# 

# 

# 

# HTML and CSS

# 

# 

# CONTENTS

[**HTML and CSS**](#_gjdgxs) **1**

[TAGS](#_sqcjg6utzbcu) 1

[**What is CSS?**](#_jqt0mq2zaa6y) **3**

[CSS Elements](#_5ohi8gjo6psw) 3

[**960 GRID SYSTEM**](#_2rlaxeg2xlto) **5**

[Push Me, Pull Me](#_27ln6zqeq4yc) 11

[Before pushing/pulling:](#_bxb5gal2f49c) 12

[Wide Open Spaces](#_h914vdcucarc) 13

# 

# HTML and CSS

HTML is the standard markup language for creating Web pages.

1. HTML stands for Hyper Text Markup Language
2. HTML describes the structure of Web pages using markup););
3. HTML elements are the building blocks of HTML pages
4. HTML elements are represented by tags
5. HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
6. Browsers do not display the HTML tags, but use them to render the content of the page

## TAGS

|  |  |
| --- | --- |
| <!DOCTYPE> | Defines the document type |
| <html> | Defines an HTML document |
| <head> | Defines information about the document |
| <title> | Defines a title for the document |
| <body> | Defines the document's body |
| <h1> to <h6> | Defines HTML headings |
| <p> | Defines a paragraph |
| <br> | Inserts a single line break |
| <hr> | Defines a thematic change in the content |
| <!--...--> | Defines a comment |
| <b> | Defines bold text |
| <blockquote> | Defines a section that is quoted from another source |
| <em> | Defines emphasized text . |
| <i> | Defines a part of text in italic font |
| <mark> | Defines marked/highlighted text |
| <pre> | Defines preformatted text |
| <strong> | Defines important text |
| <sub> | Defines subscripted text |
| <sup> | Defines superscripted text |
| <u> | Defines text that should be stylistically different from normal text |
| <form> | Defines an HTML form for user input |
| <input> | Defines an input control |
| <textarea> | Defines a multiline input control (text area) |
| <button> | Defines a clickable button |
| <select> | Defines a drop-down list |
| <optgroup> | Defines a group of related options in a drop-down list |
| <option> | Defines an option in a drop-down list |
| <label> | Defines a label for an <input> element |
| <fieldset> | Groups related elements in a form |
| <legend> | Defines a caption for a <fieldset> element |
| <datalist> | Specifies a list of pre-defined options for input controls |
| <output> | Defines the result of a calculation |
| <iframe> | Defines an inline frame |
| <img> | Defines an image |
| <area> | Defines an area inside an image-map |
| <figcaption> | Defines a caption for a <figure> element |
| <figure> | Specifies self-contained content |
| <audio> | Defines sound content |
| <video> | Defines a video or movie |
| <a> | Defines a hyperlink |
| <ul> | Defines an unordered list |
| <ol> | Defines an ordered list |
| <li> | Defines a list item |
| <dl> | Defines a description list |
| <dt> | Defines a term/name in a description list |
| <dd> | Defines a description of a term/name in a description list |
| <table> | Defines a table |
| <caption> | Defines a table caption |
| <th> | Defines a header cell in a table |
| <tr> | Defines a row in a table |
| <td> | Defines a cell in a table |
| <col> | Specifies column properties for each column within a <colgroup> element |
| <div> | Defines a section in a document |
| <span> | Defines a section in a document |
| <header> | Defines a header for a document or section |
| <footer> | Defines a footer for a document or section |
| <aside> | Defines content aside from the page content |
| <meta> | Defines metadata about an HTML document |

## New Semantic Elements in HTML5

# Many web sites contain HTML code like: <div id="nav"> <div class="header"> <div id="footer"> to indicate navigation, header, and footer. HTML5 offers new semantic elements to define different parts of a web page:

# 

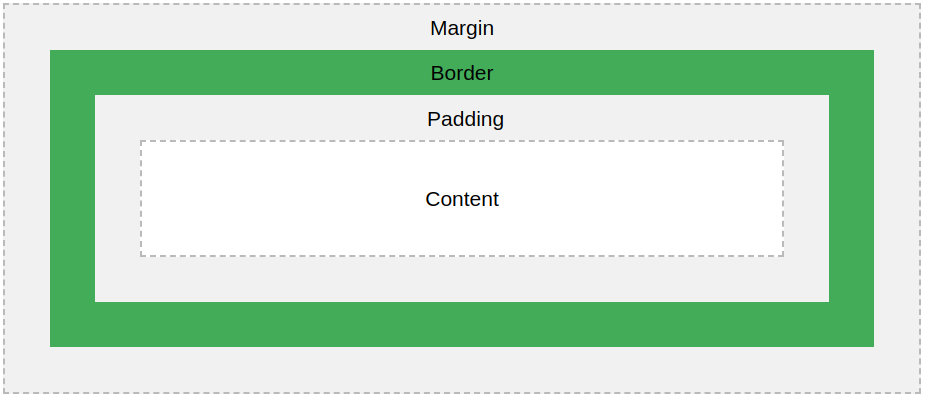
# What is CSS?

CSS stands for Cascading Style Sheets.CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External stylesheets are stored in CSS files.

## The CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:



Explanation of the different parts:

* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

## 

## CSS Elements

|  |  |
| --- | --- |
| align-content | Specifies the alignment between the lines inside a flexible container when the items do not use all available space |
| align-items | Specifies the alignment for items inside a flexible container |
| align-self | Specifies the alignment for selected items inside a flexible container |
| background | A shorthand property for setting all the background properties in one declaration |
| background-color | Specifies the background color of an element |
| background-image | Specifies one or more background images for an element |
| background-position | Specifies the position of a background image |
| background-repeat | Sets if/how a background image will be repeated |
| border | A shorthand property for border-width, border-style and border-color |
| caption-side | Specifies the placement of a table caption |
| color | Sets the color of text |
| empty-cells | Specifies whether or not to display borders and background on empty cells in a table |
| float | Specifies whether or not a box should float |
| font | Sets all the font properties in one declaration |
| font-family | Specifies the font family for text |
| font-size | Specifies the font size of text |
| font-size-adjust | Preserves the readability of text when font fallback occurs |
| font-stretch | Selects a normal, condensed, or expanded face from a font family |
| font-style | Specifies the font style for text |
| font-weight | Specifies the weight of a font |
| height | Sets the height of an element |
| justify-content | Specifies the alignment between the items inside a flexible container when the items do not use all available space |
| left | Specifies the left position of a positioned element |
| letter-spacing | Increases or decreases the space between characters in a text |
| line-break | Specifies how/if to break lines |
| margin-bottom | Sets the bottom margin of an element |
| margin-left | Sets the left margin of an element |
| margin-right | Sets the right margin of an element |
| margin-top | Sets the top margin of an element |
| max-height | Sets the maximum height of an element |
| max-width | Sets the maximum width of an element |
| min-height | Sets the minimum height of an element |
| min-width | Sets the minimum width of an element |
| padding | Sets all the padding properties in one declaration |
| padding-bottom | Sets the bottom padding of an element |
| padding-left | Sets the left padding of an element |
| padding-right | Sets the right padding of an element |
| padding-top | Sets the top padding of an element |
| position | Specifies the type of positioning method used for an element (static, relative, absolute or fixed) |
| text-align | Specifies the horizontal alignment of text |
| text-align-last | Describes how the last line of a block or a line right before a forced line break is aligned when text-align is "justify" |
| text-justify | Specifies the justification method used when text-align is "justify" |
| vertical-align | Sets the vertical alignment of an element |
| width | Sets the width of an element |
| z-index | Sets the stack order of a positioned element |

# 

# 

# 960 GRID SYSTEM:-

The first time I discovered the 960 Grid System, I was immediately excited about the possibilities of implementing complex layouts so easily.However, since I was fairly new to web design at the time, when I downloaded the files, I quickly became overwhelmed at how complicated the whole thing seemed.  
With all this code, how could this be the *easy way* to create a layout? This article is for web designers and front-end web developers who are interested in [**grid-based layout systems**](https://www.webpagefx.com/blog/web-design/a-brief-look-at-grid-based-layouts-in-web-design/) but are at a loss on how to decipher them.We’ll focus specifically on the [960 Grid System](http://960.gs/), but after reading this guide, you’ll find that most of the other grid systems out there are similar and will make much more sense after you understand a few basic principles.

Grid-Based Design

Before we get into the specifics of the 960 Grid System, let’s briefly discuss grid-based design in general. The idea is certainly not something that originated on the Web. In fact, it stems from one of the oldest and most basic design principles:[alignment](https://www.webpagefx.com/blog/web-design/the-brads-alignment-in-design/).Our brains like to simplify things to make them readily understandable. This is why we try to impose order on things that seem chaotic, like seeing a face in the craters on the moon.

Naturally, the easier it is to impose order, the quicker our brains can identify a pattern and move on. Grids are so organized and orderly that they require almost no interpretation our part.Consider the two page layouts represented in the image below.



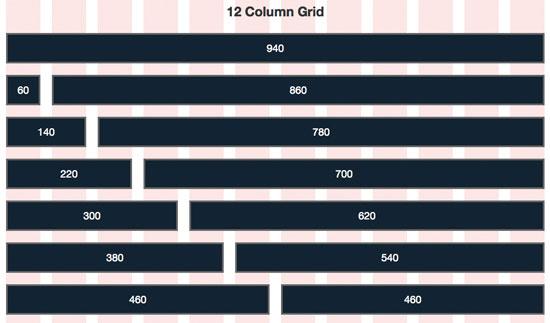
Though both of these images are simply a gathering of rectangles, the layout at the top seems fundamentally better than the one on the bottom. We can instantly recognize a pattern, accept it, and move on. The image on the bottom, however, is visually unsettling by comparison. There’s no clear pattern, order, or goal–it’s just looks like a random assortment of shapes.Our eyes have a tendency to frantically search for fractions of a second while we attempt to identify a trend, which increases the time necessary to take in the scene as a whole.  
It’s interesting to note that random can still be beautiful. Random definitely has its place in nature, art, and even design, but it’s no way to logically organize information.  
The point is that grids are among the simplest and strongest ways to create order on a page.

Why Do I Need a Grid System?  
The 960 Grid System–and other tools and systems like it–provide a fast and easy way to create grid-based layouts using CSS. They do this by providing cross-browser-tested and optimized preset column widths for you to set your content into.  These grids can help you save on your [online marketing budget](https://www.webpagefx.com/marketing-calculator.html).  
Before [CSS3](https://www.webpagefx.com/blog/web-design/css3-techniques-you-should-know/), it wasn’t exactly easy to break up a page into columns without getting into tedious math.For instance, if you have a 1,000-pixel wide container and you want to split it up into three columns, that’s 333 and 1/3 pixel per column (not exactly a nice whole number). Further, columns broken up like this would crash into each other, so a margin must be added on each side. If we add a 10-pixel margin to each side of every column, we must also subtract that 20 pixels from the width of each column. This gives us 3 columns roughly 313 pixels wide each with a margin of 10 pixels on each side (even then you’re at 999px and not 1,000px).  
Want 4 columns in a row below that? Then you have to start the process over and subtract 80px of margin from 1,000px for a total of 920px and divide that by 4 to get a column width of 230px.  
Finally, if you want to add a sidebar that’s a third of the total width of the page, you have to create a column that’s 750px for the content and one that is 250px for the sidebar, then subtract 40px of margin to get one 730px column and one 230px column.

The 960 Grid System  
The 960 Grid System is simply a way to lay out websites using a grid that is 960 pixels wide.

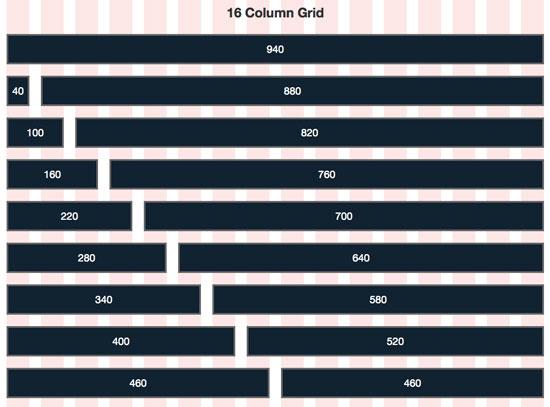


The reason it’s 960 pixels wide is because the number 960 makes for a lot of clean divisions utilizing whole numbers when factoring in column widths and margins. And it fits nicely on the majority of screens.The 960 GS comes in two primary variants: a 12-column grid and a 16-column grid (a 24-column version is included as well for those that really need extra control). In the 12-column version, the narrowest column is 60 pixels wide. Each column after that increases by 80 pixels. So the available column widths are: 60, 140, 220, 300, 380, 460, 540, 620, 700, 780, 860 and 940.



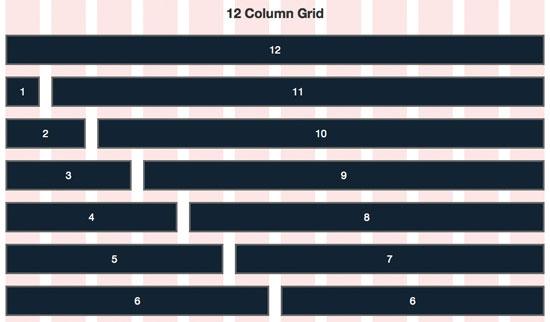
Similarly, in the 16-column version, the narrowest column is 40 pixels wide and each column after that increases by 60 pixels.

So the available column widths are: 40, 100, 160, 220, 280, 340, 400, 460, 520, 580, 640, 700, 760, 820, 880 and 940.

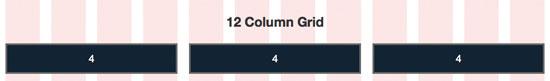


CSS Classes Now in Session

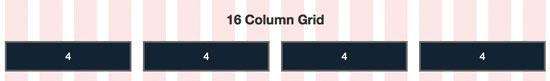
When you look at the diagrams above, consider each of the dark blue horizontal bars as a CSS class in the 960 Grid System.To create an object in your layout that is the width of one of those bars, you simply assign the proper class to your div–that’s it!  
The classes are conveniently named according to their size with the grid\_1 CSS class being the narrowest and grid\_12CSS class being the widest (grid\_16 is the widest in the 16-column version). So to reuse our image from above, take a look at the available columns now, but this time, think about them using their respective CSS classes instead of pixel widths.



This naming system makes it incredibly easy to hash out complicated layouts in seconds flat. To fill a page’s width, the trick to keep in mind is that the numbers assigned to your selected classes must equal 12 in the 12-column version and 16 in the 16-column version.For instance, using the 12-column version, if you have 3 divs of text that you want to be displayed side-by-side in a 3-column layout, simply assign the grid\_4 class to each one to total 12 (4+4+4=12).



Similarly, assigning the grid\_4 CSS class in the 16-column version makes it easy to create a 4-column layout (4+4+4+4=16).



To make sure you’re referring to the proper classes, be sure to place your 12-column elements inside a div with the class container\_12 and your 16-column classes inside a div with the class called container\_16.

If you’ve never worked with the 96 GS before, I hope you’re having your “aha” moment right now regarding just how easy it is to spec out a layout in no time at all using this system.

## Push Me, Pull Me

The 960 Grid System allows you to reposition elements independently by pushing or pulling them horizontally along the page. This is accomplished by using the push and pull CSS classes.

For instance, consider the two examples in the image below. The first example is a basic 4-column layout using only the grid\_3 class.

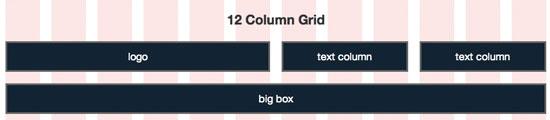
However, in the second version, I’ve pushed the first column and pulled the last column, resulting in their positions jumping over one column from where they would normally lie in the layout.



Keep in mind that you can push items as far as you want. If I had wanted to push an element two columns over, I would’ve implemented the class push\_2, and so on.

The push/pull system has major implications for how you lay out a page in your HTML document.For instance, in the example below, imagine the website’s name is typed out in a logo and placed as the first element on the page.  
As the most important element on the page, you’d like to keep the logo as the first item in your HTML markup, but visually, you actually want it to appear in the center of the page.

## Before pushing/pulling:



To visually position the logo element in between the two text columns, you would use the following HTML:

<div class="grid\_6 push\_3"><p>logo</p>

</div>

<div class="grid\_3 pull\_6"><p>text column</p>

</div>

<div class="grid\_3"><p>text column</p>

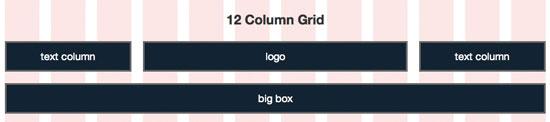
</div>

<div class="grid\_12"><p>big box</p>

</div>

This results in the layout shown in the image below.

After pushing/pulling:

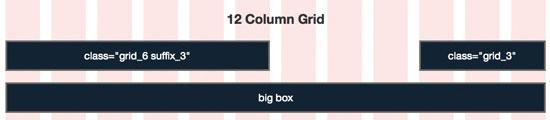


Despite the fact that the logo comes first in our markup, it will be visually positioned in the center of our page.

## Wide Open Spaces

You’ll often find that you want to create empty space in a layout ([negative space](https://www.webpagefx.com/blog/web-design/negative-space-in-webpage-layouts-a-guide/) is a good design device). To accomplish this, apply the prefix and suffix classes to your divs. These are implemented very similar to the push and pull classes.

For instance, to leave a blank space that is the width of one column before an element, use prefix\_1 class, or after an element using suffix\_1 class.



As you can see, the example above uses a suffix\_3 class to create an empty space the width of three columns after it.

# 