Introduction to CSS

Luka Abrus
Technology Specialist, Microsoft Croatia

CSS, or Cascading Style Sheets, have brought a completely new view on Web page design and development. Using CSS you can completely separate text displayed on a Web page (which is created in HTML code) and information that describes how to display and present that text (which is defined using CSS).

CSS has been introduced to solve problems and help you save time, while giving you more possibilities in designing the way your Web pages look. Although this might be the first time you've heard about CSS, you've already seen it in action many times before. Here's one typical example: some Web pages highlight their links in a specific way. They are in a different color than the rest of the text, but if you move the mouse over them, they change color or become underlined. That has been done without touching HTML code, but rather with using style definitions. We'll cover such an example later in this guide.

To be able to follow this guide, you need to have some prior knowledge of HTML. We will use HTML code as the basis and then build on it showing you what other capabilities you have when displaying and presenting page content.

What is CSS?

Since the beginning of HTML usage for web page creation, people have realized the need to separate the way the page looks and the actual content it displays. Even the first versions of HTML have supported different ways to present text using **FONT**, **B** (bold) or **I** (italic) tags. Those HTML elements still exist today, but their capabilities are far below what Web pages should provide.

As we've already said, CSS enables you to separate the layout of the Web page from its content. This is important because you may want the content of your web page to change frequently (for example, a current events page) but not the design/layout, or vice versa. It is a standard of the World Wide Web Consortium (W3C), which is an international Web standards consortium.

Practically, all the style and layout guidelines for a website are kept in CSS files that are separate from the HTML files which contain the data, text and content for a website. Simply put, when talking about displaying Web pages in the browser, HTML answers the question "What?", while CSS answers "How?".

When using CSS, you are defining how to display each element of the page. You can, for example, say to show all text in **DIV** elements in blue color, to have all links italic and bold, etc. With CSS you can also define classes, which tell the browser how to display all elements of that class.

Maybe you're asking yourself, why bother with CSS? Isn't it much simpler and faster to define everything inside the HTML page? Using HTML tags and attributes, you can modify the style of each element on your page.

But what if you have a Web site with a larger number of pages, let's say 50? Imagine the process of setting the style for each element on your 50 pages. And then, if later on down the road you want to change the font style, you'll have to manually go through each file and change all the HTML elements. You can count on a very long, boring and tiring process!

With CSS you can put all the information about displaying HTML elements in a separate page. Then you can simply connect this CSS file with all pages of your Web site, and voilà – all the pages will follow the same guidelines. Change the CSS file, and you have indirectly changed all pages of your Web site. In addition, you get much greater design capabilities with CSS, as we will show in this guide.

How do I use CSS?

Let's get started with using style sheets. CSS data is actually plain text written in a specific way. Let's take a look at the contents of a sample CSS file:

```
body
{
   font-family: Verdana;
   font-size: 9pt;
   text-align: right;
}
div
{
   font-family: Georgia;
}
.important
{
   background-color: #ffffde;
   border: thin black ridge;
   font-family: Franklin Gothic Book;
}
```

It is actually completely readable – this style sheet defines that all content within the HTML **BODY** element will use font Verdana with size of 9 points and will align it to the right. But, if there's a **DIV** element, the text within that will be written in font Georgia. We're also using a class named "important" (classes use "." notation, which we will cover later on). All elements of this class will have a set background color, a border and will use Franklin Gothic Book font. As you see, style definitions for a certain element or class are written inside curly braces ("{ }") and each line ends with a semicolon ";".

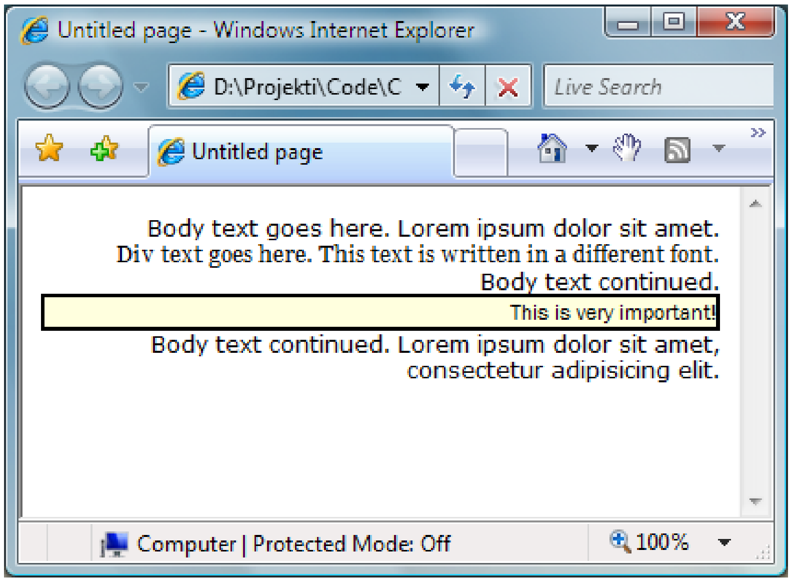
Now is the perfect time to explain the scoping of styles. All CSS definitions are inheritable – if you define a style for **BODY** element, it will be applied to all of its children, like **P**, **DIV**, or **SPAN** elements. But, if you define a style for **DIV** element, it will override all styles from its parent. So, in this case, the **DIV** element text would use font Georgia size 9 points and would be aligned to the right. As you see, **DIV** style definition for the font family has overridden **BODY** style definitions.

This goes on – if you have a **DIV** element which is also of class "important", the class definition will override **DIV** style definitions. In this case, such **DIV** element would have a background color set, a border, it would use font Franklin Gothic Book size 9 points and be aligned to the right.

Here are the elements that would be affected by the sample CSS file.

```
<html>
...
<body>
Body text goes here. Lorem ipsum dolor sit amet.
<div>Div text goes here. This text is written in a different font.</div>
Body text continued.
<div class="important">This is very important!</div>
Body text continued. Lorem ipsum dolor sit amet, consectetur adipisicing
elit.
</body>
</html>
```

And, of course, the browser would show this content as follows.

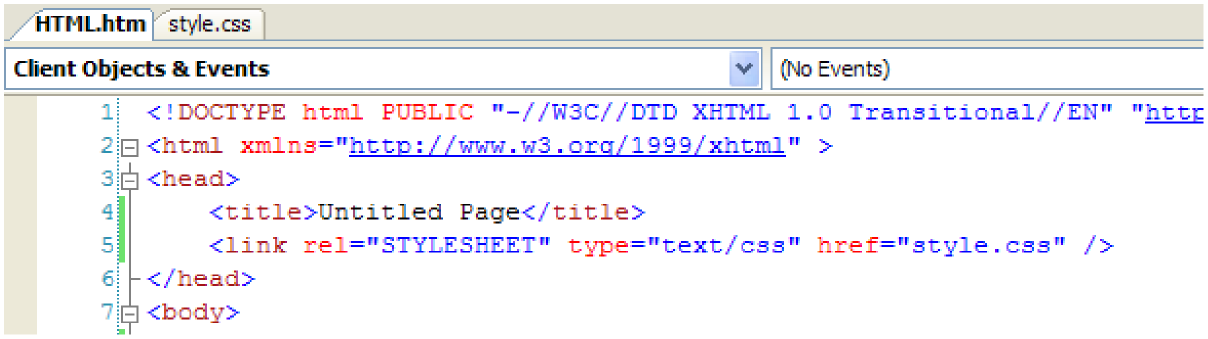

Untitled page - Windows Internet Explorer
D:\Projekti\Code\C
Live Search
Untitled page
Computer | Protected Mode: Off
100%

Body text goes here. Lorem ipsum dolor sit amet.
Div text goes here. This text is written in a different font.
Body text continued.
This is very important!
Body text continued. Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Let's go to the next step – how to connect a CSS file with an HTML page. Go ahead and open Microsoft Visual Web Developer 2005 Express Edition (you can download and install it from http://msdn.com/express/vwd). Select File | New File and choose "Style Sheet" from among the Visual Studio installed templates. Copy and paste the CSS sample above into this file and save this file as "style.css" into a folder on your computer. Now select File | New File and choose "HTML Page". Also save this HTML page into the same folder on your computer. Insert the following code into the HTML page.

```
<link rel="STYLESHEET" type="text/css" href="style.css" />
```

This code should be put within the HTML page header, within **HEAD** element. As you see, *href* attribute defines which CSS file to use. Put this **LINK** element within all HTML pages you wish to apply styles to and you're done!



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Untitled Page</title>
    <link rel="STYLESHEET" type="text/css" href="style.css" />
</head>
<body>
```

CSS data doesn't necessarily have to be in a separate file. You can define CSS styles inside of a HTML page. In this case, all CSS definitions have to be inside a **STYLE** element. This approach can be used to define the looks of elements that are specific to a certain page and will not be reused by other pages. Take a look at how that HTML page might look:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>My title</title>
      <style type="text/css">
        body
        {
            font-family: Verdana;
            font-size: 9pt;
            text-align: right;
        }
        div
        {
            font-family: Georgia;
        }
        .important
        {
            background-color: #ffffde;
            border: thin black ridge;
            font-family: Franklin Gothic Book;
        }
      </style>
   </head>
   <body>
       <div class="important">My content</div>
   </body>
</html>
```

Notice that in this example you can see how to define an element of a specific class – just add *class* attribute and set its value. All classes within CSS style definitions are prefixed with a dot (".").

The third way to define a CSS style, in addition to the previously explained methods of a separate CSS file, and the **STYLE** element within the HTML page header, is inside of a specific HTML element. To do this, you need to use the *style* attribute. Take a look at the following example:

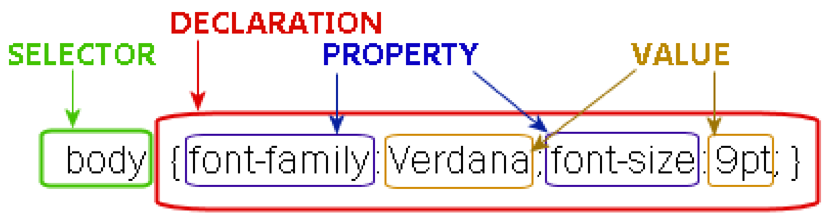
```
<span style="font-family: Tahoma; font-size: 12pt;">My text</span>
```

As you're probably guessing, all the text inside of this **SPAN** element will be displayed using 12 point Tahoma font. And remember – when applying styles directly to elements, as in this last example, these style definitions will override all element definitions and class definitions previously set in a separate CSS file or inside of HTML page header **STYLE** element.

CSS style definition syntax

To be able to write CSS files and definitions correctly, you need to remember few simple rules. Although CSS syntax is rather logical and easy to learn, there are 6 basic things you need to know. First, take a look at the structure of a style definition.

And here are 6 rules of style definitions:



1. Every CSS definition has to have a selector and a declaration. The declaration follows the selector and uses curly braces.
2. The declaration consists of one or more properties separated with a semicolon.

3. Every property has a name, colon and a value.
4. A property can have multiple values separated with a comma (e.g. "Verdana, Arial, Franklin Gothic Book").
5. Along with a value, can also be a unit of measure (e.g. "9pt", where "pt" stands for *points*). No space is allowed between the value and the unit.
6. When writing CSS code, you can use whitespaces as needed – new lines, spaces, whatever makes your code more readable.

Quiz 1

Question 1.1: What three options do you have if you want your Web page use CSS? How can you use CSS?

Answer 1.1:

- You can put your CSS code in a separate file and link it to a Web page using the **LINK** element.
- You can put your CSS code inside a **STYLE** element in the header of the HMTL page.
- Or you can put your CSS code inside of *style* property or attribute of the specific HTML element you wish to apply the style to.

Question 1.2: What is wrong with the following CSS definition?

```
    body (     color = "black";    )
```

Answer 1.2: Usage of whitespaces is allowed, but the problem is in using parentheses (instead of curly braces) and the equal sign (instead of a colon). Correct code should look like this:

```
    body {     color: "black";    }
```

Question 1.3: Which CSS code would be applied to an element, the one defined within the **STYLE** element of the page or the one defined within the *style* property of the element? Take the following code as an example. What would be the size of the font in the **DIV** element?

```
<html>
   <head>
      <title>My page</title>
      <style type="text/css">
      div
      {
         font-family: Verdana;
```

```
          font-size: 9pt;
       }
       </style>
    </head>
    <body>
       <div style="font-weight: bold; font-size: 12pt;">My text</div>
    </body>
</html>
```

Answer 1.3: Both would be applied, but the one defined within the *style* property would be completely applied and it might override the code defined within **STYLE** element of the page. **DIV** element would be styled as if it had the following style (the font-size would be overridden by value in the *style* property):

```
font-family: Verdana; font-weight: bold; font-size: 12pt;
```

Selecting elements to describe

Let's get started with real CSS examples. When writing CSS style definitions, the first thing you need to do is to select elements to apply styles to. If you are writing CSS code inside the *style* attribute of an element, you don't have to bother – that style will be applied to that element (and its children) only.

But when using a separate CSS file, or internal style elements inside an HTML document, selecting the elements is crucial. You can select any HTML element from the page to apply a style to, but the capabilities of CSS go far beyond just the elements on the page.

Basic HTML elements

The simplest way to select which elements to apply styles to, is to specify the HTML elements themselves. For example, selector *b* would define the style for all **B** elements on the page, and selector *p* would define the style for all **P** elements on the page. To write a HTML element selector, just write the tag name without "<" and ">" signs. The CSS code below defines styles for **H1** and **DIV** HTML elements.

```
h1 { font-size: 16pt; font-weight: bold; }
div { border: solid 1px black; }
```

Remember: when you define a style of HTML element, it will be applied to all such elements on the page.

Besides this simple selecting of HTML elements, you can use another technique – contextual selectors. Contextual selectors, as their name implies, select elements based on their context. They can be applied to all types of selectors - HTML elements, classes and all others you'll learn as you go forward with this guide.

So if you want to apply styles to all **B** elements that are part of **DIV** elements, you would write:

```
div b { font-family: Times New Roman; }
```

This style would be applied to every **B** element that is a child of a **DIV** element, for example:

```
<div>This is my text <b>Times New Roman font</b> This is my text</div>
```

With contextual selectors you can go as deep as you want. You can write for example *div b i u { … }*, which would define the style for all **U** elements which are inside **I** elements which are inside **B** elements which are inside **DIV** elements. You get the picture!

Let's look at one practical example. As you know, all pictures that are also links by default have a border. You can change that by adding the *border="0"* attribute to all **IMG** elements, or you can use a contextual selector. You just need to select all pictures (**IMG** elements) that are part of links (**A** elements).

```
a img { border: none; }
```

There is one more situation we need to explain which might make your CSS development a bit easier. What if you want to apply the same style both to all **DIV** and all **SPAN** elements? You could write the same style twice, or only once using the following syntax:

```
div, span { font-family: Segoe UI, Verdana; }
```

Just use a comma to separate selectors with the same style. If you don't use a comma, this style would be applied to all **SPAN** elements that are within **DIV** elements (contextual selector). The comma means that the style will be applied both to **SPAN** and **DIV** elements. Separating selectors with a comma can be used for any selector, whether it is a class, ID selector or a pseudo class, as you will soon learn.

CSS classes

Controlling the way all HTML elements look can be useful, but also limiting. It's excellent to be able to change every paragraph, table cell or image with one line of CSS code, but sometimes you'll want to change only few paragraphs or images, not all of them. You can add CSS code through the *style* attribute of each element, but for more elements that method gets too complicated.

The answer is, as you've seen before, to use classes. There are two types of classes – generic classes that can be applied to any element, and classes that can be applied only to a certain type of HTML element.

Let's start with generic classes. Their selector starts with a dot ("."), which states that it is a class. You can name it anything you want.

```
.important { background-color: #FFFFDE; }

.myclass1 { font-family: Verdana; }

.boooring { color: Gray; }
```

To apply a class to a certain HTML element, use its *class* attribute where you state the class name without the dot.

```
<div class="myclass1">sdfs</div>
```

```
<i class="boooring">italic</i>
```

In this example, the contents of **DIV** element would use Verdana font and the contents of **I** element would be written in gray color.

You can also use classes which can be applied only to certain HTML elements. Selectors of these classes start with the HTML element name, followed with the dot and the class name.

```
div.big { font-weight: bold; font-size: 16pt; }
```

These classes can be applied only to a specified element, in this case a **DIV** element.

```
<div class="big">Big bold text.</div>

<span class="big">Normal text - class not applied.</span>
```

So now you know how to apply styles to group of elements, but how do you apply a style only to one element on the page? You can write everything in its *style* attribute, but there is another way – using ID selectors. The syntax of ID selectors is very similar to classes, but instead of a dot you must use a hash sign ("#").

```
#title { font-size: 22pt; }
```

This style can be applied only to the element which has ID "title". As you know, ID's uniquely define elements on a page and are specified in the *ID* attribute of an HTML element.

```
<div id="title">My Title</div>
```

The ID selector name must start with a letter, and can only use letters and numbers. For example, "#3rdTitle", "#My_Title" and "#Important!" isn't correct.

Links

The most common usage of CSS is to define how hyperlinks look. With the selectors you've seen so far you can only define the general design of links. But links can have special states. You can apply different styles to them, depending on the position of user's mouse and whether they have been clicked. For example, they can be underlined when the mouse is over them, gray when the user has already visited them or glowing in yellow when user clicks them.

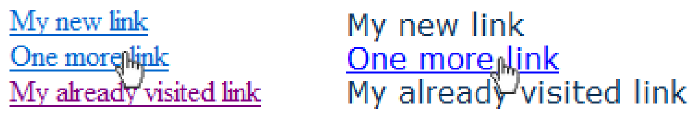
This effect is accomplished with pseudo classes. Here are the four states that links can be in:

Pseudo class	**Link state**
a:link	Normal link
a:visited	Already visited link
a:hover	Mouse hovers the link
a:active	User is clicking on the link

With these pseudo classes you can create link tricks that you've already seen on different Web sites. Look at the example below which defines the different styles for each link state.

```
a:link { color: #123456; text-decoration: none; font-family: Verdana; }
a:visited { color: #123456; text-decoration: none; font-family: Verdana; }
a:hover { color: blue; text-decoration: underline; font-family: Verdana; }
a:active { color: blue; text-decoration: underline; font-family: Verdana;}
```

Take a look at the picture – which one do you like better, the default link colors (left) or the ones you specify (right)?



As you see, a difference occurs when user moves the mouse over the link – it changes color. This is defined with *a:hover* selector. Also, the link has the same color whether it has already been visited or not, which makes it look much more uniform. Now it's up to you to implement this CSS code and adjust it to fit your Web site's colors and fonts!

When writing HTML code for the links, you don't have to write any special code, as the previous CSS example will be applied to all links in the document.

```
<a href="page2.htm">My new link</a>
```

Pseudo classes can be combined with other selectors, too. For example, if you want to have different styles of links, one for the left menu, and a second style for the right menu on the page, you can use two classes (.right and .left) with pseudo classes.

```
a.left:link { font-family: Verdana; ... }
a.left:hover { ... }
...
```

So now you have a special class of links called "left" to which you can apply different styles using pseudo classes. You can read the code above as – for every link which is of class "left" and in a specific state, apply this style. If you want to apply this style to a link, don't forget to define its class:

```
<a href="page2.htm" class="left">My new link</a>
```

With selectors you've just learned that you can accomplish almost anything. There are actually few more types of selectors and ways to select elements on the page, but with the ones you've just learned, you're good to go. The next step is something we've already been using, but haven't thoroughly explained – CSS properties.

Quiz 2

Question 2.1: If you want all links and **DIV** elements use Verdana font, how would you write the shortest CSS code?

Answer 2.1: Use selectors separated with comma.

```
a, div { font-family: Verdana; }
```

Question 2.2: How would you write CSS code to make all elements of class "title" that are part of a **DIV** element use Verdana font? For example:

```
<div>Normal text. <span class="title">Verdana text.</span> Normal text. <b
class="title">Verdana text.</b></div>
```

Answer 2.2: You should write a contextual selector that selects all elements of class "title" within **DIV** elements.

```
div .title { font-family: Verdana; }
```

Question 2.3: How would you make the text of only the following HTML element underlined without changing the HTML code?

```
<div class="content" id="titleContent">My content.</div>
```

Answer 2.3: Add an ID selector:

```
#titleContent { text-decoration: underline; }
```

Properties of elements

Now that you've learned how to use CSS and how to select elements to apply styles to, you need to learn what styles you can actually apply to elements. The purpose of this part of the guide isn't to teach you about all of the properties that exist, but rather to give you an overview of the possibilities that lie before you.

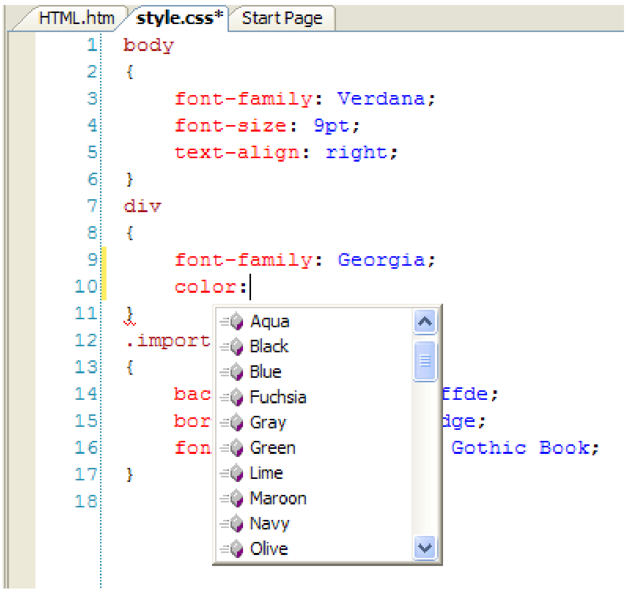
Text properties

Text is the most important part of every web page. The content of the page is the reason why visitors come back, and information sharing is why the internet is so powerful. So it's obvious that you'll have to put a lot into getting good content for your site, and then presenting it in a clear and readable way.

Let's start with the simplest text attribute – its color. For color definitions, the rules are the same as in HTML. Colors can be defined in hexadecimal or by using color names.

```
div { color: black; }

p { color: #00003D; }
```

We won't list all color names here, but you can't go wrong – if you know the name of the color (like "black", "white", "red", "blue", "maroon" or "aqua"), try it out. Visual Web Developer Express will also help you by popping up a context menu of options as you type:



```
body
{
     font-family: Verdana;
     font-size: 9pt;
     text-align: right;
}
div
{
     font-family: Georgia;
     color:
}
.import
{
     bac                                  ffde;
     bor                                  dge;
     fon                                   Gothic Book;
}
```

The next text attribute is its size. Remember one important thing – always specify the unit of measure, as specifying only the size value doesn't mean anything. In the following table you can see a list of available units and their abbreviations which you'll use in CSS.

Unit	**Abbreviation**	**Description**
em	em	Computed font size
ex	ex	Height of lowercase "x"
pica	pc	Picas (1 pica = 12 points)
point	pt	Points (1 point = 1/72 inches)
pixel	px	Pixels on the screen
millimeter	mm	Millimeters.
centimeter	cm	Centimeters
inch	in	Inches (1 inch = 2,54 centimeters)

When specifying text size, use the *font-size* attribute, like in the following examples:

```
div { font-size: 12pt; }
span { font-size: 24px; }
```

The usual text size on Web pages is, when specified in points, somewhere between 9 and 12. That depends on you and the way you want your page to look. But if you have larger blocks of text, don't use a too small a font size, as it will be too difficult to read.

After specifying the text size and color, define the font in which it will be written. The font directly influences the readability of the text. The font type easiest to read is serif type. Serif fonts have ornaments and finishing strokes on each character, which makes letters more distinguished. For example, some serif fonts are Times New Roman, Garamond or Georgia. On the other hand, there are also sans-serif fonts, without ornaments and with plain endings. Examples are Arial, Verdana, Tahoma and Trebuchet.

Usually the sans-serif fonts are much nicer to see and look better on the screen, but it is suggested that, if you have larger blocks of text, you write them in serif fonts, as they are much easier to read.

Also, be careful when selecting unusual fonts to display text. Fonts are installed on the client computer (check Control Panel – Fonts to see which fonts you have installed), and if you are using a non typical font, the visitor of the page might not have that font installed. It all depends on the operating system of the visitor, so be sure to use only the standard, most common fonts, like Arial, Verdana, Times New Roman, etc.

Font is defined with the *font-family* attribute which can have more values separated with a comma. This is useful if you want to use a font that might not be available to all users. For example, font Segoe UI is installed with Windows Vista, but it isn't available to Windows XP users. So you could write the following CSS code:

```
div { font-family: Segoe UI, Arial; }
```

All the text within the **DIV** element would be written in Segoe UI font for Windows Vista users, but in Arial for all visitors not using Windows Vista. The fonts are checked in the order they are specified and the text will be displayed with first available font. If no specified font is available, the text will be displayed using the default font (most often Times New Roman).

You can specify additional attributes which can make the text underlined, **bold** or *italic*. To underline a text, use the *text-decoration* attribute which can, for example, have values "none", "underline" or "line-through". If you want to make text bold, use the *font-weight* attribute which can have values like "normal" or "bold". To make text italic, use the *font-style* attribute and its "italic" value.

```
div { text-decoration: underline; font-style: italic; font-weight: bold; }
```

And finally, if you want to align text, use its *text-align* attribute. Text can be aligned to the left ("left"), right ("right"), it can be centered ("center") or justified ("justify").

```
div { text-align: center; }
```

Now you've learned the typical text attributes, you can combine them to make your text more readable. Just specify all attributes you need:

```
div { font-family: Segoe UI, Verdana; font-size: 14pt; color: Black; text-
decoration: underline; font-style: italic; font-weight: bold; text-align:
center; }
```

As you might have noticed, this is just an example of all the properties. You wouldn't want your text to be italic, bold and underlined at the same time, would you?!

Backgrounds

When using plain HTML without CSS, you can define the background only for the whole document using the **BODY** element. But with CSS you can do it for any HTML element. The property you need to set is *background-color*. You can set its value similar to the *color* attribute, or to colors in HTML.

However, you aren't limited only to colors. You can even set an image to the background of any element with the *background-image* property. Now, before we move forward, let's explain why the combination of these two properties is important.

For example, if you put a dark image in the background of a **DIV** element, you will probably want the font color to be white. But in the case of document still loading or if background image is inaccessible (someone deleted it from the server?), there will be no background, only the text written in white (same as the default background color of the HTML page), which makes it completely unreadable. So the solution is – always define a similar background color and background image which will be displayed behind the text making it easier to read, and it won't disrupt your page design while the background image is still loading.

Let's show how to use these two properties together:

```
div { background-image: url(background.gif); background-color: Black; }
```

The background color property is set in a familiar way, but the background image is rather new. It uses *url()* to specify the address of the image. This is actually a standard way to define external components of CSS, most often images, which will be used on the page. Don't use quotation marks; just put the image location between parentheses. Your image can be in a different folder – you would write `url(my_folder/background.gif)`.

If the background image is smaller than the element it needs to fill, it will be repeated so that the whole element is filled. That is the default behavior, but you can, of course, change it with CSS. Let's start building a simple scenario – you want to have your logo as the background image, but also want it displayed only in the top right part of the page. You also want it fixed, as you want it to be displayed in the top right all the time, even when user scrolls the page.

The first step is to make the background fixed, so it won't move when the user scrolls. The background will seem as a separate layer of the page, completely detached from the content. We need to set the *background-attachment* property to "fixed" (it's default value is "scroll").

```
body { background-attachment: fixed; ... }
```

The second step is to position the background to start from a certain location on the page. If you don't position it, it will start from the top left and fill the rest of the page. But you can set its position using pixels (e.g. 15px), relative position (e.g. 10%) or use descriptions like "bottom", "top", "left", "right" or "center". Let's position the background to start from the top right part of the page using the *background-position* property.

```
body { background-position: right top; ... }
```

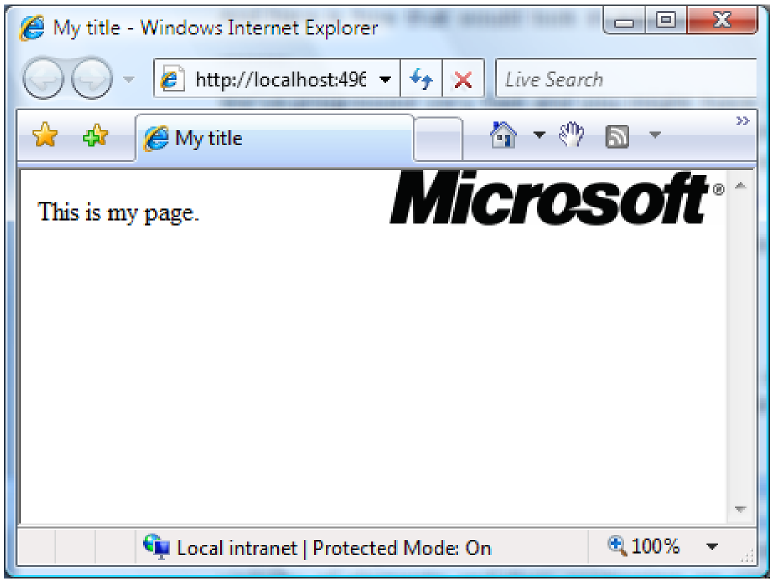
But we're still not done. When positioning the picture in the top right position, it will still be repeated to fill the rest of the page, until it reaches bottom right part of the page. So, the third step is to stop the background from repeating. We will use the property *background-repeat*, which can have value "repeat-x" (repeats the background only in horizontal direction), "repeat-y" (repeats the background only in vertical direction), "no-repeat" (the background isn't repeated and it is only displayed once), or the default value of "repeat".

```
body { background-repeat: no-repeat; ... }
```

And that's it. Let's put it all together and create a CSS declaration that would put a fixed and not repeating background image in the top right part of the page.

```
body { background-image: url(logo.gif); background-color: white;
background-attachment: fixed; background-position: right top; background-
repeat: no-repeat; }
```

And here is how that would look in a browser – our logo is positioned right where we want it to be.


My title - Windows Internet Explorer
http://localhost:496
Live Search
My title
Local intranet | Protected Mode: On
100%

This is my page.

We've progressed very fast and you might have gotten all of these properties mixed up. That's perfectly OK, because you can use a simple *background* property and put all those values in it. Look at the previous example written only using the *background* property.

```
body { background: white url(logo.gif) no-repeat fixed right top; }
```

This will save you from writing large blocks of code and will make your CSS much more readable.

Although we've explained backgrounds using the **BODY** element, the logic is completely the same for any other element, whether it is a **DIV**, a **TABLE** or any other type of element.

Quiz 3

Question 3.1: How can you specify that the text of a **DIV** element should use font Calibri, if it available, or Arial?

Answer 3.1: Specify multiple fonts separated with a comma.

```
div { font-family: Calibri, Arial; }
```

Question 3.2: Can your background image be on a different server? How would you write your CSS code then?

Answer 3.2: Yes, it can be. Just specify it's full URL using *url()*.

```
body { background-image: url(http://www.otherserver.com/logo.gif) }
```

Question 3.3: How would you define a background image which would be displayed filling only the top of the screen?

Answer 3.3: You need to set "repeat-x" value of the *background-repeat* property which specifies that the background will only be repeated horizontally and then position the image in the left top corner of the page.

```
body { background: url(logo.gif) repeat-x fixed left top; }
```

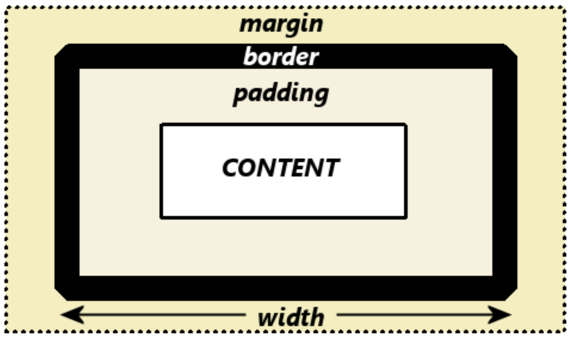
Borders and margins

Let's advance your knowledge of HTML elements. Every element is displayed in a specific way, but there are several properties that might help you display them just the way you want. First, let's take a look at the usual properties of HTML elements, for example a **DIV** element.

Each **DIV** element has a content area in which all of its content, like text or images, is displayed. A **DIV** element can also have a border, which can be of any color and any thickness. Between its border and the content is an area defined by the padding – you can define just how much space you need. If you set the padding to 0, the content will be glued to the border, but if you set it to 5 pixels, the content will be 5 pixels inside of the border.

Outside of the border is an area defined by the margin. It defines how much space must be left between the element and its neighbors. If you set the margin of the **DIV** element to 0, its neighbor element will be right next to the border of the **DIV**. But if you set it to 20 pixels, there will be a space of 20 pixels between the **DIV** and any of its neighbors.

Let's start with defining the borders of our elements through CSS. For that, you can use



border-width, *border-color* and *border-style* attributes. The width of the border can be defined using a value and a unit (for example, "2px", "6pt", - the same way when defining text size) or using a descriptive property, like "thin", "medium" or "thick". Color is defined as any other color in CSS and HTML. The style of the border defines the line with which the border will be drawn. It can be "solid", "dotted", "groove", "ridge", "outset", etc. Here's how you would define a normal black border which is 1 pixel wide.

```
div { border-style: solid; border-width: 1px; border-color: Black; }
```

Similar to when defining a background, you don't need to write all the properties separately, you can use the *border* property to list all the values.

```
div { border: solid 1px black; }
```

Defining margins and padding for elements is a bit easier, as they don't have any additional properties, just the length. So, if you want to specify a margin or padding, use the *margin* or *padding* properties.

```
div { margin: 10px; padding: 5px; }
```

Here's one practical example. Let's say you're creating a page with lots of pictures, like an album. You want all of the pictures to have a uniform look, a thin border, and you also want to make sure that they aren't too close to each other. You could use the following style:

```
img.album { margin: 10px; border: solid 1px black; }
```

As you don't want this style applied to all images on the page, you created a class named "album" which can be applied only to **IMG** elements. Of course, don't forget to apply the attribute *class="album"* to all **IMG** elements you want to have a black border 1 pixel wide and a margin 10 pixels wide (every image in the album will have a 10 pixel space in every direction).

We've simplified everything a bit since we've only shown you how to apply margin, border or padding to all sides of the element. But you have many more properties at your disposal. You can specify the left, right, top and bottom border with the *border-left*, *border-right*, *border-top* and *border-bottom* properties respectfully. You can even specify the style, width and color of the border on each side with *border-left-style*, *border-left-width* and *border-left-color* properties. This works with other sides too, like *border-top-color* or *border-right-style*. This gives you much more flexibility – for example, you can create a left border that is different from the right one.

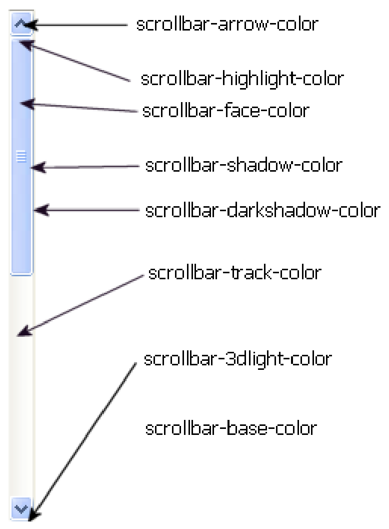
Similar options are available to margin and padding properties. You can set left, top, bottom and right margin widths with *margin-left*, *margin-top*, *margin-bottom* and *margin-right* properties. And also the padding widths – use *padding-left*, *padding-top*, *padding-bottom* and *padding-right* properties.

We'll mention two more useful CSS properties for defining width and height of an element, the *width* and *height* properties. They are useful if you want to specify a strict proportion of an element. For example, in HTML, when working with forms, you can only specify the width of the **INPUT** element by the number of characters, which is pretty inaccurate. Using CSS, you can easily define its width in pixels:

```
input { width: 150px; }
div { width: 300px; height: 450px; }
```

Scrollbars

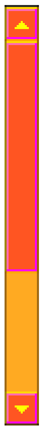
If you're working on a page to which design is rather important (for example, a band or an artist Web site), you might want to control the looks of something that is outside of the standard page area. With CSS you can change the colors of page scrollbar. Here are the properties you can change:



As you see in this picture, you can use 8 different properties. Here's how a custom scrollbar definition would look in CSS:

```
body

{

   scrollbar-3dlight-color: #FFD700;

   scrollbar-arrow-color: #FFFF00;

   scrollbar-base-color: #FF6347;

   scrollbar-darkshadow-color: #FFA500;

   scrollbar-face-color: #FF5522;

   scrollbar-highlight-color: #FF69B4;

   scrollbar-shadow-color: #FF00FF;

   scrollbar-track-color: #FFAA22;

}
```

We've created a custom scrollbar for the whole web page, using the **BODY** element selector. But you can create a custom scrollbar for other elements, like a **TEXTAREA** element. The previous example would result in the scrollbar shown in the following picture.

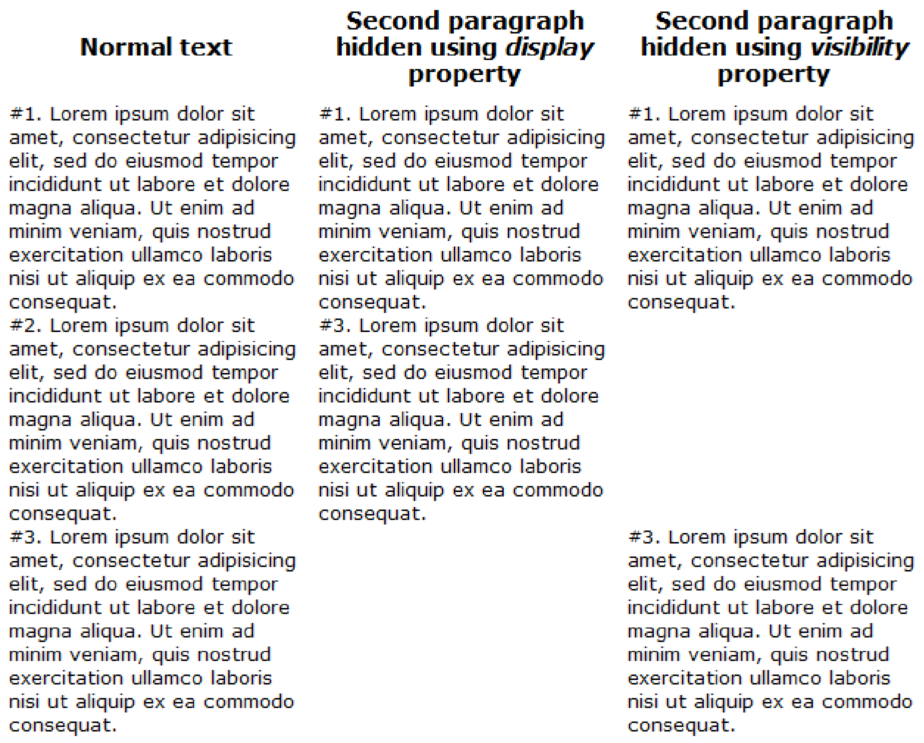


... And few more properties

We've covered all of the most common CSS properties, but there are still some that need mentioning.

Within HTML documents, you can also have elements that won't immediately be displayed to the user. The following properties will mostly be used within the *style* attribute of HTML tag and you'll dynamically change them with a client-side script written in JavaScript (more about this topic in the JavaScript guide).

The first property responsible for displaying an element is called *display*. The important thing to remember is that all elements that are hidden using this property won't be displayed on the page, and the page will be displayed as if they aren't even in the HTML code. The second property used for displaying an element is called *visibility*, and it has a different effect. Elements that are hidden using the *visibility* property won't be displayed on the screen, but there will be a blank space left for them on the page. They will simply be invisible, but their place on the page will be reserved and blank.



Normal text	**Second paragraph hidden using *display* property**	**Second paragraph hidden using *visibility* property**
#1. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.	#1. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.	#1. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
#2. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.	#3. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.	
#3. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.		#3. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Before we go on, let's explain one fact about HTML elements. They can, roughly, be divided into three groups: block elements, inline elements and list elements. Block elements (like **DIV**) take the whole available width and nothing can stand horizontally by them. Inline elements (like **SPAN**, **IMG** or **A**) are displayed in the row with all other elements, that can be to their left or to their right. List elements (like **LI**) are special, as they are displayed as a part of a list.

This comes into play when working with the *display* property – you can set its value to "none" (element isn't displayed), "block" (element is displayed, suitable for block elements) or "inline" (element is displayed, suitable for inline elements). So if you want to completely hide a block element, change the *display* property value from "block" to "none", and if you want to hide an inline element, change the value from "inline" to "none". Mixing up "inline" and "block" might lead to errors displaying elements on the page.

```
<div style="display:none;">This isn't displayed on the page.</div>

<div style="display:block;">This is displayed on the page.</div>
```

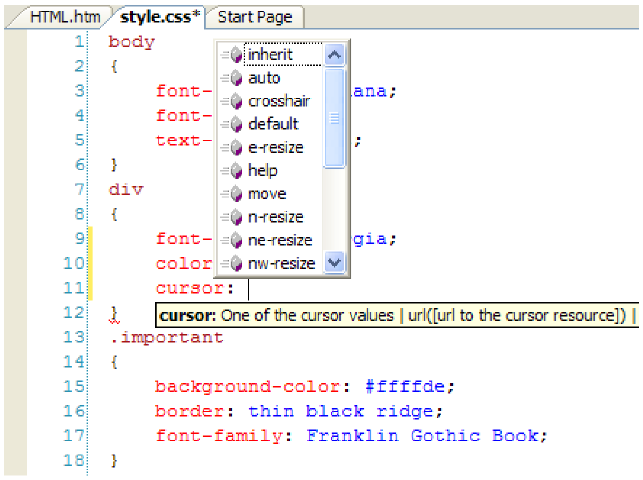
When working with the *visibility* property, things are a bit simpler. You can change its value between "visible" and "hidden".

```
<div style="visibility: hidden;">This text is invisible.</div>
```

But remember, even if the text is invisible on the page, it is still in the HTML source code. So don't hide any important or sensitive information this way, as it can be accessed when viewing the source code.

Before the end of this lesson, we'll introduce you to one final CSS property that you might use. If you want to change the mouse cursor that is displayed when moving over an element, you can do it using the *cursor* property. As before, we won't list all the available values as Visual Web Developer Express can provide them for you, but we will name a few of the most interesting. For example, if you want your cursor to be a hand (like when hovering over links), use the "pointer" value. It can also be an hourglass if you use the "wait" value, or a crosshair if you use the "crosshair" value.

```
<div style="cursor: crosshair;">Aim here!</div>
```



```
body
{
    font-        ana;
    font-
    text-        ;
}
div
{
    font-        gia;
    color
    cursor: 
}   cursor: One of the cursor values | url([url to the cursor resource]) |
.important
{
    background-color: #ffffde;
    border: thin black ridge;
    font-family: Franklin Gothic Book;
}
```

Quiz 4

Question 4.1: How would you define the style of a **DIV** element that has a border, and a space of 10 pixels to its left and right direction?

Answer 4.1: Use the *border*, *margin-left* and *margin-right* properties.

```
#myDiv { border: solid 1px black; margin-left: 10px; margin-right: 10px; }
```

Question 4.2: How would you create a textbox 150 pixels wide in which text would be written using Verdana font?

Answer 4.2: We can write this CSS definition within HTML code. Just use *width* and *font-family* properties.

```
<input type="text" style="width: 150px; font-family: Verdana;"
name="myText" />
```

Question 4.3: What is the most secure way to hide HTML content from visitors – use *display*, *visibility* or something else?

Answer 4.3: Something else – completely remove this HTML content from the Web page, as everything, even when hidden using CSS, is available when viewing HTML source code (in Internet Explorer go to View – Source).

Conclusion

As you've seen in this guide, although rather simple and straightforward, CSS can add great design elements to your Web pages, and can increase the design possibilities of simple HTML. If you are interested in CSS, don't stop here – we've only shown you some basic CSS capabilities, and there is a lot more to be discovered. Start with MSDN documentation and tutorials for CSS, http://msdn.microsoft.com/library/default.asp?url=/workshop/author/css/css_node_entry.asp, and soon you'll be on your way to becoming a CSS expert.

Author bio: Luka Abrus works for Microsoft Croatia as Technology Specialist. Prior to coming to Microsoft, Luka has worked as Web developer with several companies and as an author for Croatian IT magazines writing over hundred articles. He has written three books on Web development and Microsoft .NET. In his spare time, Luka is active in sports and playing piano.