

Flexbox

A little history

15 years ago we built our web interface structures with tables. Tables were reliable because content inside a table cell would never expand beyond the wall of the cell. You could tell the table, and / or the table data cell children how wide to be using pixels or percents, and it rarely backfired. You could nest tables inside tables to get even more granular control. In this example, (which you can still easily do in Dreamweavers Design view) I first built a one column, two row table. Inside each row I inserted a one row 3 column nested table. Using this structure, you can easily drop logo and banner images into cells 1 and 3, along with an expander pattern image from Photoshop in cell 2. Then you can stack buttons in cell 4, content in cell 5, some action boxes in cell 6 and you have state of the art web design from last century.

1	2	3
4	5	6

seemed like too much to ask. We had that control with tables last century. Surely Al Gore, who invented it all, could improve on it.

In this book so far we've built an interface structure that relies on floats and positioning. It is very reliable once all the bugs are worked out, and it performs well in all browsers going back several years as of 2015.

Enter flexbox

Think of flexbox as a flexible 3 car garage. By giving the garage a property of **display: flex;** all 3 of the bays in the garage are exactly the same size: height and width. If anything gets bigger, everything maintains the same relationship. For example, if the viewport (browser) gets bigger, the garage gets bigger, and the bays inside the garage maintain their relationship.

If we put a bigger car in one of the garage bays, that bay will get taller to accomodate the content, and it's sibling bays will get taller too, to maintain the equal relationship between the 3 garage bays.

If you remember the dark days of tables, you are probably thinking that this was just how tables behaved, and you are right. But even better, with flexbox, we can wrap the garage bays...that was never possible with table data cells. We can also easily control alignment both inside and outside the bays in both x and y directions, including justify. Plus we can even re-order the bays, all without touching the html markup.

Then search engines got smarter and it was found that content inside tables was not easily searchable. Establishing relationships between images and text in adjacent table rows or columns required a lot of extra programming in the `<table>`, `<tr>` and `<td>` tags.

Something better was required and it was discovered that unordered lists were friendlier to search engines because of the implied relationships between content inside the `` `` tags.

By using a complex matrix of `` `` and `<div>` tags, combined with floats and positioning (position: absolute, relative) the web gradually moved towards an interface design environment that as almost as effective as tables used to be.

We just finished out our interface using that very structure.

Behind the scenes the web community and browser developers were always searching for something easier and more reliable. Everyone wanted a structure that allowed for simple alignment capabilities in both the x and y direction. And despite the fluid nature of the viewport (browser width), we needed a way to accurately control column and row width and height. It never

Flexbox for the menu

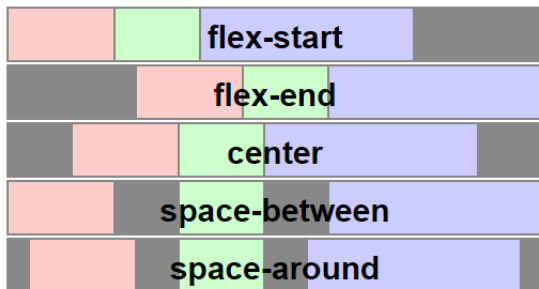
STEP ONE: Edit these two style sheet rule as shown in bold brown

NOTE: We told the parent `` tag to have a **`display: flex;`** property. This means that its children (the list items) are now displaying in the default **row** behavior. They are also using the default left alignment. But, because we are using flexbox, we now have a lot of tools at our disposal to make these buttons much prettier using alignment properties that have been common in word processing programs for decades.

NOTE: for a full explanation of the Flexible Box Layout model, go to this address and search for **space-around**:

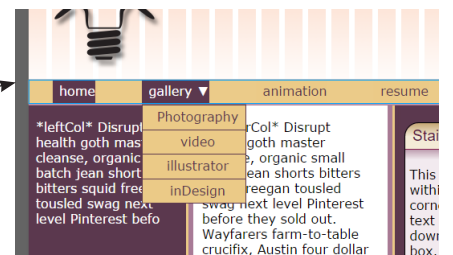
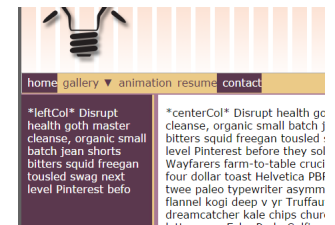
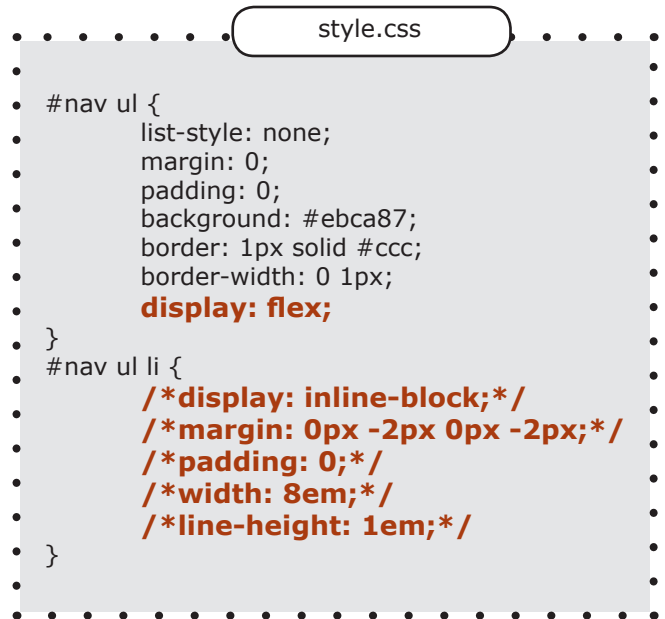
<http://www.w3.org/TR/css3-flexbox/#flex-property>

Screenshot below is from the w3.org



STEP TWO: add **`justify-content: space-around;`** to the `#nav ul`

On our menu, **`justify-content`** has caused the `` tags to spread out evenly across the parent `` element. The **`space-around`** property does some fancy math to adjust margins between list items. But there is still a problem: they get their width based on the text in the anchor tags. We can fix that by speaking directly to the child `` tags.



Flex grow, shrink, basis-width

Similar to how you can tell a `<div>` or `<td>` to have a width property in pixels, em's or percents, we can also speak to the child `` flex elements.

STEP ONE: edit the `#nav ul li` rule as shown

This tells the `` tags (children of the `` flexbox parent) to neither grow nor shrink. And to have a fixed width of 8em, which currently matches the drop menu.

But notice what happens when the viewport gets narrower than 40ems (5 `` @ 8em each). The `` blows through the sides of the parent wrapper. You should play around with different combinations of numbers for grow, shrink and basis-width to see how it behaves.

STEP TWO: Modify `#nav ul li` as shown.

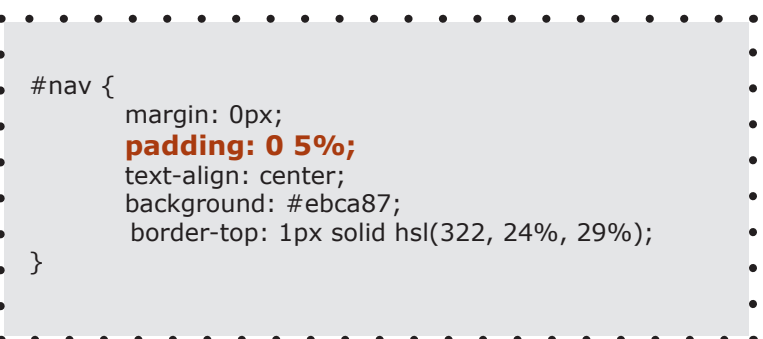
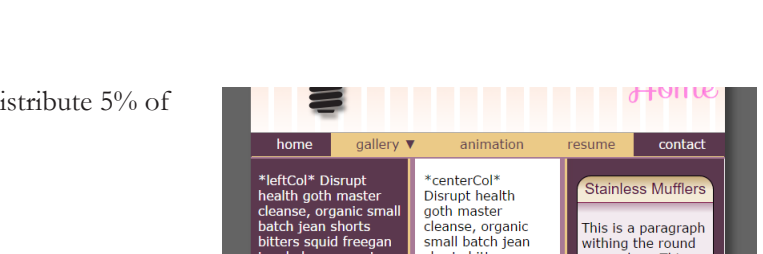
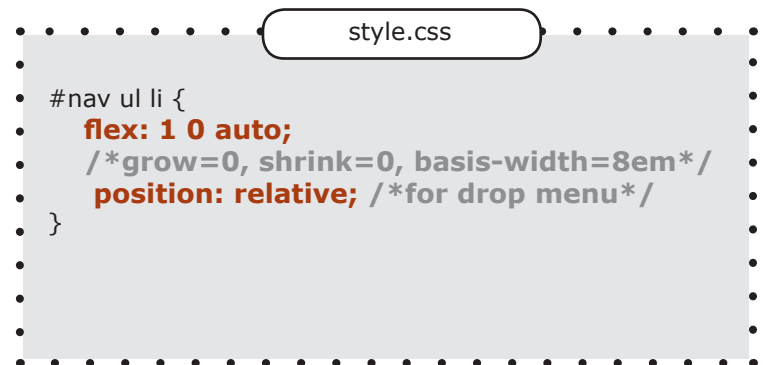
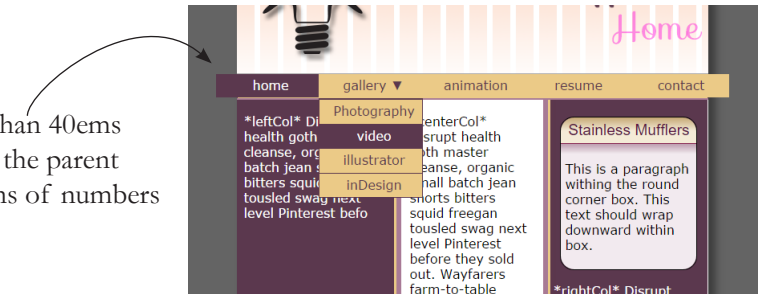
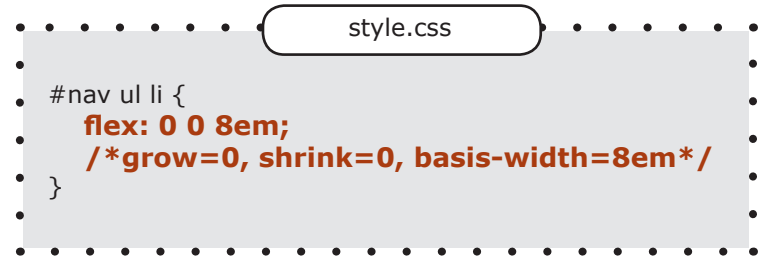
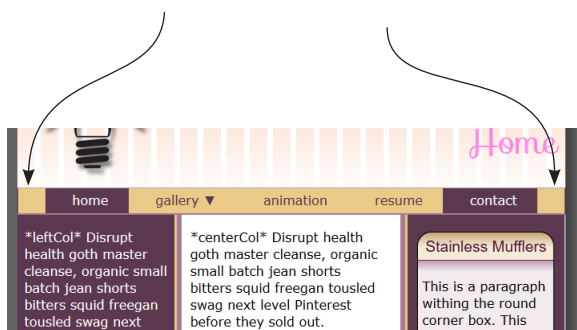
NOTE: the position relative is required for the drop menu both for proper width, and because we absolutely position the child `` dropdown.

This is perhaps our best look so far, as the `` tags, are growing to fill the available space. The dropdown width doesn't match, but we can fix that later. To get that margin back on the left and right sides of the buttons, we can modify margin and padding on the parent ``

STEP THREE: delete the border property on `#nav ul`

STEP FOUR: then modify the `#nav` property as shown to distribute 5% of padding on either side of the `` tag.

This gives us a reasonably pretty menu that survives larger viewport widths. (Still need to fix that dropdown width)



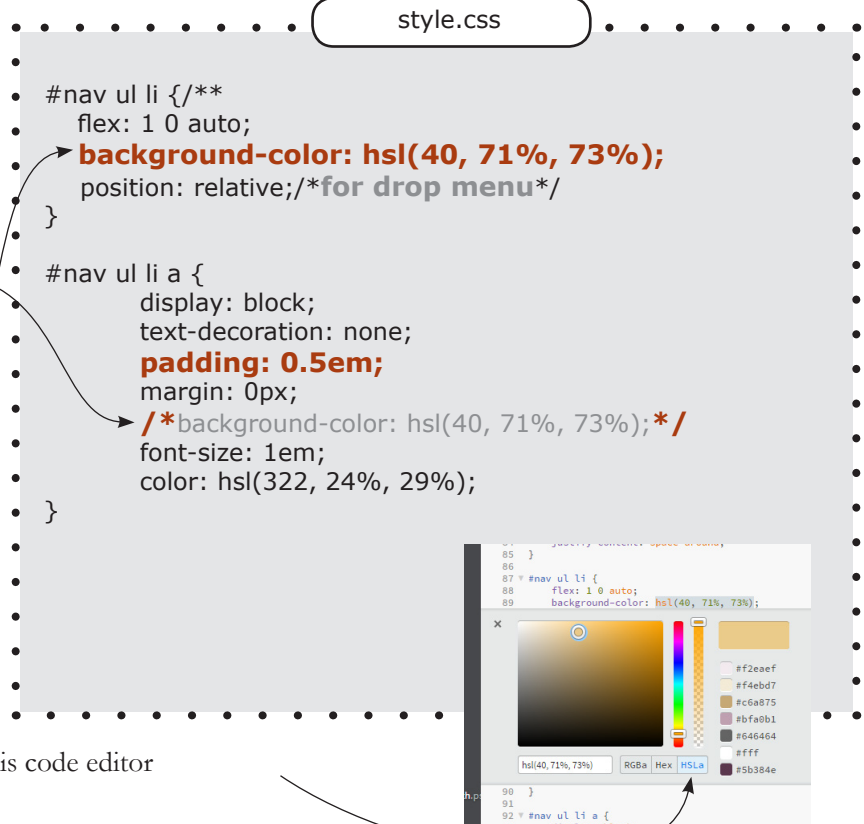
Menus

To get more precise control over our rollovers we need to move the background colors and hover effects out of the `<a>` tags and onto the `` tags.

STEP ONE: cut the background property out of `#nav ul li a`

STEP TWO: paste it into the `#nav ul li` rule

You may have noticed that I've changed a lot of my hex colors to `hsl()` colors. I'm using a new free code editor called Brackets. If you click in a hexadecimal color code and press `CTRL + e`, Brackets will launch a color picker where you can switch your color from hex to HSLa by simply clicking the HSLa button. This color mode, and this code editor makes a lot of things easier, including animations.



STEP THREE: Add this new rule to fix the color of the gallery anchor text when the drop menu is hanging

STEP FOUR: Cut the background-color property out of `a:hover` so it is just a text color change.

STEP FIVE: Add a new `li:hover` rule and paste in the background-color you cut from the `a:hover` rule. This should bring back your normal rollover look, but now it's happening on the `` tag.



Button eyecandy

Next I would like to add inline borders to the sides of the buttons on rollover. Because borders on rollover make the buttons (list items) wider, we need to also add the borders to the list items in their normal "off" state. We can do this by adding invisible (alpha: 0;) borders, and then simply turn them on at mouseover. They will be like Christmas lights on a tree in August...waiting for electricity.

STEP ONE: add invisible borders to the sides of `#nav ul li`. Because I'm using a white border on tan, I made it 2 pixels, but with a darker color scheme such as blue and grays, 1 pixel is plenty.

```
style.css
#nav ul li {
  flex: 1 0 auto;
  background-color: hsl(40, 71%, 73%);
  position: relative;
  border: 2px solid hsla(0, 0%, 100%, 0);
  border-width: 0 2px;
}
```

STEP TWO: Modify `#nav ul li:hover` as shown. This will switch the border color from invisible to visible on rollover (hover). The lights on the tree should light up now.

```
style.css
#nav ul li:hover {
  background-color: hsl(322, 24%, 29%);
  border-color: hsl(0, 0%, 100%);
}
```

urhere overhaul

Before we fix the appearance of our urhere buttons, let's re-think it. Currently, we have anchor tags with a class of urhere:

```
<a href="index.html" class="urhere">home</a>
```

That gets affected by our style sheet of `a.urhere{ /*declarations here*/ }`

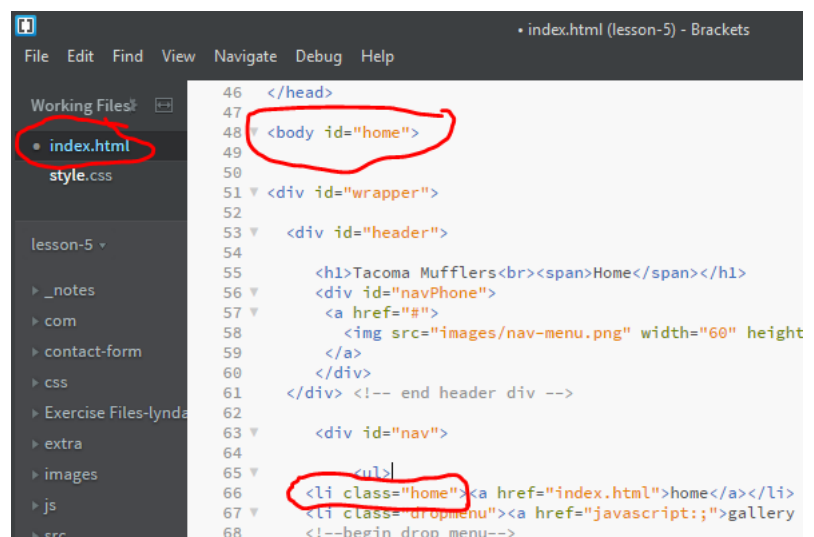
There is a better way to do the urhere function, but it will require some changes to the html markup. Here they are:

STEP THREE: Open up your `index.html` file and modify the body tag like this:

```
<body id="home">
```

STEP FOUR: Also on `index.html`, modify the home button by deleting the urhere class, and rewriting the entire list item as shown:

```
<li class="home"><a href="index.html">home</a></li>
```



urhere overhaul

STEP ONE: Repeat that process on *all* your html pages. Pictured are the changes on the **gallery.html** page. Note there was already a class of dropdown on the tag. It is ok to have dual classes on an element, as long as they don't conflict with each other.

NOTE: if you like, you can add the <li class="[page-name]"> to all the tags in the nav on all the pages. The urhere function only gets triggered if there is a matching <body id="[page-name]"> tag. This means you can do things like import your nav div with a php include, while maintaining the urhere function.



STEP TWO: In your style sheet, delete the **a.urhere** rule and replace it with these two rules. This new code will bring back your urhere function but it will be based on whether there is a matching id and class on the relevant tags before it lights up the button.

If you've never seen a style sheet rule like this before...think of it as chaining selectors to one set of declarations. I'm saying: "if there is an id of home, with a child class home; or id gallery & child class gallery, etc.... the following rules shall apply.

STEP THREE: Pictured is this look on our school project, as well as my personal website remodel scheme. I'm using 1 pixel button borders because a darker color scheme allows it.

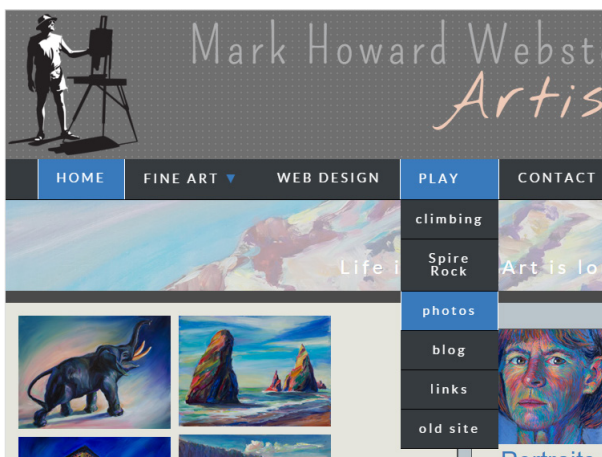
style.css

```

#home li.home,
#gallery li.gallery,
#animation li.animation,
#resume li.resume,
#contact li.contact {
  background-color: hsl(322, 24%, 29%);
  border: 2px solid hsl(0, 0%, 100%);
  border-width: 0 2px;
}

#home li.home a,
#gallery li.gallery a,
#animation li.animation a,
#resume li.resume a,
#contact li.contact a {
  color: hsl(0, 0%, 100%)
}

```

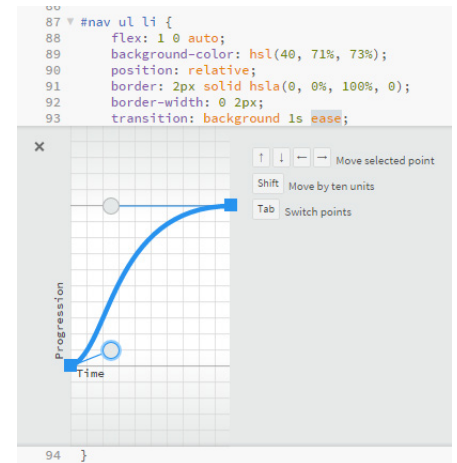


Animated button transitions

STEP ONE: add an animation transition to `#nav ul li`

```
style.css
#nav ul li {
  flex: 1 0 auto;
  background-color: hsl(40, 71%, 73%);
  position: relative;
  border: 2px solid hsla(0, 0%, 100%, 0);
  border-width: 0 2px;
  transition: background 1s ease;
}
```

STEP TWO: if you are using the free Brackets code editor (<http://www.brackets.io>) click in the word **ease**, and press CTRL + e. This will launch the animation editor. Yank on the bezier curve handles to modify the easing of the animation. To make it longer change the **1s** to **1.5s**. Shorter would be **0.5s**. It's measured in one thousands of a second.



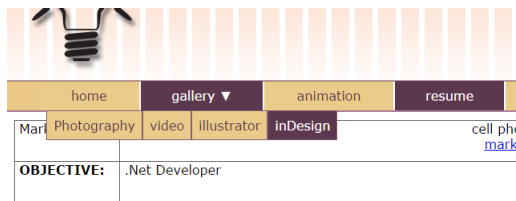
STEP THREE: add these 3 new style sheet rules. They will bring some control over the colors and borders on our dropdown menu. Note that I am using a lot of specificity in the style sheet selectors to get control over the colors.

It's time to fix the width on that drop menu. Because the flex property allows the parent list items to grow, the child list items need a way to grow with their parent.

STEP FOUR: edit this style sheet rule by changing the **display: block;** to **display: flex;**
Not quite what we had in mind!

```
style.css
#nav ul li.dropdownmenu:hover li {
  border: 1px solid hsl(322, 18%, 44%);
  border-width: 1 0 0 0;
}
#nav ul li.dropdownmenu:hover li a {
  color: hsl(322, 24%, 29%);
}
#nav ul li.dropdownmenu:hover li a:hover {
  color: hsl(0, 0%, 100%);
}
```

```
style.css
#nav ul li.dropdownmenu:hover ul.leveltwo {
  display: flex;
}
```



Drop menu width

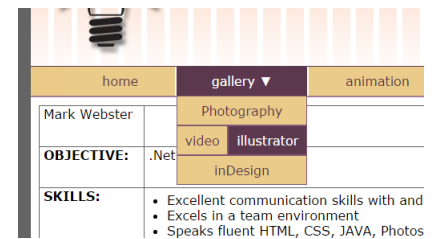
Our problem is that the `display: flex;` property on the parent `` tag is displaying as a row. We need to enable the “wrap” property for that flex element. This allows the

`<li class="dropdown">` flex item to wrap. It’s weird because it’s positioned absolutely...but in its own way it is wrapping, and it gives us a matching width on the drop to the parent menu.

STEP ONE: Edit the `#nav ul` style sheet rule

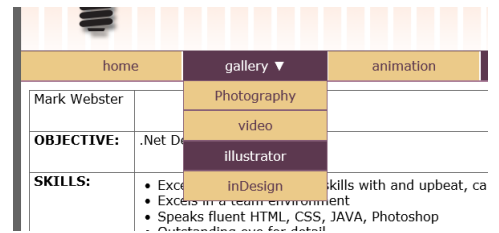
```
style.css
#nav ul {
  list-style: none;
  margin: 0;
  padding: 0;
  background: hsl(40, 71%, 73%);
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
}
```

Note how the dropdown is wrapping. It matches the width of the main buttons, but the wrap property cascades down to the child list items. We can fix that by speaking to the `ul ul` child flex items.



STEP TWO: add this style sheet rule. By setting the basis-width to 8em, we prevent it from getting small enough to wrap. It will still grow and shrink with the viewport, but 8em seems to be a happy value.

```
style.css
#nav ul li.dropdown:hover ul.leveltwo {
  display: flex;
}
#nav ul li.dropdown:hover ul.leveltwo li {
  flex: 1 1 8em;
}
```



Mobile menu

The smartphone menu needs some tweaking. It works ok for an iPad, but it's too large to fit on the screen of an old iPhone, and not aligned correctly.

STEP ONE: Down in media queries, edit your style sheet rules as shown. Go through them carefully one rule at a time. I've tweaked various things until it seems happy on all devices and viewports.

style.css

```

/*media queries area*/

#nav{
  margin: 0;
  position:relative;
  border:none;

  /*this hides nav on phone at first arrival,
  afterward jquery controls it*/
  display: none;
}

#nav ul {
  width: 8.5em;
  position:absolute;
  right:0;
  top: 0;
  border-bottom: 1px solid hsl(322, 24%, 9%);
}

#nav ul li { /*removed some declarations here*/
  border-top: 1px solid hsl(322, 24%, 29%);
  flex: 0 0 8.5em;
  border-left-style: none;
  border-right-style: none;
}

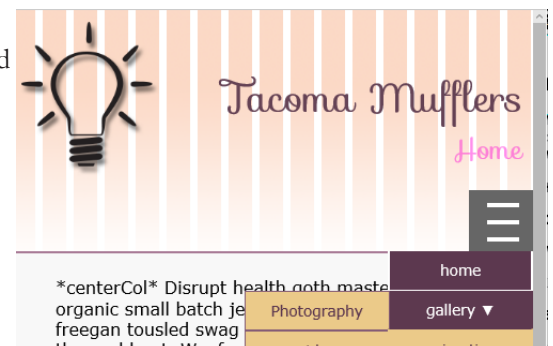
#nav ul li: hover{
  border-left-style: none;
  border-right-style: none;
}

#nav ul li a {
  font-size: 0.85em;
  padding: 0.7em;
}

#nav ul ul { /*removed some declarations here*/
  position:absolute;
  right: 8.4em;
  top: 0;
  border-style:
  none;

```

Note that the flyout menu arrow is on the wrong side of the word (gallery) and it's pointing down, not left. We can fix that but it will require altering the html markup. We will have to put both arrows (down and left) in there at the same time, and tell them to show or hide depending on viewport width.



Flyout arrows

STEP ONE: Locate your `<li class="dropdown">` and edit it as shown in bold brown. This will give you arrows on both sides of the word gallery.

```
index.html
<li class="dropdown">
  <a href="javascript:;">
    <span class="small">&#x25C0;</span>
    gallery
    <span class="big">&#x25BC;</span>
  </a>
  <!--begin drop menu-->
  <ul class="leveltwo">
    <li><a href="gallery.html">Photography</a></li>
    <li><a href="video.html">video</a></li>
    <li><a href="illustrator.html">illustrator</a></li>
    <li><a href="indesign.html">inDesign</a></li>
  </ul>
  <!--end drop menu-->
</li>
```

STEP TWO: cut and paste that new code into all of your html pages.

STEP THREE: up in the top nav area of your style sheet, add this new rule. It will hide the arrow meant for small screens.

```
style.css
.dropdown a .small {
  display: none;
}
```

STEP FOUR: Down in media queries, add these two new rules. They will show the left arrow, and hide the down arrow.

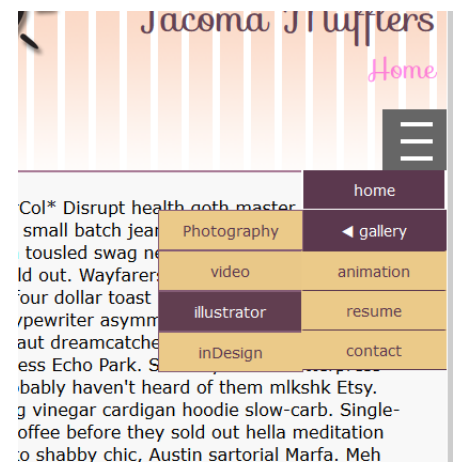
STEP FIVE: Clean up your custom javascript. Remove any extra stuff to make it match this exactly.

```
index.html
<script>
$(document).ready(function(){
  $('#navPhone').click(function(e){
    $('#nav').slideToggle(300);
    // cancel the default action - fixes page jump
    e.preventDefault();
  });//end navPhone.click function
$(window).resize(function(){
  if($(window).width() > 712){// must be a computer
    $('#nav').show();
  } else {// window is < 712 = smartphone
    $('#nav').hide();
  }//end if window.width
});//end window.resize
});//end document.ready

</script>
```

```
style.css
/*media queries area*/

.dropdown a .small {
  display: inline-block;
}
.dropdown a .big {
  display: none;
}
```



Move custom jquery to an external script

STEP ONE: Make sure you have your website backed up before we do the next step. Get out your flash drive and copy/backup the entire website

STEP TWO: Examine the code at the top of your website. Note the lines of code numbered 11, 13 & 34 in the screen shot pictured here. These lines import code. Lines 11 & 13 import javascript libraries. These are also commonly known as ".js" files because of the file extension *.js, which stands for javascript.

Next note the code inside the starting and stopping `<script></script>` tags. Although this is code written in the jquery language, it can be imported from an external js file. For example: rather than having 19 lines of custom jquery script, we can import it from an external js file with one line of script:

`<script src="js/myjquery.js"></script>`

STEP THREE: Highlight and **cut** all the jquery script between (but not including) the starting and stopping `<script></script>` tags. If your code looks like the screenshot above, **cut** the code from line 16 to 31. Then delete the starting and stopping `<script></script>` tags

STEP FOUR: In Dreamweaver (or any code editor), choose file>new>Blank Page>JavaScript

STEP FIVE: Choose edit>paste to bring in your cut jquery code.

STEP SIX: Choose file>save as. Navigate down into your "js" folder. Name the file `myjquery.js`

```

1  <!doctype html>
2  <html lang="en">
3
4  <head>
5
6  <meta charset=utf-8>
7  <meta name="viewport" content="width=device-width, initial-scale=1.0">
8  <title>
9  Tacoma Mufflers - Resume
10 </title>
11 <script src="http://use.edgefonts.net/sofia.js"></script>
12
13 <script src="js/jquery-2.1.4.min.js"></script>
14
15 <script>
16   $(document).ready(function(){
17
18     $('#navPhone').click(function(e){
19       $('#nav').slideToggle(300);
20       // cancel default action - fixes page jump on smartphones
21       e.preventDefault();
22     }); //end click function
23
24     $(window).resize(function(){
25       if($(window).width() > 712){ // must be a computer
26         $('#nav').show();
27       } else { // window is < 712 = smartphone
28         $('#nav').hide();
29       } //end if window.width
30     }); //end window.resize
31   }); //end document.ready
32
33 </script>
34 <link rel="stylesheet" type="text/css" media="screen" href="style.css">
35
36 </head>
37
38 <body id="resume">
39

```

