

Developing Mobile Websites

Flexible Grids

Lesson 1, Activity 2: The Flexible Grid: A More Responsive Layout Strategy

The Case for Flexible

Responsive Web design is the process by which we craft the markup and stylesheets for a site to adapt to the user's environment. Users viewing our site on a smartphone, a tablet device, a laptop with a small screen resolution, or a desktop with high-resolution monitor all see pages laid out in a format friendly to their particular device, with minimal (if any) need for scrolling and zooming. Responsive design usually means making use of CSS3 media queries and often means CSS3 flexible images.

A first step toward a responsive design is implementation of a flexible, fluid grid layout. Instead of formatting element widths with pixels, we will instead use percentages: what had been a 980-pixel, fixed-width, center-justified design - that is, a page which remains ever at 980 pixels wide, regardless of the context in which we view it - will become a fluid layout, where the page scales to fill a set percentage of the screen, and elements within the design scale bigger and smaller (but remain in proportion) as the size of the device or browser changes.

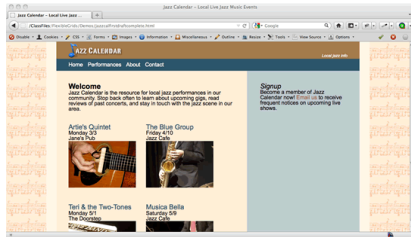
Why do this?

1. The diversity of ways in which users access our sites has gotten even greater in recent years (and will continue to increase) - browsers on video game consoles, televisions, smartphones like the Android or iPhone, and tablets like the iPad now represent a larger and larger chunk of audience share for many sites.
2. Screen sizes are getting both larger and smaller - for most sites, assuming that most users will access our pages on an average screen of 760 or 980 or whatever pixels no longer holds.

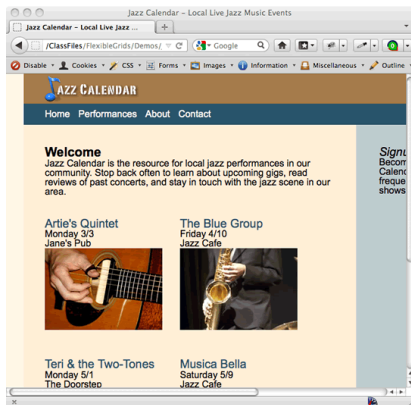
Case Study: The Jazz Calendar Site

Our goal is to develop a site for a client that shares information about local jazz music performance. Aimed at an audience of local jazz enthusiasts, the site will feature upcoming concerts at local clubs, give users a chance to sign up for email alerts, offer ways to interact with others, and - over the course of our work here - be optimized for viewing on mobile devices.

The screen shot below shows the first, nonresponsive, design goal. A music-note background image fills the page; the main content sits in a 960px-wide, fixed-width container. It looks fairly good if our screen and browser are wide enough:



If we resize our browser, so that the width of the browser is, say, 700 pixels, the design doesn't quite work. The fixed width of the shell of the design remains 960 pixels wide; extending to the right of the now-narrower browser and forcing a horizontal scroll:



Here is, below, a guide to the widths of the various top-level elements of the design. The main content shell is 960 pixels wide; margin:0 auto centers the element in the browser window; the inner shell (the #main div) sits centered within the main shell. The main content - the wider, left-side column - floats left, with a total width of 566 pixels (494 pixels of width plus 36 pixels of padding right and left.) The right column, with the "Sign up" content, floats right; it has a total width of 331 pixels (259 + 36 + 36.)

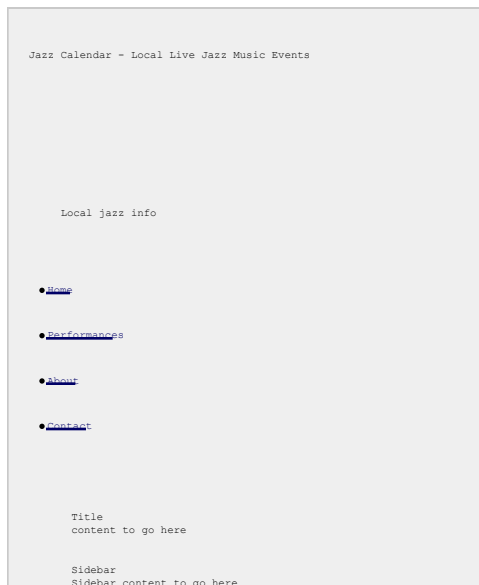


Our first effort renders the shell of the design - the guts of the main content is missing, but we've got the basics done. We use a CSS reset (based on [Eric Meyer's widely-used code](#)) to ensure a consistent baseline of CSS before crafting our own stylesheet. A reset removes inconsistencies among browsers' defaults: as things like line heights, margins, and heading font sizes differ between Firefox, Internet Explorer, and Chrome (and among various versions of each browser), a reset file like the one mentioned enables us to start with a level playing field, allowing us to deliver a more-consistent presentation regardless of the browser (or device) our users might employ to view our site.

We've used good HTML5 markup - the <nav> tag wraps the main navigation element, the <aside> tag holds the right sidebar, etc. We use margin:auto to center elements and float the two main columns left and right.

Code Sample:

[FlexibleGrids/Demos/jazzallfirstdraft.html](#)



Jazz Calendar

123 Fake Street, Sometown, USA | 555-123-4567

Code Sample:

[FlexibleGrid/Demos/css/jazzcalendar1draft.css](#)

```
body {
  background-image:url('../images/bg_page_notes.gif');
  font-family:Arial,sans-serif;
}

a {
  text-decoration:none;
  color:#a65c4c;
}

#container {
  width:960px;
  margin:0 auto;
  background:#fff5ea;
}

#main {
  width:900px;
  margin:0 auto 53px auto;
  background-color:#c0cbd0;
}

header {
  background-color:#a67947;
  padding:0 0 0 36px;
}

header h3#tagline {
  float:right;
  font-size:12px;
  color:#fff;
  font-style:italic;
  margin:36px 36px 0 0;
}

nav {
  background-color:#2d5668;
  padding:10px 0 10px 36px;
}

nav ul {
  display:inline;
}

nav ul li {
  display:inline;
}

nav ul li a {
  color:#fff;
  text-decoration:none;
  margin-right:10px;
}

#main #maincontent {
  width:494px;
  float:left;
  padding:36px;
  background-color:#ffedd9;
}

#main #sidebar {
  width:259px;
  float:right;
  padding:36px;
}

#main #sidebar h3 {
  font-style:italic;
  font-size:20px;
}

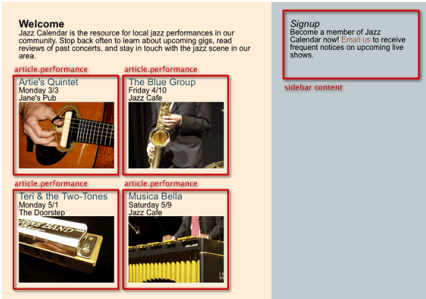
#contactinfo {
  clear: both;
  background-color:#a67947;
  padding:10px 0 10px 36px;
}

#contactinfo p {
  color:#fff;
}
```

A First Complete *Jazz Calendar* Draft

Our next pass at rendering the design matches the original specs: the individual-gig elements sit in the main content well, and the sidebar holds the correct content. The featured gigs are each an <article> (of class "performance"), sized and floated so that they display two across in the focus area of the page.

We've rendered the design successfully - one might take issue with the graphic design choices, perhaps, but we have achieved our first goal: the original design specs are realized as good Web code. But the page is inflexible - resize your browser or view the design on a mobile device and the page remains exactly the same. Our next task is to introduce some flexibility into the page.



Code Sample:

[FlexibleGrid/Demos/jazzcalendar1draft/complete.html](#)

```
Jazz Calendar - Local Live Jazz Music Events

---- CODE OMITTED ----
```

```

Welcome
Jazz Calendar is the resource for local jazz performances in our community. Stop back often to learn about upcoming gigs, read reviews of past concerts, and stay in touch

Article's Quintet
Monday 3/3

Jane's Pub

The Blue Group
Friday 4/10

Jazz Cafe

Tori & the Two-Tones
Monday 5/1

The Doorstep

Musica Bella
Saturday 5/9

Jazz Cafe

---- C O D E   O M I T T E D ----

```

The outer markup is exactly the same as in our first draft: the #container div wraps the whole site, #main sits inside it, and #maincontent and #sidebar float left and right, respectively. We've added elements inside #maincontent - each article of class performance represents one upcoming gig, with title, image, and detail content.

Code Sample:

[FlexibleGrids/Demos/css/jazzallfirstdraftcomplete.css](#)

```

body {
  background-image:url('../images/bg_page_notes.gif');
  font-family:Arial,sans-serif;
}

---- C O D E   O M I T T E D ----

#main #maincontent .performance {
  margin:36px 0 0 0;
  width:230px;
  float:left;
}

#main #maincontent .performance img {
  margin:0 10px 10px 0;
}

#main #maincontent .performance h3 {
  font-size:20px;
}

#main #maincontent .performance h3 a {
  color:#2D5668;
  text-decoration:none;
}

---- C O D E   O M I T T E D ----

```

We give each article of class performance a fixed width (230 pixels) and float it left, so that the gigs display two-across inside of their #maincontent container.

Making the Inflexible Flexible

Consider the page [FlexibleGrids/Demos/inflexibledemo.html](#) - open up the file in your browser. It's a layout similar to our *Jazz Calendar* page, but much more simple. A header (gray background) sits at top, two columns (red and blue) float left and right, respectively, in the middle of the page, and a footer (green) sits at bottom. The 960px-wide design is fixed in width and center justified. The columns split by a ratio of 600px (the left red column) and 360px (the blue sidebar).



But note that, when we resize the browser, the design remains at a fixed width:



Code Sample:

[FlexibleGrids/Demos/inflexibledemo.html](#)

```

I'm Inflexible

I'm Inflexible

main content
main content
main content
main content
main content
main content

```

```

    sidebar content
    sidebar content
    sidebar content
    sidebar content
    sidebar content

```

```

    footer content
    footer content
    footer content

```

Code Sample:
[FlexibleGrids/Demos/css/inflexibledemo.css](#)

```

#container {
  width:960px;
  margin:0 auto;
}

header {
  background-color:#ccc;
}

#maincontent {
  width:600px;
  float:left;
  background-color:#f00;
}

#sidebar {
  width:360px;
  float:right;
  background:#00f;
}

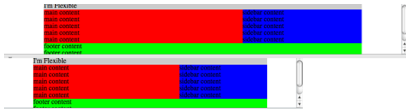
#contactinfo {
  clear: both;
  background-color:#f0f0;
}

```

A revision to this same page now shows a flexible, fluid grid - please check out the page [FlexibleGrids/Demos/flexibledemo.html](#) in your browser, too. Instead of a fixed, 960px container for the page, we now assign a width of 80% to the container. (80% is somewhat arbitrary, but seems close to the spirit of the original, fixed-width design.) Most importantly, we have set the width of the floated elements - the left red main column and the right blue sidebar column - with percentages rather than fixed pixels.

We find the relative percentage-width values by dividing the fixed width of the element by the width of its container element. Thus, for the left (red) sidebar, $600 / 960 = 0.625$ - that's 62.5%. Similarly, the right (blue) sidebar gets a percentage width of 37.5%, since $360 / 960 = 0.375$.

The design now shrinks or expands - with the entire design a percentage of the browser width, and the #maincontent and #sidebar elements scaling proportionally - as the browser resizes narrower or wider:



The markup remains the same - we've not updated the HTML, just the CSS. But this subtle change - moving from fixed pixel widths to flexible percentages - already goes a long way toward rendering the design more responsively.

Code Sample:
[FlexibleGrids/Demos/css/flexibledemo.css](#)

```

#container {
  width:80%;
  margin:0 auto;
}

header {
  background-color:#ccc;
}

#maincontent {
  width:62.5%;
  float:left;
  background-color:#f00;
}

#sidebar {
  width:37.5%;
  float:right;
  background:#00f;
}

#contactinfo {
  clear: both;
  background-color:#f0f0;
}

```

Lesson 1, Activity 4: A Better Jazz Calendar

Let's now apply the same process to the *Jazz Calendar* page - open the file [flexibleGridsDemos/jazz-atl-flexible.html](#) in your browser. We'll use a width of 90% for the outermost (#container) div. The container div (#main) inside of #container gets a width of 93.75%, since $900 / 960 = 0.9375$.

The left column - with the "Welcome" title and the four individual gig listings - gets a width of 54.888889%. The containing element (#main) has a width of 900px; the element itself (#maincontent) has a width of 494px. Thus, $494 / 900 = 0.54888889$, or 54.888889%. No worries about a bunch of decimal points at the end - that's fine.

The same goes for the right column: a 259px-wide element sits inside a container of width 900px. The calculation is $259 / 900 = 0.28777778$, or 28.777778%.



Leaving the padding and margin values for individual elements in pixels would mean that the elements won't line up all the time - as the browser window resizes, the padding and margins would remain fixed, and thus would not render the design so that the logo and the left side of the main navigation, say, line up vertically. We address this by making the paddings and margins percentage based, as we did with the widths.

Here's the CSS for the final design: flexible and (somewhat) responsive.

Code Sample:

[flexibleGridsDemos/css/jazz-atl-flexible.css](#)

---- C O D E O M I T T E D ----

```
#container {
  width:90%;
  margin:0 auto;
  background:#fff5ea;
}
```

```
#main {
  width:93.75%;
  margin:0 auto 5.8888889% auto;
  background-color:#c0cbd0;
}
```

```
header {
  background-color:#a67947;
  padding:0 0 4%;
}
```

---- C O D E O M I T T E D ----

```
#main #maincontent {
  width:54.888889%;
  float:left;
  padding:4%;
  background-color:#ffedd9;
}
```

---- C O D E O M I T T E D ----

```
#main #maincontent .performance {
  margin:36px 0 0 0;
  float:left;
  width:46.558704%;
}
```

---- C O D E O M I T T E D ----

```
#main #sidebar {
  width:28.777778%;
  float:right;
  padding:4%;
}
```

---- C O D E O M I T T E D ----

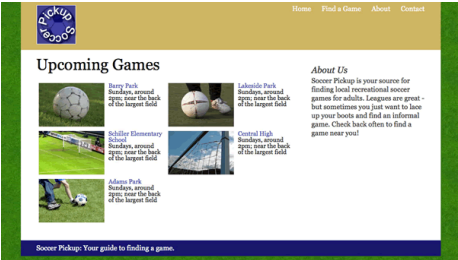
We've gone a long way toward making an originally inflexible page more responsive. But there are still problems - the images, in particular, offer a challenge, since they don't scale to match the changing widths as we resize the browser. And the page certainly isn't optimized for mobile viewing. We'll address these issues as we progress through the course.

Lesson 1, Activity 6: The Pickup Soccer Site

Duration: 20 to 30 minutes.

In this exercise, you will create the first page of a site dedicated to listing local soccer (football, if you aren't in the US!) games - a place for folks to find pickup games at local fields where anyone can join in the game. The recreational soccer player who might not be interested in joining a formal league can find a game, and those looking for players to join their game can list the location and time for their games.

The screen shot below shows the design specs. We'll render the design first as a fixed-width, pixel-based page. Your assignment is as follows:



1. Open [FlexibleGrids/Exercises/pickupsoccerinflexible/index.html](#) and [FlexibleGrids/Exercises/pickupsoccerinflexible/css/style.css](#) in your editor.
2. Edit HTML and CSS:
 - Use a 960px-wide div for the container of the page;
 - Style the left main column as 593px; style the right sidebar column as 259px wide;
 - Display the individual game listings, with each div as 286px wide.

Solution:

[FlexibleGrids/Solutions/pickupsoccerinflexible/index.html](#)

```
<!DOCTYPE html>
<html>
<head>
  <title>Soccer Pickup</title>
  <link rel="stylesheet" type="text/css" href="css/reset.css" />
  <link rel="stylesheet" type="text/css" href="css/style.css" />
</head>

<body>
  <div id="container">
    <header>
      <a href="index.html"></a>
      <nav id="mainnav">
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="find-a-game.html">Find a Game</a></li>
          <li><a href="about.html">About</a></li>
          <li><a href="contact.html">Contact</a></li>
        </ul>
      </nav>
    </header>
    <div id="maincontent">
      <h1>Upcoming Games</h1>
      <div class="game">
        <a href="#"></a>
        <h3><a href="#">Barry Park</a></h3>
        <p>Sundays, around 2pm; near the back of the largest field</p>
      </div>
      <div class="game">
        <a href="#"></a>
        <h3><a href="#">Lakeside Park</a></h3>
        <p>Sundays, around 2pm; near the back of the largest field</p>
      </div>
      <div class="game">
        <a href="#"></a>
        <h3><a href="#">Schiller Elementary School</a></h3>
        <p>Sundays, around 2pm; near the back of the largest field</p>
      </div>
      <div class="game">
        <a href="#"></a>
        <h3><a href="#">Central High</a></h3>
        <p>Sundays, around 2pm; near the back of the largest field</p>
      </div>
      <div class="game">
        <a href="#"></a>
        <h3><a href="#">Adams Park</a></h3>
        <p>Sundays, around 2pm; near the back of the largest field</p>
      </div>
    </div>
    <div id="sidebar">
      <h3>About Us</h3>
      <p>Soccer Pickup is your source for finding local recreational soccer games for adults. Leagues are great - but sometimes you just want to lace up your boots and find an informal game. Check back often to find a game near you!</p>
    </div>
    <div id="footerinfo">
      <p>Soccer Pickup: Your guide to finding a game.</p>
    </div>
  </div>
</body>
</html>
```

A nav tag wraps an unordered list for the navigation elements. We use divs of class game to render the individual game listings.

Solution:

[FlexibleGrids/Solutions/pickupsoccerinflexible/css/style.css](#)

```
---- CODE OMITTED ----

#container {
  width:960px;
  margin:0 auto;
  background:#fff;
}

header {
  background-color:#d0b462;
}

header h3#tagline {
  float:right;
  font-size:12px;
  color:#fff;
  font-style:italic;
  margin:36px 36px 0 0;
}

---- CODE OMITTED ----

#maincontent {
  width:593px;
  padding:10px 0 36px 36px;
  float:left;
}

#maincontent h1 {
  font-size:36px;
  margin:5px 0 20px 0;
```

```

)

#maincontent .game {
float:left;
width:286px;
line-height:14px;
font-size:14px;
margin:5px;
}

#maincontent .game img {
float:left;
margin:0 10px 0 0;
}

#sidebar {
width:259px;
float:right;
padding:36px;
color:#333;
line-height:20px;
}

#sidebar h3 {
font-style:italic;
font-size:20px;
margin:0 0 5px 0;
}

#footerinfo {
clear: both;
background-color:#f5f570;
padding:10px 0 10px 36px;
}

#footerinfo p {
color:#fff;
}

```

We float the two main columns left (#maincontent, at 593 pixels wide) and right (#sidebar, at 259 pixels wide). The nav element floats right. Individual .game elements float left, with a width half that of the containing element.

Lesson 1, Activity 8: Making *Pickup Soccer* Flexible

Duration: 20 to 30 minutes.

Let's apply the same process to the *Pickup Soccer* page that we did to the *Jazz Calendar* page: we'll change a fixed-width design into a more flexible, percentage-based design. Keep in mind the key rule here: percentage widths are calculated by dividing the pixel width of the element by the pixel width of its containing element.

1. Open [FlexibleGrids/Exercises/pickupsoccerflexible.css](#) in your file editor.
2. Using the fixed-width page from exercise 1 as a starting point, find percentage widths for each element. The markup won't change - just the CSS.
3. Be sure to update margins and padding for each element, so that things line up regardless of the size of the browser window.
4. View [FlexibleGrids/Exercises/pickupsoccerflexible/index.html](#) in your browser to check your work. Don't forget to resize your browser window (horizontally), to make sure that things work correctly!

Solution:[FlexibleGrids/Solutions/pickupsoccerflexible.css](#)

The left (`#maincontent`) column gets a width of 593/960, or 61.7708333333%. Similarly, the right (`#sidebar`) column gets a width of 259/960, or 26.9791666667%. Paddings and margins get a corresponding percentage width.