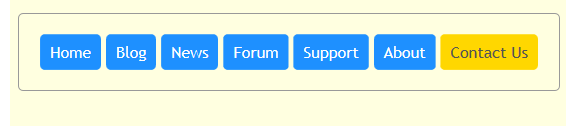
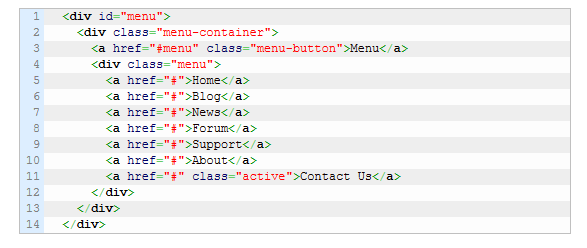
Responsive CSS menu with a disappearing “Menu” button

When viewed on a large screen this menu looks like the image shown below. The idea of this menu is that it has a “Menu” button to open a menu in small screens (horizontal one disappears and a menu button appears). When the menu’s been opened the “Menu” button is hidden. It’s pure CSS responsive menu with no JavaScript through the :target CSS pseudo class. Let’s walk step by step on concepts of this menu:

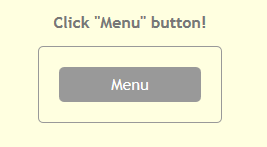
1. It looks like a classic horizontal menu on the desktop. The menu has inline-block items.



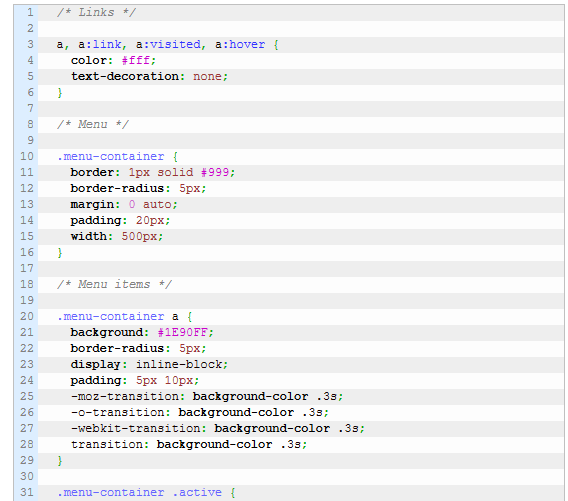
1. The menu has a main container with an id, “menu”. Here’s the HTML for the menu:

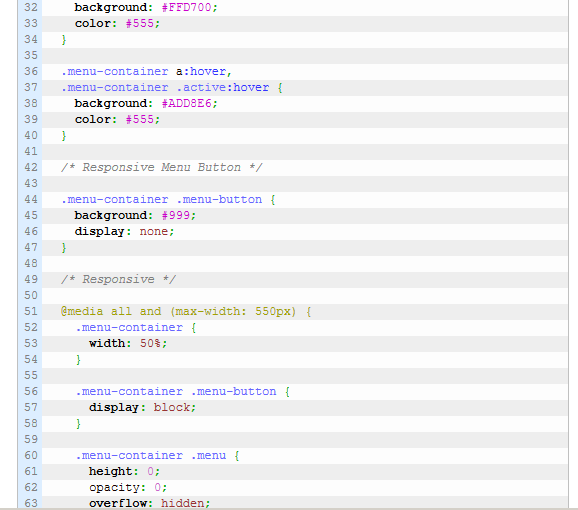


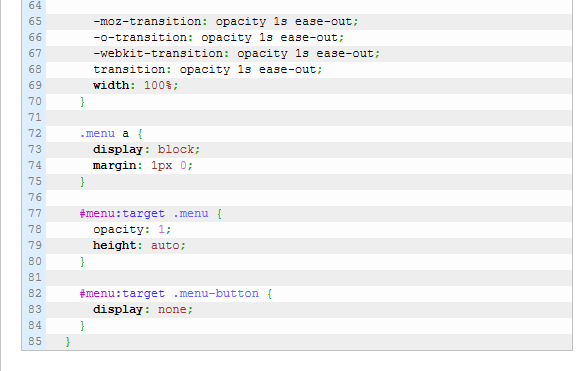
1. Besides the menu, there’s another container that has a hidden link by default. It’s a “Menu” button for responsive. Also, this container has the menu itself.
2. The “Menu” button has “#menu” as a “href” attribute value. This allows us to scroll the screen to the “menu” id when the “Menu” button is clicked (remember your internal links from semester 1!).
3. If there’s an id on the page that is matched to the URI part after “#” the :target CSS pseudo class points to the element with this id. This allows us to detect when the “Menu” button is clicked.
4. When the small screen is detected (with help of CSS media queries), the menu is hidden and the “Menu” button is shown. Also, the display property is changed from inline-block to block for menu items for responsive.



1. With the help of :target the “Menu” button click is detected and based on :target the button’s hidden and the menu’s shown.



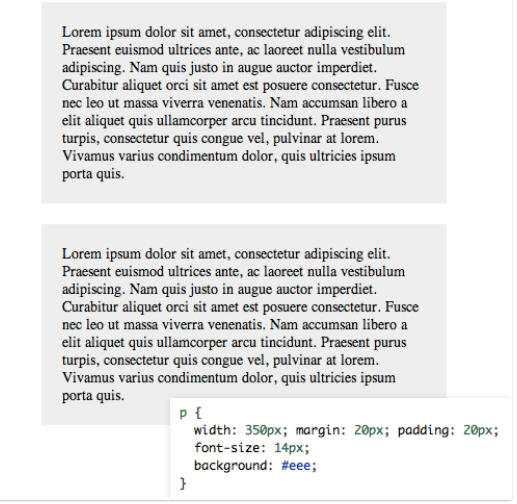




The Display Property in particular inline-block

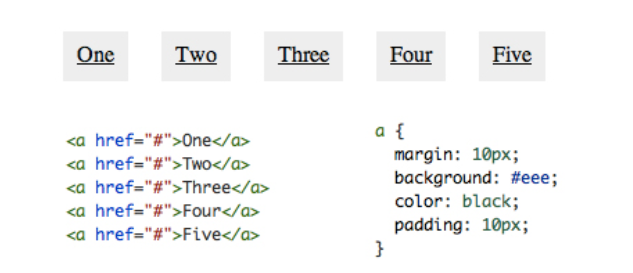
Web browsers render different elements in different ways. Some elements are block-level, meaning that their default display value is set to block. Block-level elements have a definable width and height and automatically create a new row in the layout as they’re created.

One example of a block-level element is a paragraph. If I put two paragraphs into a HTML document, they stack on top of each other rather than appearing side by side.



Other elements have their display value set to inline by default. This means they don’t have a definable height and width and will not create new rows in a layout (thus they appear “inline”). Inline elements are generally the type of thing that you institute within a paragraph or other block-level element: strong, em, anchor, etc.

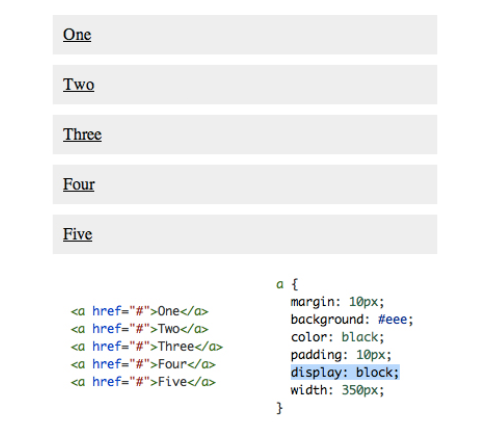
Here’s what happens if I put a bunch of anchor tags into an HTML document. Unlike the paragraphs above, which automatically stack, these elements are created side by side. Their width and height are determined by the content they contain.



Setting Your Own Display Value

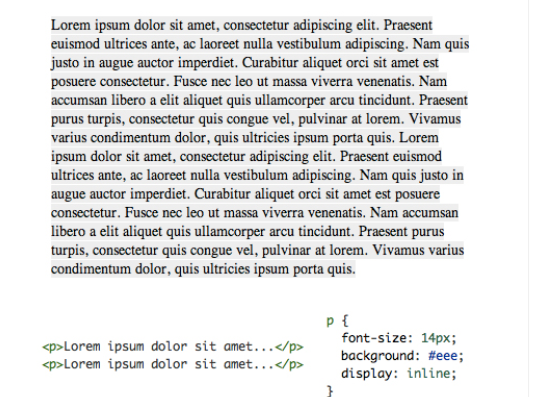
One really interesting feature of CSS is the ability to change the default display behavior of elements. Though a standard behavior is already set for a given element, we can override that for our own purposes.

For instance, we can easily take those inline anchor tags from the second example above and make them act more like the block-level paragraphs from the first example. We do this be setting the display property to block in our CSS.



Now our anchor tags behave like block-level items, so each one creates a new line and can have a custom height/width value applied.

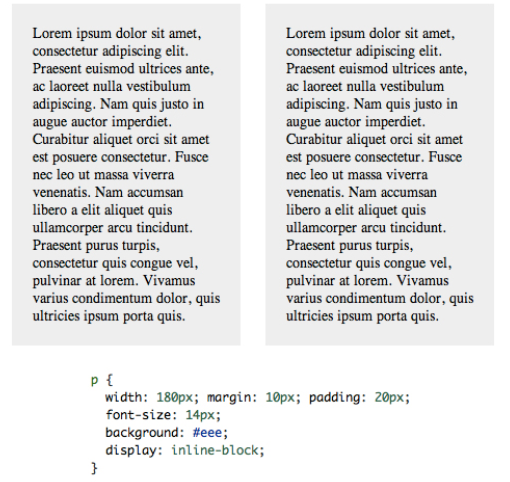
As you can imagine, it’s more difficult to work with the reverse of this example by taking a block-level element and setting the display value to inline. Instinctively, you might use display: inline to try to make the two paragraphs above appear side by side, but when you do this, the result is instead that the two flow together into a single paragraph.



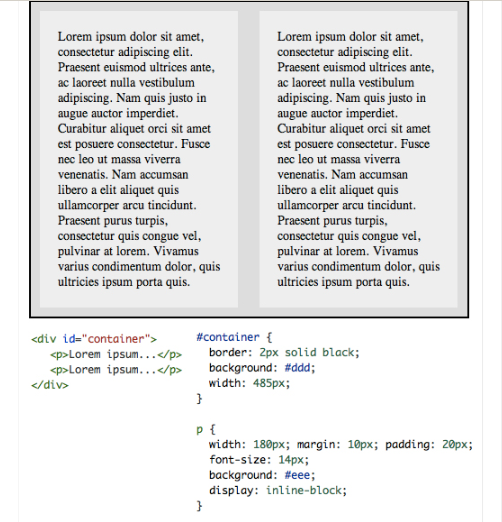
## display: inline-block

There are tons of values for the display property beyond what we’ve mentioned already, but by far one of the most interesting and useful of the bunch: inline-block.

Watch what happens when we take our two paragraphs from the original example above and apply a display value of inline-block.

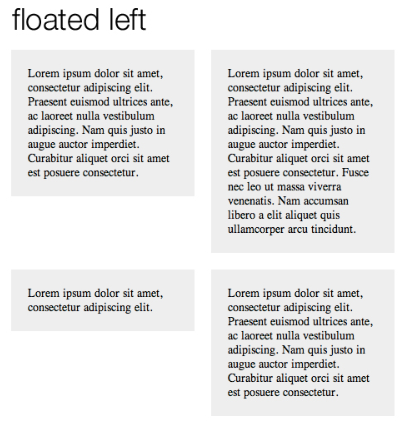


Looks a lot like a float right? So what happens if we add in a parent container? Does it have the collapsing problem that we saw with floats? Nope! Everything works just how we expect it to.

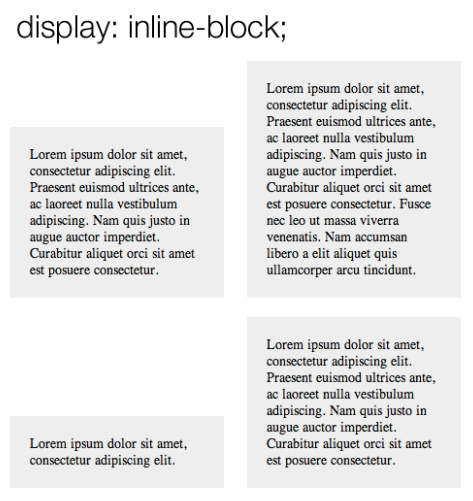


What’s happening here is that we’re telling the browser to display the paragraphs inline, but allow them to retain their block-level characteristics. This means we can set a width and height manually and have the two elements remain distinct, but also have them appear next to each other in the document flow.

One of the first problems that you will encounter between floats and inline-block is when you have several elements with varying heights. For instance, here’s what happens when you float a bunch of paragraphs to the left:



Now here’s what happens when you use inline-block to pull off the same thing. Notice that the bottom edges of the paragraphs are aligned instead of the top as in the previous image.

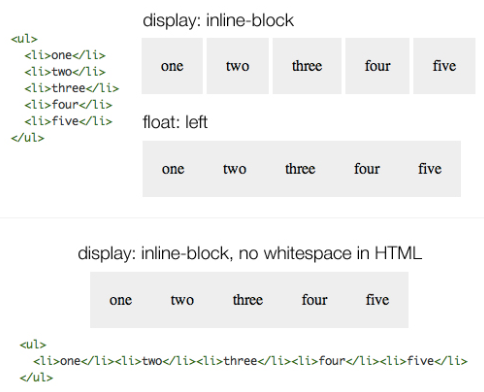


Fortunately, this isn’t a huge problem. To address the issue, make sure you set the vertical-align property to top.



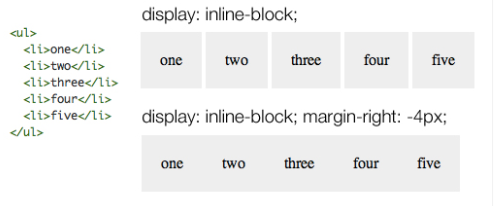
## The Whitespace Issue

There’s another important place that the behavior of inline-block differs from floats. I find it bizarre that anything in HTML and CSS could be whitespace aware when it comes to layout, but that’s exactly what we find here. Consider the following examples.



Here we can see that when a group of list-items is floated, they smash right up against each other like we would expect, allowing us to manually set the gap without any unexpected extra space. However, when we do the same with inline-block, there’s a little bit of default space that won’t even go away if we set our margins to 0.

As you can see, one solution is to take out the whitespace in our HTML and push the elements right up against each other. Once again, I find this pretty confusing but it does work. An alternative solution that produces the same result without messing with the visual hierarchy in your HTML is to apply a margin of -4px on the list items.



Browser Support for inline-block



## The :target pseudo-class:

## The :target pseudo-class allows us to style elements on the page based on the URL. If the URL has an identifier (that follows an #), then the :target pseudo-class will style the element that shares the ID with the identifier.

## Put another way - URLs with an # followed by an anchor name, link to a certain element within a document. The element being linked to is the target element. The :target selector can be used to style the current active target element.

<style>:target{border: 2px solid #D4D4D4;background-color: #e5eecc; }</style></head>

<body><h1>This is a heading</h1><p><a href="#news1">Jump to New content 1</a></p><p><a href="#news2">Jump to New content 2</a></p><p>Click on the links above and the :target selector highlight the current active HTML anchor.</p>

<p id="news1"><b>New content 1...</b></p>

<p id="news2"><b>New content 2...</b></p>

<p><b>Note:</b> Internet Explorer 8 and earlier versions do not support the :target selector.</p></body>