Layouts

Positioning, Floats, Flex, Media Queries, and Navigation

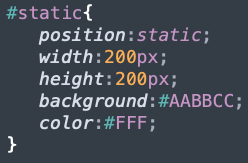
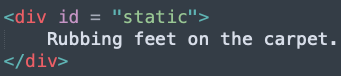
What separates a professional and an amateur web developer is their understanding of how HTML elements are positioned, and how we can move elements from their natural spots in a web page. Elements are then positioned using the top, bottom, left, and right properties. Understanding these basics is fundamental to any web developer.

## Positioning

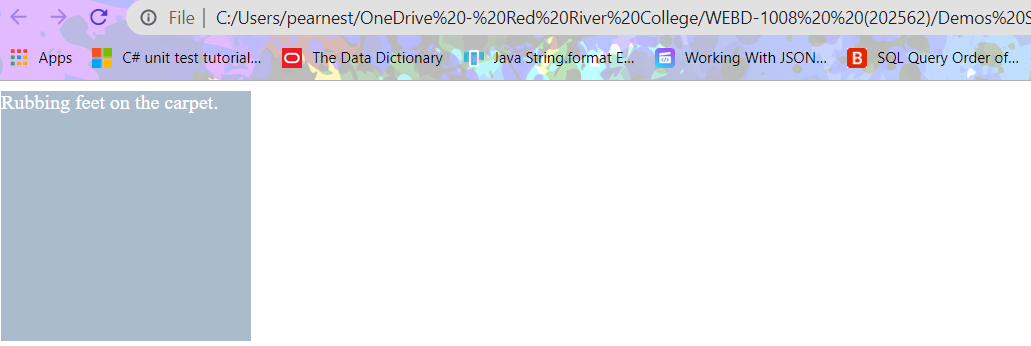
### Static

The *default* position for any HTML element is *static*. This means the element will position itself exactly where we would expect it to.

Consider this example:



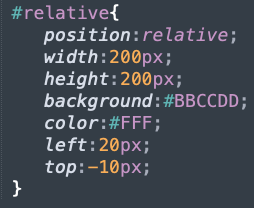
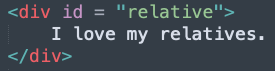
This div, with no margin, should appear in the very top left of the browser window.



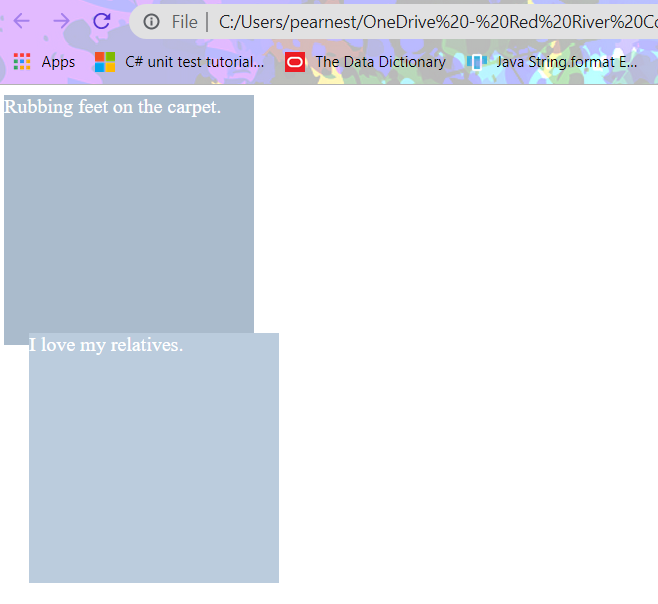
And it does. As expected. There are other position options we can use to move our elements around.

### Relative

Using a position of *relative* gives us access to four properties: top, right, bottom, and left. With those we can move an element relative to where it would have showed up in the natural flow of the page.



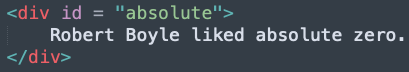
By changing the position, and adding values to the left and top properties, we can move this element. Note that this element will be moved *from* the left 20 pixels, and *from* the top -10 pixels. This would be the result:



### Absolute

**An *absolute* positioned element is positioned relative to the first parent element that has a position other than *static.***

An element using *absolute* positioning is removed from the flow of the page, and placed exactly where the coordinates provided in the CSS say it will be placed.

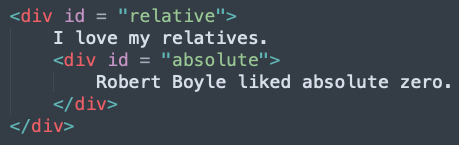


Remember that *absolute* positioned elements are removed from the flow of the page. Since no parent elements have a position other than *static*, this element will appear at the top right of our browser window.

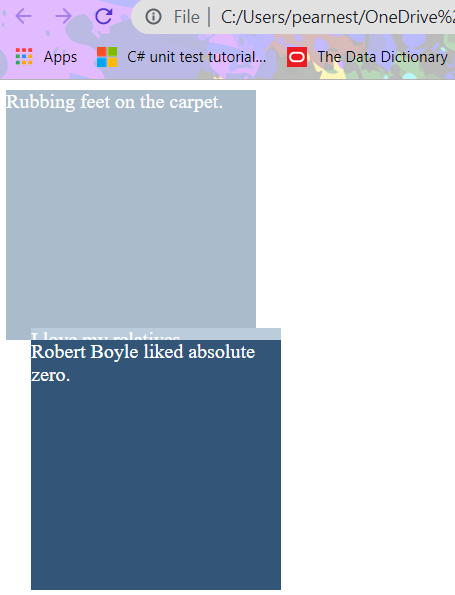


But what if a parent had a position other than static? That means the *absolute* positioned element would be positioned relative to *that* parent element.

Consider if we moved the *absolute* div tag into the *relative* one.

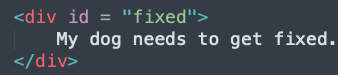


Now where would the *absolute* div appear? Not at the top right of the browser window, but the top right of the parent element with a position other than *static.*

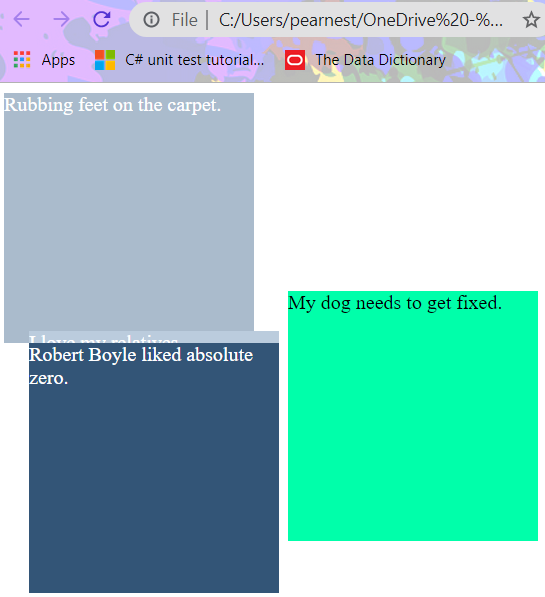


### Fixed

Similar to *absolute* positioned elements, *fixed* elements are also removed from the flow of the page, but when positioned, they never move from that spot. (thus, the “fixed”)

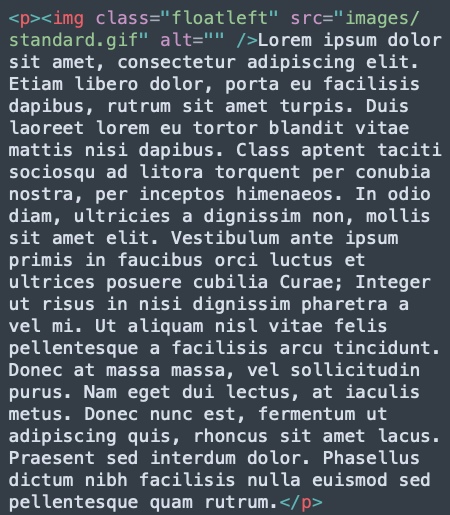


This element will always appear 40% from the top of the window, and 40% from the left. No matter the size of the window.

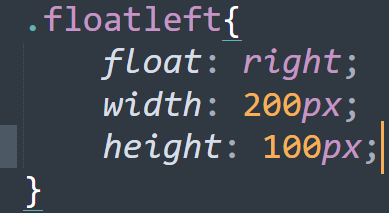


## Floats

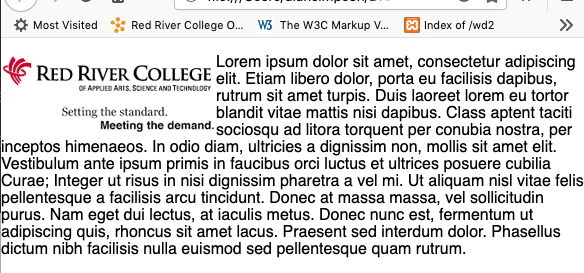
A common practice in any print media is to have an image, and then have the text related to that image float around it. By default text next to an image will align itself to the bottom of that image.



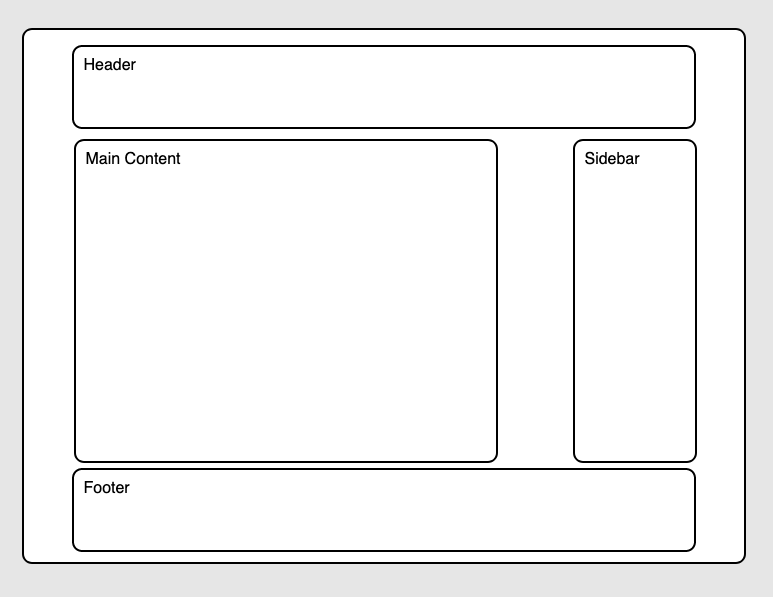
Here we have an image inside a paragraph. With some CSS, we’ll apply a float class to that image, while also reducing its size (not its physical size, just its dimensions).



Which would produce this result:



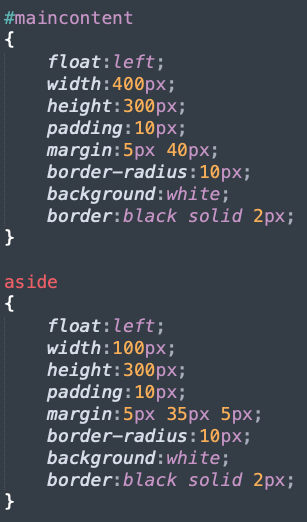
Floats can also be used to help lay out HTML elements on a page. Oftentimes web pages will have a main content area with a sidebar next to it. As block elements though, this doesn’t happen naturally. So, we can use floats. Consider this layout:



With the HTML:

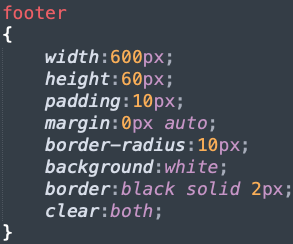


What we can do is apply floats to the section and aside tags in our CSS.



There’s one issue though. With floats, every succeeding element will try to wrap itself next to the previous one, like we saw with the text wrapping around the image. As it is right now, the page would collapse and the footer would wrap to the top, appearing behind the section and aside.

To get around this, we need to tell the footer to ignore the floats above it. Float has a sister property called *clear,* which is used to ignore all preceding tags.



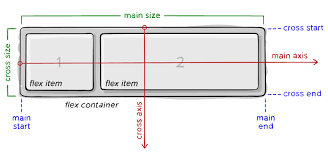
## Flex

The Flexbox Layout aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word "flex").

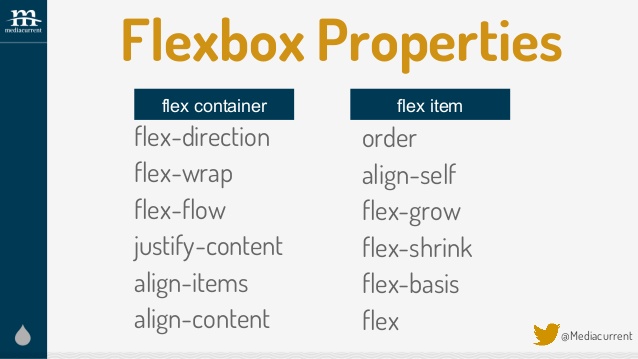
The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate all kinds of display devices and screen sizes). A flex container expands items to fill available free space or shrinks them to prevent overflow.

Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility (no pun intended) to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).

**The Flexbox Model**

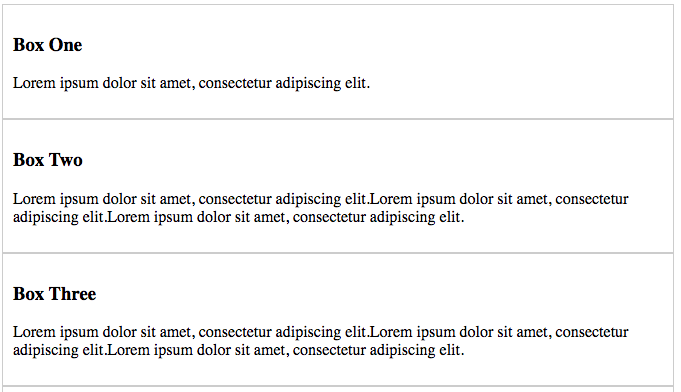
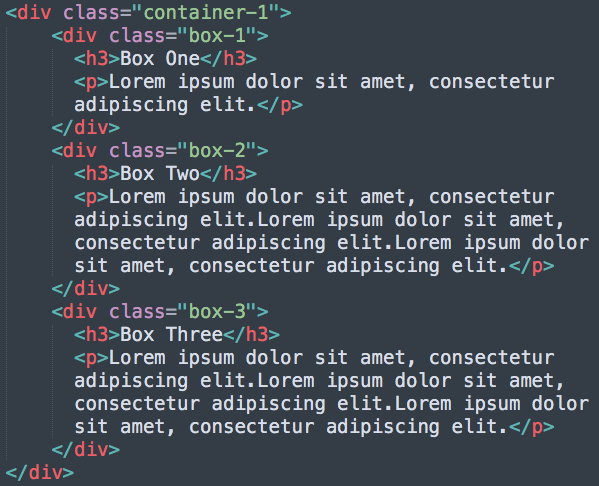


**Flexbox Properties**

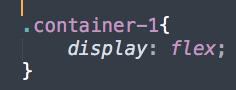


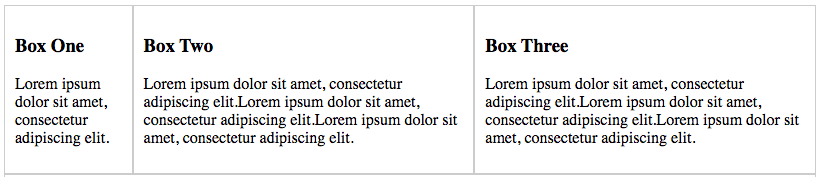
Flex uses containers to control the layout of its child elements.

Consider this example:

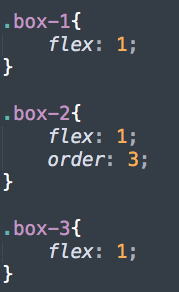


Here the boxes are laid out exactly as we would expect them to. If we wanted to have them layout horizontally we can add a flex to the container:

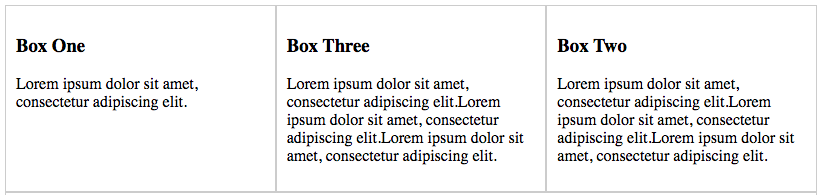


This would be the result in the browser:

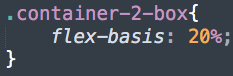
Notice boxes two and three are wider than box one. This is because these boxes have more content than Box One. To make them equal width, we apply flex properties to these items.

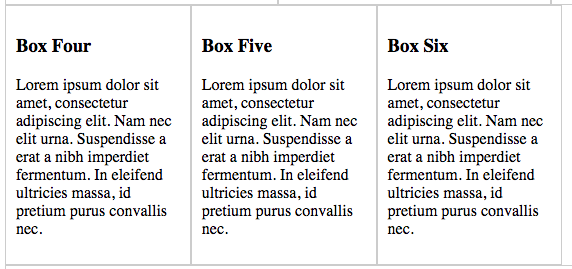


This makes the widths equal, and *order:3* moves the 2nd box to be the one on the right.

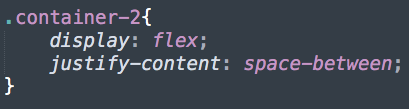


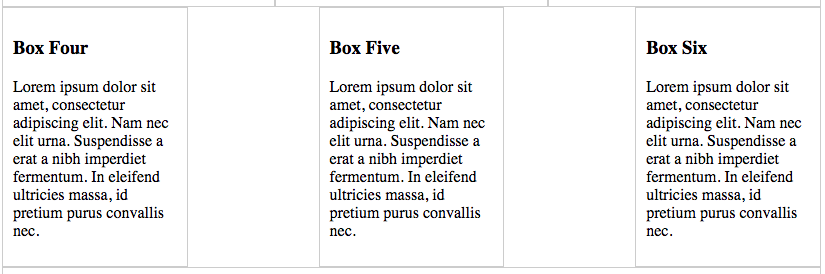
To give flex-items widths, we use *flex-basis* to apply values:





There are a few flex properties that can control what to do with the remaining margins. For example, if we wanted the space to be distributed evenly between the boxes:





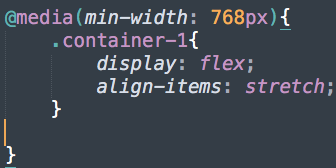
## Media Queries

Media queries are used to handle resolution situations. Many websites are created with a desktop computer, or larger resolution in mind. But what if we want to access that same site on a mobile device? We can create a media query that will alter the look of the site depending on what resolution the page is being viewed on.

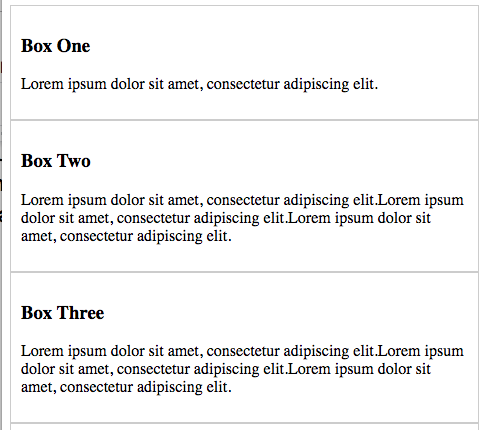
The first thing to do is to add this line to the HTML to let the browser know that this page is going to be responsive.



Those boxes in the earlier example would look very compact on a lower resolution. To remedy this we can declare a media query in our CSS, and say that flex should only apply if the width is a minimum amount.



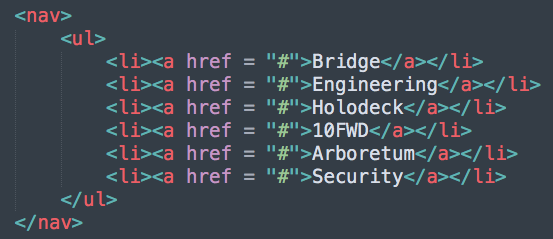
Viewed in a minimum width browser:

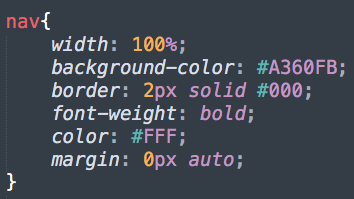


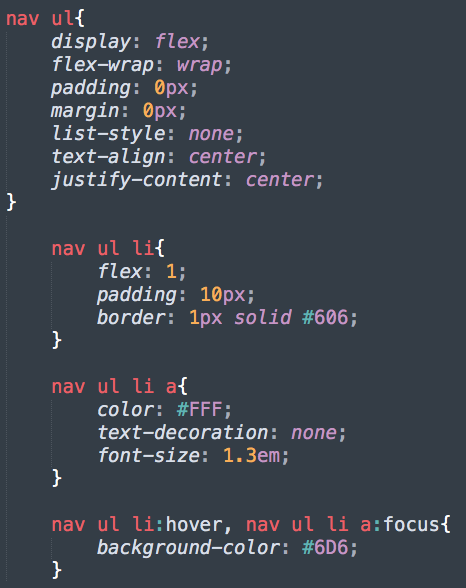
Note that there are a few media query strategies. You may want to set a max-width and do your mobile styles inside the query, rather than what was done above. That’s also acceptable.

## Navigation

To create navigation structures in HTML, lists are usually used. They provide a lot of styling options and intrinsic structure that a navigation list should have.







Using *flex* will make the list horizontal by default.



When the mouse hovers over a link: 

But how will this navigation look in mobile view? By applying the flex styles in a media query, we can account for lower resolution screens as well.

