries for responsive design

The second set of changes begins to make the website responsive to mobile devices. This is accomplished by using CSS media queries to detect the screen size of the visitor's device and then to adjust the page content accordingly.

The first task is to decide on a minimum screen width for rendering the full web page. Mobile devices come in a variety of sizes, and the same device has different dimensions in landscape mode than it does in portrait mode. **Figure 11** lists the viewport sizes for some common mobile devices.

Device	Portrait Mode	Landscape Mode
iPhone 5/SE	320 x 568	568 x 320
Galaxy S5	360 x 640	640 x 360
iPhone 6/7/8	375 x 667	667 x 375
iPhone X	375 x 812	812 x 375
Pixel 2	411 x 731	731 x 411
Pixel 2 XL	411 x 823	823 x 411
iPhone 6/7/8 Plus	414 x 736	736 x 414
iPad	768 x 1024	1024 x 768
iPad Pro	1024 x 1366	1366 x 1024

Figure 11: Viewport sizes for common mobile devices, in pixels.

The original website uses a fixed-width design of 950 pixels, meaning it renders as intended on any device with a screen with a width of 950px or greater. Figure 11 reveals that the available width on an iPad in landscape mode and on an iPad Pro in either landscape or portrait mode is greater than 950, so the page will render correctly on those devices with no changes. The next lower value is an iPad in portrait mode, which has a width of 768, and the available widths go down from there as the devices get smaller. 950 pixels therefore seems to be a good cutoff point — devices with a screen width of 950 pixels or greater get the full page, while anything smaller triggers the responsive adjustments.

It's not necessary to define only one cutoff point. Mid-sized devices, such as an iPad in portrait mode and larger mobile phones, have more available width than smaller mobile phones. As the web designer, you can choose the number and size of however many cutoff points you want. The examples in this session, however, define only the one breakpoint at 950 pixels.

After defining the cutoff points for triggering responsive content, the next step is to determine what adjustments to make if the screen size is below the minimum. This can include rearranging the layout, hiding certain content, reducing image size, replacing a traditional horizontal menu with a drop-down menu, etc.

The full-size banner image designed for the original website isn't going to work well on a mobile device because, with a height of 263px, it would occupy a significant portion of the device's viewport. For this reason, a new banner image was created with a height of only 48px for use on mobile devices. In the HTML document, this image is wrapped in a div named #mobile_logo so the CSS can reference it.

```
<div id="mobile_logo" style="text-align: center">  
    
</div>
```

A pair of media queries in the CSS stylesheet are employed to detect and respond to screen sizes on either side of the 950px breakpoint. The first one, shown in **Listing 19**, hides everything except the content in the #main div and applies some other styles to make the content look good in a smaller viewport.

Listing 19: This media query tells the browser to hide everything except the main content area (#main) when the viewport is 949px wide or less.

```
@media screen and (max-width: 949px) {
  #logo, #menu, #sidebar, #slideshow, #footer {
    display: none;
  }
  #mobile_logo {
    display: block;
  }
  #container, #body2col, #main {
    width: auto;
    float: none;
  }
  #body2col {
    background-image: none;
  }
  #main {
    padding: 20px;
  }
}
```

We don't want the mobile logo to show when the page is not in mobile device mode, so the second media query hides the #mobile_logo div when the viewport is 950px or wider.

Listing 20: The companion piece of CSS tells the browser to hide the #mobile-logo div when the width of the viewport is 950px or greater.

```
@media screen and (min-width: 950px) {
  #mobile_logo {
    display: none; }
}
```

The effect is that the page renders in desktop browser mode when the viewport is 950px or wider, but switches to mobile device mode when the viewport is less than 950px. **Figure 12** shows how the home page appears on screens less than 950 pixels wide (but still wider than an iPhone).



Figure 12: After the changes in revision 2, the site is responsive to viewports less than 950px wide.

So, where are we at this point? The website has come a long way toward being fully responsive, but a couple of things remain to be taken care of. The main problem is that, although the page looks nice on a small screen, there's no menu! Not very useful, is it? On to revision 3.

Revision 3 - creating a navigation menu for mobile devices

Traditional horizontal navigation menus fail on smaller viewports for two reasons: they're usually too wide to fit on the screen, and even if they did fit the font would typically be uncomfortably small. One solution is to replace the traditional horizontal menu with a menu designed specifically for mobile devices, and to have that menu appear in place of the standard one when the screen width is below a certain minimum.

On the desktop, the Reindeer Toys & Games website features a horizontal menu that sits below the banner image and matches its width. This looks good on the desktop but is too wide for mobile devices. The responsive design for mobile devices therefore needs a new menu, one that works on any viewport below the 950px threshold. The typical approach is to create a "hamburger" menu that is always visible in its collapsed state and drops down to reveal the menu choices in response to a touch.

Figure 13 shows the new mobile menu in its collapsed state in a viewport just below the 950px breakpoint. Note the "hamburger" icon on the right edge of the menu.

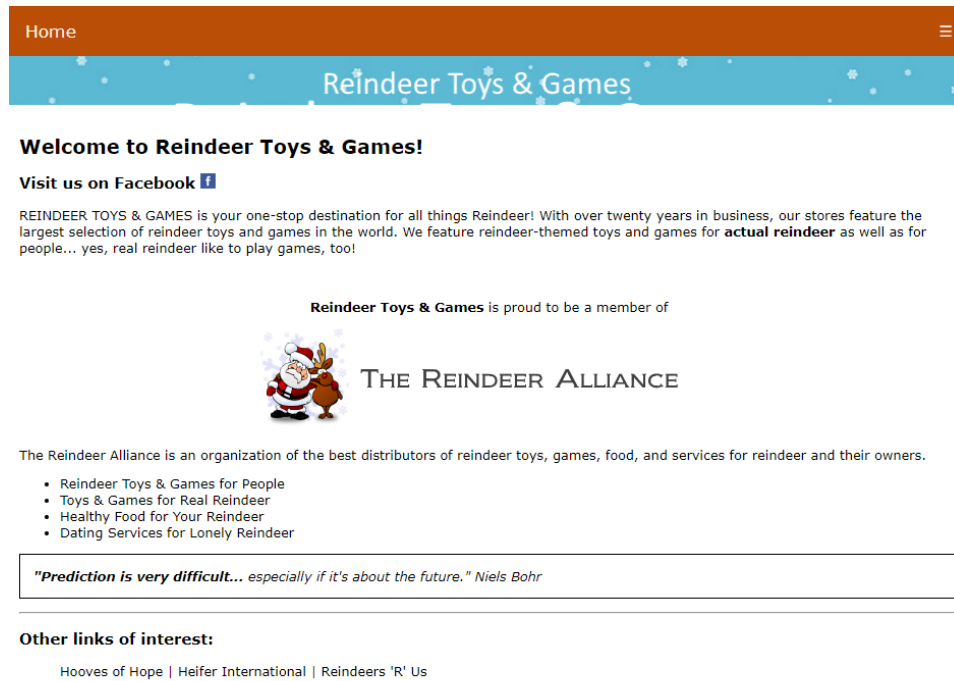


Figure 13: The mobile menu appears in collapsed mode at the top of the page when the viewport is less than 950px wide.

These changes are implemented with a new stylesheet, some additional code in the HTML files, and a little piece of JavaScript to activate the pop-up menu in response to a touch on the hamburger icon. The code for this responsive menu is adapted from the excellent example at www.w3schools.com/howto/howto_js_topnav_responsive.asp.

The responsive menu

Listing 21 shows the HTML for the responsive menu, which goes in the `<body>` of the HTML document in the form of an unordered list wrapped in a div with an ID of `topnav` (`#topnav` in the CSS stylesheet). The `` tag itself is wrapped in a div with an ID of `MobileTopNav` and a class named `topnav` (`.topnav` in the CSS stylesheet). The mobile stylesheet can then reference each of these as necessary to control their appearance and behavior.

One decision to be made is how many, and which, choices to show on the mobile menu. With too many items the menu would be taller than the viewport on smaller phones. If this is not acceptable, you may decide it's not necessary for all content to be accessible on a mobile device. Reindeer Toys & Games decided to limit the number of choices to what it considers the most important for users of mobile devices in order that the entire menu fits on smaller devices.

Note that the last item in the list in Listing 21 is different than the others — it contains the hamburger icon and is linked to a little piece of JavaScript whose function it is to expand (drop down) the rest of the menu when touched, or collapse it if it's already expanded. The

hamburger icon itself does not require an image — it's the HTML entity `☰`. The list item for the hamburger icon also gets its own class name *icon*.

Listing 21: The `#topnav` div defines the mobile menu.

```
<nav id="topnav">
  <ul class="topnav" id="MobileTopNav">
    <li><a class="active" href="index.htm">Home</a></li>
    <li><a href="orderitems.htm">Online Ordering</a></li>
    <li><a href="index.htm">Store Locations</a></li>
    <li><a href="index.htm">Mission Statement</a></li>
    <li><a href="index.htm">Contact Us</a></li>
    <li><a href="index.htm">Holiday Specials</a></li>
    <li><a href="index.htm">History</a></li>
    <li><a href="index.htm">Ordering Guidelines</a></li>
    <li><a href="index.htm">Reinderr Chow</a></li>
    <li><a href="index.htm">Reindeer Gear</a></li>
    <li><a href="index.htm">Sleds & Sleighs</a></li>
    <li class="icon">
      <a id="hamburger" href="javascript:void(0);" style="font-size:15px;"
        onclick="hamburgerMenu()">&#9776;</a>
    </li>
  </ul>
</nav>
```

The CSS for the responsive portion of the design resides in a new stylesheet named *mobile_topnav.css*. In addition to doing some things with list style and color, it contains the media queries to activate the responsive design on devices whose screens are 949 pixels wide or less.

The first part of the *mobile_topnav.css* stylesheet, shown in **Listing 22**, hides all but the top line of the menu when the menu is collapsed and floats the hamburger icon to the right.

Listing 22: This code applies the *display: none* style to all but the first child element in the unordered list.

```
@media screen and (max-width:949px) {
  ul.topnav li:not(:first-child) {display: none;}
  ul.topnav li.icon {
    float: right;
    display: inline-block;
  }
}
```

The other part, shown in **Listing 23**, handles what happens when *responsive* is appended to the *topnav* class, which is accomplished by the JavaScript.

Listing 23: These rules control the behavior of the mobile menu when the viewport is less than 950px wide.

```
@media screen and (max-width:949px) {
  ul.topnav.responsive {position: relative;}
  ul.topnav.responsive li.icon {
    position: absolute;
    right: 0;
  }
}
```

```
    top: 0;
  }
  ul.topnav.responsive li {
    float: none;
    display: inline;
  }
  ul.topnav.responsive li a {
    display: block;
    text-align: left;
  }
}
```

The final piece is the JavaScript, which goes at the bottom of the HTML document. It toggles the menu between expanded and collapsed by appending or removing the *responsive* class to the *MobileTopNav* ID. It also changes the hamburger menu icon to an “X” when the menu is expanded, indicating that a touch will collapse the menu, and restores it to the hamburger icon when the menu is collapsed.

Listing 24: This JavaScript toggles the behavior of the mobile menu between its collapsed state and its expanded state.

```
<script>
function hamburgerMenu() {
  var x = document.getElementById("MobileTopNav");
  var y = document.getElementById("hamburger");
  if (x.className === "topnav") {
    x.className += " responsive";
    y.innerHTML = "X"
  } else {
    x.className = "topnav";
    y.innerHTML = "&#9776;";
  }
}
</script>
```

Together, these relatively simple modifications and additions make the mobile menu to function in a way that fits on smaller screens and works well with a touch interface. **Figure 14** shows the home page on a simulated iPhone 6 with the menu collapsed and then expanded.

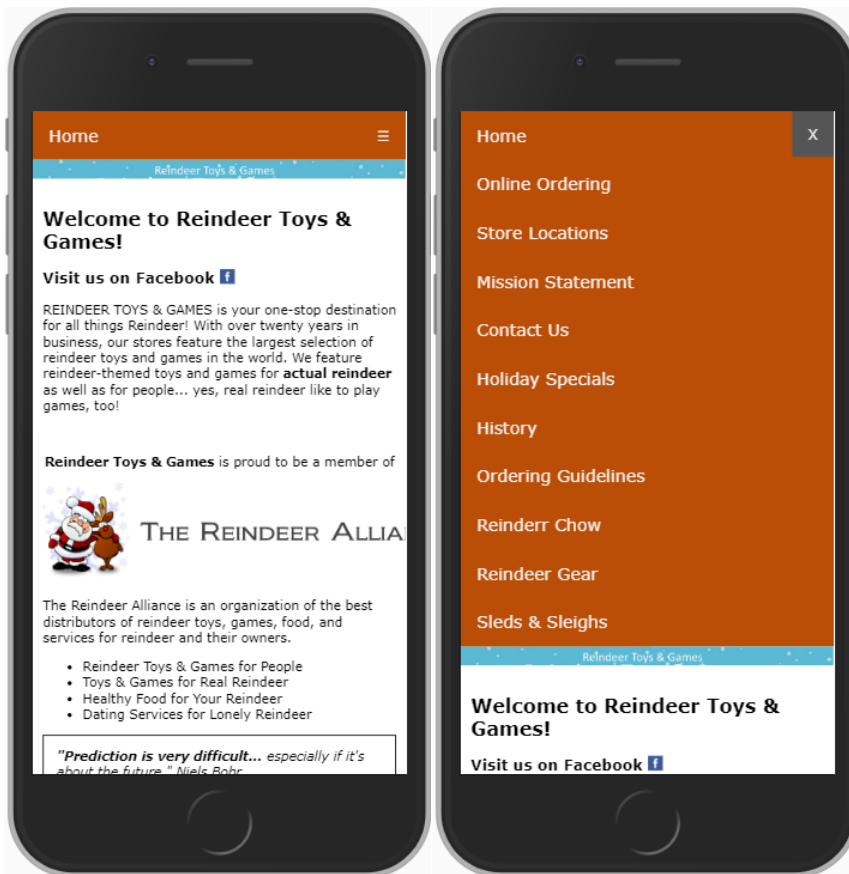


Figure 14: Touching the hamburger icon expands the menu; touching the X collapses it.

Revision 4 - adjusting images for viewport size

Although the page is now responsive to the size of the viewer's device, there is still a problem with the Reindeer Alliance image on smaller screens. The HTML for the image, which is still as it was in the original design, specifies the image's width and height (450px by 100px), as shown in **Listing 25**.

Listing 25: The original HTML defines the Reindeer Alliance image with a fixed width and height.

```
<p style="text-align: center;">
  <a href="index.htm">
    
  </a>
</p>
```

This is fine if the space available to display the image in the viewport is at least 450px wide, but it causes a problem on viewports smaller than that. Depending on the device and the browser, the image is either truncated (as shown in Figure 14) or the image fits but the rest of the page is narrowed — a weird result shown in **Figure 15**.

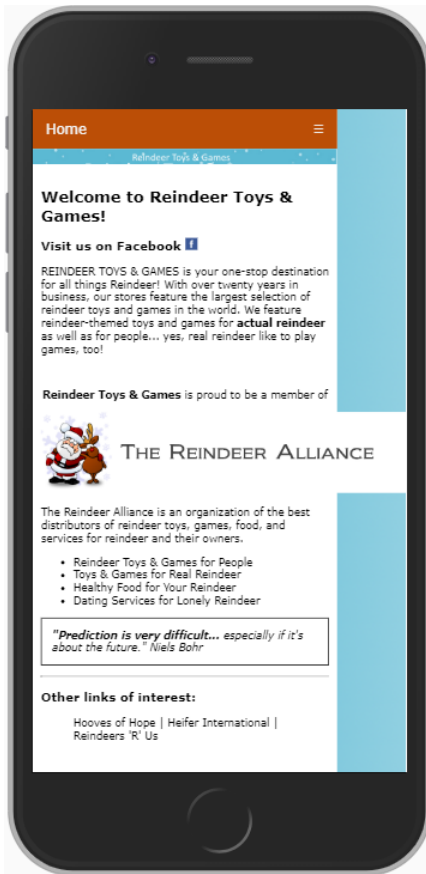


Figure 15: This weird result can happen if the `` tag defines a width for the image that's greater than the width of the viewport.

There are several ways to resolve this issue by making the image responsive along with the rest of the page.

Solution 1

The first solution is to modify the `` tag and use inline CSS to specify a width of 100%, as shown in **Listing 26**. This is the easiest change to make, but not the best solution.

Listing 26: Defining the image's width as 100% means it grows or shrinks to fill the entire width of its container regardless of whether it's larger or smaller than the actual width of the image.

```
<p style="text-align: center;">
  <a href="index.htm">
    
  </a>
</p>
```

The original image is 450px-wide centered in a 730px-wide container, which is a ratio of about 60% with 20% white space on either side. After applying the `width="100%"` style, the image stretches or shrinks to fill the width of the container regardless of its size. This can have undesirable consequences, because on a desktop browser with a large viewport the image may appear artificially large.

Solution 2

A better solution is to use an inline CSS style that enables the image to fill 100% of the container, as in solution 1 above, but never to be more than the actual width of the image.

On desktop browsers this looks the same as the original design, with white space to either side of the image if the container is wider than the image. As the viewport gets progressively smaller, the white space shrinks and finally disappears entirely at 450px. Below that, the image itself shrinks but continues to fill 100% of the width of the container.

This is also an easy solution to implement and, in my experience, the results look good on all devices if the image quality is such that it continues to look good as it shrinks. For those reasons, this is my current favorite solution. **Listing 27** shows how it's done, and **Figure 16** shows what it looks like on a simulated iPhone6.

Listing 27: Styling the image with a maximum width of 100% means it's never wider than its actual width but can shrink to fill its container; *height: auto* keeps the ratio of width to height constant as the image shrinks.

```
<p style="text-align: center;">
  <a href="index.htm">
    
  </a>
</p>
```

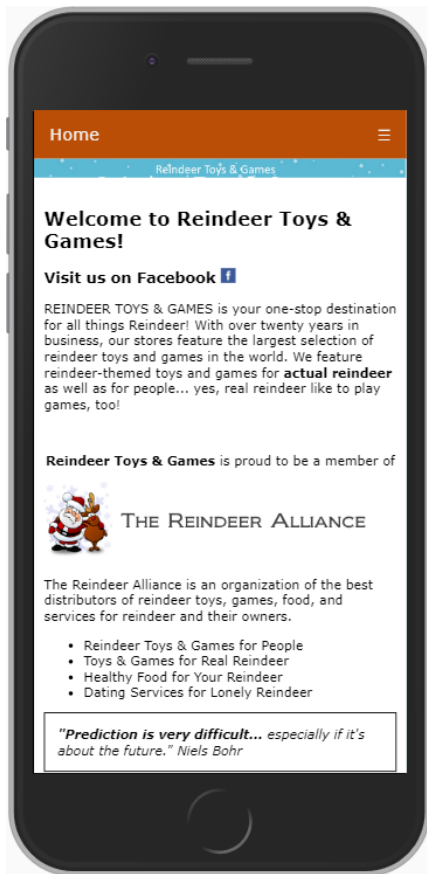


Figure 16: With this solution, the image shrinks to fill the entire width of the container when the container is narrower than the image, but it never grows larger than the image's actual size.

Solution 3

A more involved solution is to create different size versions of the image — or even different images altogether — for different size viewports, and then to use the HTML5 `<picture>` tag to control which one gets displayed. The idea behind this approach is to render an image whose width, in proportion to the width of its container on a mobile device, is roughly the same as the width of the full-size image is in proportion to the width of a desktop browser. This solution involves more work because you need to create multiple images, but in return it gives you more control over what the viewer sees on various size devices.

Listing 28 shows how this is accomplished, using arbitrarily different size images and viewport breakpoints. On viewports with a width of 700px or greater, the original 450x100 image is displayed. The 350x78 image is displayed on viewports below 700px down to 600px. Below 600px and down to 400px the 250x56 image is displayed, while any device with a viewport smaller than 400px gets the 200x44 version of the image.

Listing 28: The HTML5 `<picture>` tag can be used to select different images depending on the width of the viewport.

```
<p style="text-align: center;">
```

```
<a href="index.htm">
  <picture>
    <source media="(min-width: 700px)"
      srcset="images/ReindeerAllianceLogo_450x100.png">
    <source media="(min-width: 600px)"
      srcset="images/ReindeerAllianceLogo_350x78.png">
    <source media="(min-width: 400px)"
      srcset="images/ReindeerAllianceLogo_250x56.png">
    
  </picture>
</a>
</p>
```

Figure 17 shows how this looks on a simulated iPhone 6, which has a viewport that's 375px wide. This is less than 400px, so the 200x44px image is displayed. With this solution, the percentage of white space to the left and right of the image remains approximately the same as when the page is viewed in a full-width desktop browser.

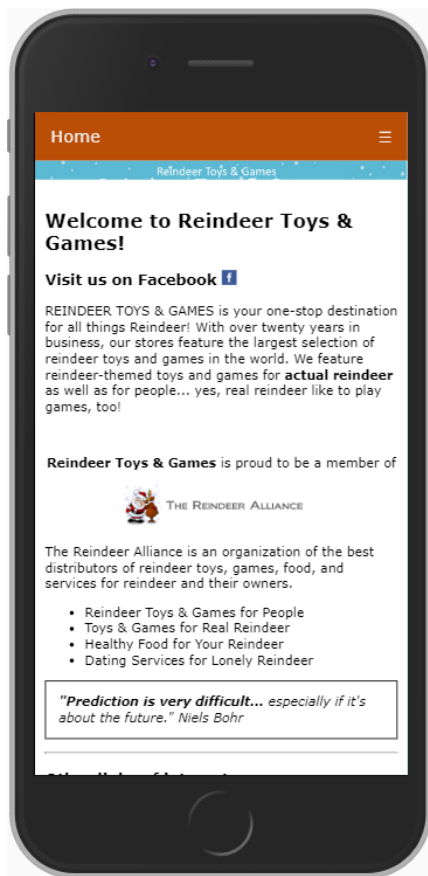


Figure 17: The image is rendered on a mobile device with roughly the same proportional width as on a desktop browser.

Revision 5 – fixing the Safari landscape zoom bug

There is a bug (or perhaps it's a feature) in the mobile Safari browser that zooms the font when the device is rotated into landscape mode. You won't be able to see this in the mobile browser "emulators" in Chrome or Firefox, only in Safari on an actual iOS device.

Figure 18 is a screenshot from Safari on an actual iPhone 6S showing the home page in portrait mode, where it looks as intended for a viewport of that size (375px wide).



Figure 18: The page looks as intended on an iPhone in portrait mode.

When viewed in landscape mode (**Figure 19**), the same page on the same phone exhibits the font zoom bug. The difference in font size is clear in comparison to how the page looks in Figure 18.



Welcome to Reindeer Toys & Games!

Visit us on Facebook

REINDEER TOYS & GAMES is your one-stop destination for all things Reindeer! With over twenty years in business, our stores feature the largest selection of reindeer toys and games in the world. We feature reindeer-themed toys and

Figure 19: The font is enlarged when the phone is rotated into landscape mode.

Some people I've talked to consider this a feature, as they're accustomed to being able to rotate their iPhone into landscape mode to make the font larger if it's difficult to read in portrait mode. Other people consider this a bug. If you're one of the latter and want to fix it, one way is to apply the following WebKit CSS rule to the *html* selector (i.e., it applies to everything on the page) to override the Safari browser engine's default.

```
html {  
  /* Prevent font size increase in Safari when changing to landscape mode. */  
  -webkit-text-size-adjust: 100%;  
}
```

With this rule applied the font looks more reasonable in landscape mode, as shown in **Figure 20**. Compare to Figure 19 to see the difference.



Welcome to Reindeer Toys & Games!

Visit us on Facebook 

REINDEER TOYS & GAMES is your one-stop destination for all things Reindeer! With over twenty years in business, our stores feature the largest selection of reindeer toys and games in the world. We feature reindeer-themed toys and games for **actual reindeer** as well as for people... yes, real reindeer like to play games, too!

~~Reindeer Toys & Games is proud to be a member of~~

Figure 20: After the WebKit rule is applied, the font is no longer enlarged when the phone is rotated into landscape mode.

Summarizing the revisions

The modifications in the previous steps have transformed an original website that wasn't at all responsive into one that's fully responsive to different viewport sizes, all without changing the basic structure of the layout. The original appearance of the site remains the same when viewed on a desktop browser, but when viewed on a tablet or a mobile phone the content is now adjusted to work well on those devices. The result is happy customers and a happy company whose website is now accessible to a much larger audience. The final code for this example is included in the session materials.

Flexbox

Flexbox is the short name for the CSS3 Flexible Box layout module. It's another way to tell the browser how to arrange the structural elements – i.e., the “boxes” – on a web page. You can think of flexbox as an alternative to float-based layouts.

To use flexbox, you define a container (usually a `div`) for the layout and add items (also usually `divs`) within it. The HTML elements that are direct children of a flexbox container are the *flex items* for that container. You then set flexbox CSS properties and values to control how the flex items are arranged within their container.

It's called “flexible” because the items within the flexbox container can be made to grow to fill the available space or shrink to avoid overflowing the boundaries of their container. Flex items can be arranged either horizontally in rows or vertically in columns, and can be set to wrap or not to wrap when the container gets smaller.

Flexbox is not new – its earliest implementation was in 2009. By now it's implemented in all modern browsers, so you can use it with confidence. If you want to use flexbox and need

to support older browsers, there are browser engine-specific properties you can set such as `-webkit-flex` for Safari and `-ms-flex` for IE.

Superficially, flexbox is very simple. What follows is a quick introduction to flexbox basics, hopefully enough to get you started.

Basic flexbox

At its most basic level, all you need to implement flexbox is a bit of HTML and one CSS rule. The code in **Listing 29** defines an HTML div with three other divs inside it, along with the CSS rule to make it a flexbox container.

Listing 29: The `display: flex` rule makes the `container` class a flexbox container.

```
<style>
  .container {
    display: flex;
  }
</style>

<div class="container">
  <div>First flex item</div>
  <div>Second flex item</div>
  <div>Third flex item</div>
</div>
```

Figure 21 shows what this looks like in a browser. It isn't very impressive at first. By default, there is no space between flexbox items so the content is all run together.

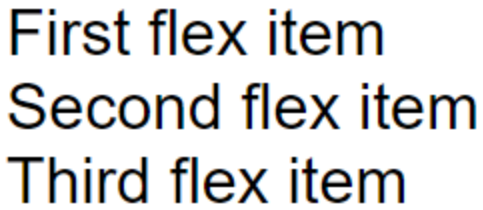


First flex itemSecond flex itemThird flex item

Figure 21: The code from Listing 29 looks like this in the browser.

Flexbox is already doing things behind the scenes, though. Note that the items in Figure 21 are arranged horizontally in a single row. That's because there is a flexbox property named `flex-direction` whose initial (default) value is `row`, so it doesn't need to be specified explicitly. The other values for `flex-direction` are `row-reverse`, `column`, and `column-reverse`.

HTML divs are block items. Without the `display: flex` rule, and assuming you're not using `display: inline`, each div would be displayed on a separate line, as shown in **Figure 22**.



First flex item
Second flex item
Third flex item

Figure 22: Without the `display: flex` rule, the divs within the container are displayed in default block mode.

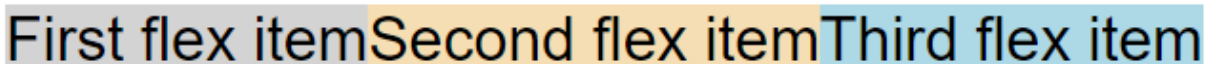
It's easy enough to see where the boundaries between the flex items falls in Figure 21 because the text gives it away, but it will be more difficult when there is white space surrounding or between the items. To make it easier to see the boundaries, we can assign class names to the flex items in the HTML (**Listing 30**) and then add CSS rules to apply different background shading to each item (**Listing 31**). With the background shading in place it's much easier to see the boundaries between the items, as shown in **Figure 23**.

Listing 30: The divs that define the flex items are each given a unique class name so the CSS can reference them.

```
<div class="container">  
  <div class="firstitem">First flex item</div>  
  <div class="seconditem">Second flex item</div>  
  <div class="thirditem">Third flex item</div>  
</div>
```

Listing 31: These CSS rules use the class names to add background shading and space between the items.

```
.container {  
  display: flex;  
}  
.firstitem {  
  background-color: lightgrey;  
}  
.seconditem {  
  background-color: wheat;  
}  
.thirditem {  
  background-color: lightblue;  
}
```



First flex itemSecond flex itemThird flex item

Figure 23: Background shading makes it easier to see the boundaries between the flex items.

By default, the width of each flex item is determined by the width of its content. Although it isn't easy to tell from Figure 23, the width of the second item is greater than the width of the other two because the text in that item is longer. Laying a ruler along the top of the

image makes it easy to see that the first item has a width of approximately 185px, while the second has a width of about 240px (see **Figure 24**).

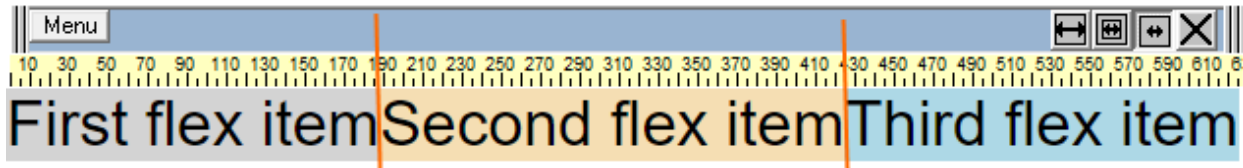


Figure 24: The second flex item is wider than the other two because its content (the text) is longer.

Unless we do something about it, the flex items are bunched together starting at the left edge of their container regardless of the width of that container. In this example there are no constraints on the width of the container, so it occupies the full width of the browser window. **Figure 25** shows how the flex items look in a much wider browser window.

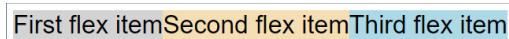


Figure 25: The items are bunched together starting at the left edge of the flex container.

In a real design, you'd probably want some spacing between the flex items. One way to do that is to apply the *justify-content* property to the container. One of the values for the *justify-content* property is *space-between*. This gives the result shown in **Figure 26**. Note that the width of each item is still determined by its content.

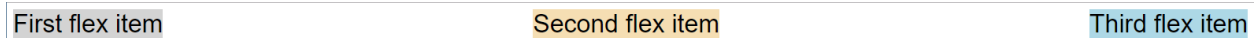


Figure 26: Applying the *justify-content: space-between* rule distributes the available white space between the items.

Other values for the *justify-content* property include *space-evenly* and *space-around*, which yield different results.

Another way to add spacing between the flex items is to add a margin instead of using the *justify-content* property. This gives you control over the amount of space between the items, but the amount of space remains constant regardless of the width of the container. Adding a 20px right margin to the first and second item gives the result in **Figure 27**.

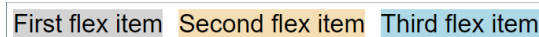


Figure 27: Adding a 20px right margin to the first and second item puts that amount of some space between the items.

Suppose we want the width of each item to grow to fill the available width of the container. The *flex* property gives us a way to do that. Applying the *flex: 1* rule to the container div yields the result shown in **Figure 28**.

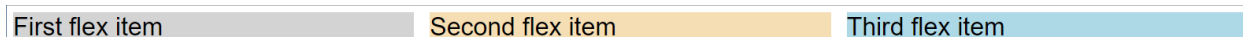


Figure 28: Applying the *flex: 1* rule to the container div tells the browser to grow the items to fill the available space.

The HTML code has remained unchanged though all of this. **Listing 32** shows the current state of the CSS.

Listing 32: These CSS rules add background shading and space between the items.

```
.container {
  display: flex;
}
.container div {
  flex: 1;
}
.firstitem {
  background-color: lightgrey;
  margin-right: 20px;
}
.seconditem {
  background-color: wheat;
  margin-right: 20px;
}
.thirditem {
  background-color: lightblue;
}
```

When the viewport is narrowed to the point where it is less than the width required for all the text in all three items to fit on a single line within the box, the text wraps to a second line. However, all three items are still in one row. **Figure 29** shows how this might look on a mobile phone.

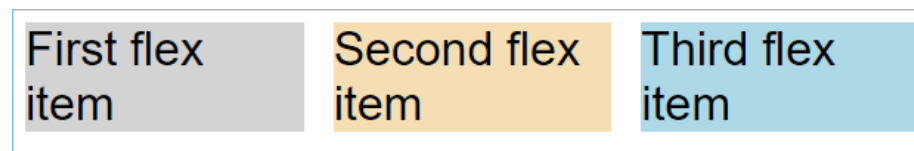


Figure 29: The same code looks like this when the viewport is too narrow to show all three items on a single line.

So far, the content (i.e., the text) in all three items has been approximately the same length. Another useful feature of flexbox is how it handles situations where the content is of significantly unequal lengths. In **Listing 33**, some additional text has been added to the second item.

Listing 33: The text in the second flex item is now much longer than the other two.

```
<div class="container">
  <div class="firstitem">First flex item</div>
  <div class="seconditem">Second flex item<br>"Now is the time for all good citizens
to come to the aid of their country."</div>
  <div class="thirditem">Third flex item</div>
</div>
```

Figure 30 shows how flexbox renders this content. Note the all three flex items have the same height and the shading goes all the way to the bottom in each of them – a desirable result that can be difficult to achieve in float-based layouts.

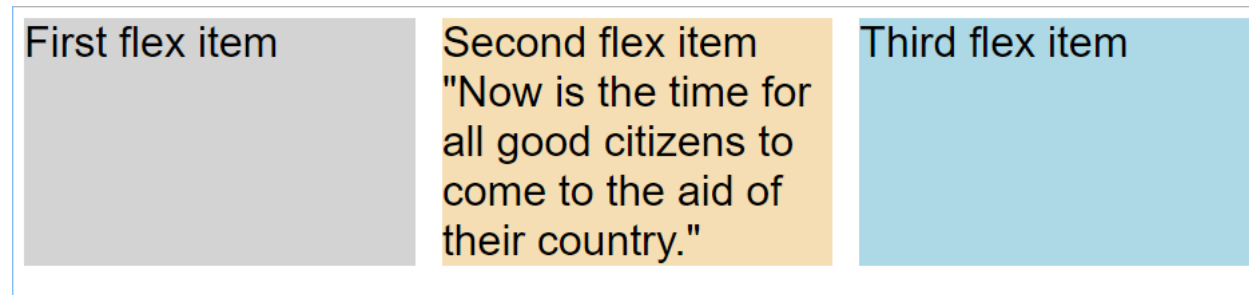


Figure 30: Flexbox maintains the same height for each item even when the size of the content differs.

Applying flexbox to Reindeer Toys & Games

Now that you have a sense of what flexbox is all about and what it can do, let's see how it might be applied to the Reindeer Toys & Games website. To do this, we'll start over again with the original website shown in Figure 4, the one the was not responsive at all.

Except for a couple of minor changes to the HTML, applying flexbox to this design is accomplished entirely with modifications to the CSS stylesheet. Media queries are not needed because flexbox is inherently responsive to the size of the viewport. As a result, the flexbox version of the Reindeer Toys & Games website is responsive to mobile devices although in a different way than the examples in the previous section.

Recall that the HTML for the original website defines the basic structure using nested divs, a condensed version of which is shown in **Listing 34**.

Listing 34: The logo, menu, body2col, and footer divs are written sequentially at the same level in the HTML, while the sidebar and main divs are nested within body2col.

```
<div id="container">
  <div id="logo">...</div>
  <div id="menu">...</div>
  <div id="body2col">...</div>
    <div id="sidebar">...</div>
    <div id="main">...</div>
  <div id="footer">...</div>
</div> <!-- end of container div -->
```

The basic structure of the HTML can remain unchanged when we apply flexbox styles to it, but we do need to think about how the structural elements are put together and how we want them to behave under flexbox. Looking back at Figure 4 and Figure 5, you can see how the structure illustrated in Listing 34 determines the appearance of the web page: because divs are block elements, the four children (first-level descendants) of the *container* div are displayed one on top of the other in a single column, while CSS floats are employed to display the two divs within the *body2col* container side by side in a row. This suggests the basic approach to flexbox for this website.

The first modification is to remove the fixed width from the CSS for the *container* div and replace it with properties indicating that we want it to be a flexbox container with a flow direction of *column*, as shown in **Listing 35**. The *flex-flow* property is a shorthand way to specify both *flex-direction* and *flex-wrap* in one step. The initial (default) for *flex-wrap* is *nowrap*, so it does not need to be explicitly specified – it’s included in Listing 35 only for the sake of completeness. The *flex-basis* property is the default size of an element; in this case, it’s the flexbox equivalent of *width*.

Listing 35: The *display: flex* property is used to tell the browser that the *container* div is a flexbox container.

```
#container {
  margin: 0px auto;
  text-align: left;
  display: flex;
  flex-flow: column nowrap;
  flex-basis: 950px;
}
```

The next step is to remove the floats and set the flexbox properties so the items within the *body2col* container are arranged in a row. We also want the main content to wrap under the sidebar when the viewport is less than 950px wide.

```
#body2col{
  display: flex;
  flex-flow: row wrap;
}
```

To maintain the proportion of the width of the sidebar to the width of the main content, they are assigned a *flex-basis* of 200px and 700px respectively.

```
#sidebar {
  padding: 10px 10px 10px 25px;
  margin-top: 15px;
  flex-basis: 200px;
}

#main {
  padding: 10px;
  flex-basis: 700px;
}
```

The items on the menu under the logo should also wrap so they remain visible when the viewport is less than 950px wide.

```
#menu {
  height: auto;
}
#menu ul {
  margin: 0px;
  padding: 0px;
  list-style: none;
  display: flex;
```

```
flex-flow: row wrap;  
}
```

That's about it. The only changes we need to make to the HTML in its original state are to (a) include the `<viewport>` meta tag and (b) make the logo and the Reindeer Alliance logo shrink when the viewport is small, which is done by replacing their fixed width and height with `style="max-width: 100%; height: auto;"` as in previous revisions.

Although these changes may not be a complete solution, they are enough to illustrate how flexbox can be used to make this site responsive to smaller viewports. **Figure 31** shows the how the home page looks on a simulated iPhone6. Note that the banner image and menu have shrunk to fit the width of the viewport and the items on the menu have wrapped so all of them remain visible. The main content has wrapped below the sidebar, so scrolling is required to see it. If the other way around is preferred, the `order` property can be used to make the main content appears first, above the sidebar.

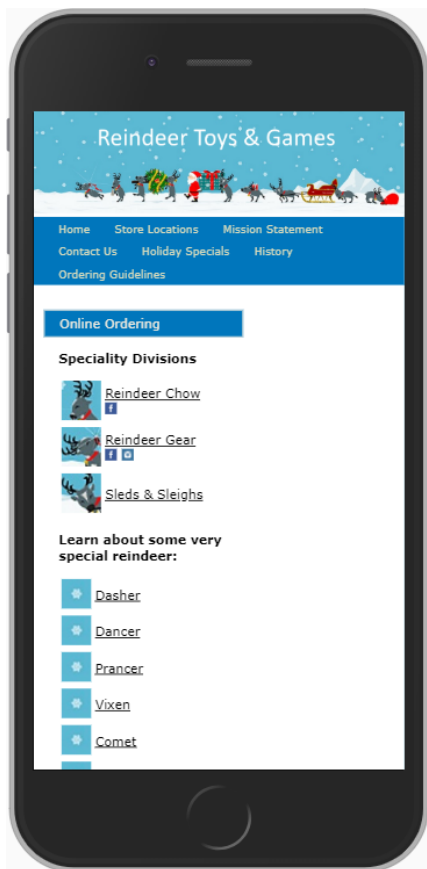


Figure 31: The home page on a simulated iPhone6 using flexbox.

All the code for this example is included in the session materials.

Where to go from here

Like anything else, flexbox has its own quirks and complexities. If you're interested in information beyond what's presented here, the references section at the end of this paper

has links to several resources I've found helpful. In addition, your own search will turn up many sites with flexbox documentation and examples. Some of these are interactive so you can play with the code from your browser and see what happens — a great way to learn.

Tools for creating and testing responsive designs

Among the many challenges of developing responsive websites is the need to test your design on multiple browsers and multiple devices. Most browsers these days provide developer tools that let you simulate mobile devices. This is tremendously useful, but as far as I know these tools still use the browser's own rendering engine so the results sometimes differ from what you would see on the actual device.

Ideally, you'd use developer tools for the initial development work and then test on as many actual devices as possible to fine-tune your design. Following are some of the tools you may find useful.

Chrome browser developer tools

- F12 to activate
- includes a built-in “emulator”, use Ctrl+Shift+M while in dev tools or click the icon

Firefox browser developer tools

- F12 to activate
- includes a built-in “emulator”, use Ctrl+Shift+M, no need to be in developer tools first

BrowserStack Mobile Browser Emulator (<https://www.browserstack.com/mobile-browser-emulator>)

- not free, but advertises “instant, browser-based access to the latest mobile devices”

WindowWatch

- freeware from Brew City Software
- an oldie but a goodie
- v1.4.11 works on Windows 10, but you may have to search for a download site
- quickly set the height, width, and position of any window on your machine
- includes on-screen rulers you can turn on or off (as used in Figure 24)

Beyond Compare (www.scootersoftware.com)

- one of the best all-time indispensable utilities for many reasons
- file compare is great for inspecting the differences between versions of your design

Replace Studio Pro (www.funduc.com)

- the successor to Search and Replace
- great for locating, and optionally replacing, all occurrences of a string in a group of files

Visual Studio Code (<https://code.visualstudio.com/>)

- free, open-source code editor from Microsoft
- has language-specific features built in, including HTML and CSS
- the more I use it, the more I like it

CSS Validator – <http://jigsaw.w3.org/css-validator/>

HMTL Validator – <https://validator.w3.org/>

Summary

Web development has become a complex process potentially involving many different tools and skill sets, and the need to enable websites to respond to mobile devices has made it even more so. However, if you can master a few basic concepts like the ones presented in this paper, you can learn how to develop responsive websites without having to become a subject matter expert in the field. Making your website responsive to mobile devices increases its reach to a much wider potential audience and helps ensure that it remains relevant in today's mobile-centric world.

References

Books

CSS – The Missing Manual, 4th edition, by David Sawyer McFarland

CSS Pocket Reference, 5th edition, by Eric A. Meyer

HTML5 Pocket Reference, 5th edition, by Jennifer Niederst Robbins

Southwest Fox Conference Papers

Creating Mobile Applications with Angular 2 and Web Connection, Rick Strahl, Southwest Fox 2016

Creating Beautiful Web Sites Easily Using Bootstrap, Doug Hennig, Southwest Fox 2016

Building Mobile Web Applications with Angular JS, Bootstrap and Web Connection, Rick Strahl, Southwest Fox 2015

Magazine Articles

Flexing Your HTML Layout Muscles with Flexbox, Rick Strahl, CODE Magazine, Mar-Apr 2016

Online Resources

HTML5

HTML5 Tutorial

<https://www.w3schools.com/html/default.asp>

HTML5 Style Guide and Coding Conventions

https://www.w3schools.com/html/html5_syntax.asp

HTML Responsive Web Design

https://www.w3schools.com/html/html_responsive.asp

HTML Styles – CSS

https://www.w3schools.com/html/html_css.asp

HTML JavaScript

https://www.w3schools.com/html/html_scripts.asp

CSS and responsive design

CSS Indexes – a listing of every term defined by CSS

<https://drafts.csswg.org/indexes/>

CSS @media Rule

https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

CSS Media Queries

https://www.w3schools.com/css/css3_mediaqueries.asp

Responsive Web Design – Introduction

https://www.w3schools.com/css/css_rwd_intro.asp

Responsive Web Design – Media Queries

http://www.w3schools.com/css/css_rwd_mediaqueries.asp

Responsive Web Design – The Viewport

https://www.w3schools.com/css/css_rwd_viewport.asp

How To – Responsive Top Navigation

https://www.w3schools.com/howto/howto_js_topnav_responsive.asp

Convert a Menu to Dropdown for Small Screens

<https://css-tricks.com/convert-menu-to-dropdown/>

Flexbox

CSS Flexbox (Flexible Box)

https://www.w3schools.com/csS/css3_flexbox.asp

A Complete Guide to Flexbox | CSS Tricks

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Basic Concepts of Flexbox

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox

Flexbox Froggy

<https://flexboxfroggy.com>

Flexbox Zombies

<http://mastery.games/courses/139425>

Biography

*Rick Borup is owner and president of Information Technology Associates, LLC, a professional software development, computer services, and information systems consulting firm he founded in 1993. Rick earned BS and MBA degrees from the University of Illinois and spent several years developing software applications for mainframe computers before turning to PC database development tools in the late 1980s. He began working with FoxPro in 1991, and has worked full time in FoxPro and Visual FoxPro since 1993. He is a co-author of the books *Deploying Visual FoxPro Solutions* and *Visual FoxPro Best Practices For The Next Ten Years*. He has published articles in *FoxTalk*, *FoxPro Advisor*, and *FoxRockX* and is a frequent speaker at Visual FoxPro conferences and user groups. Rick is a Microsoft Certified Solution Developer (MCSD) and a Microsoft Certified Professional (MCP) in Visual FoxPro.*

Copyright © 2018 Rick Borup. Windows® is a registered trademark of Microsoft Corporation in the United States and other countries. All other trademarks are the property of their respective owners