Introduction to CSS

By: Shantanu Mishra

GDG J28

What is CSS??

- CSS stands for **Cascading Style Sheets** and is a language used for *describing the look and formatting of a document written in a markup language*.

HTML was NEVER intended to contain tags for formatting a web page!
HTML was created to describe the content of a web page, like:
<h1>This is a heading</h1>
*** This is a paragraph. ***

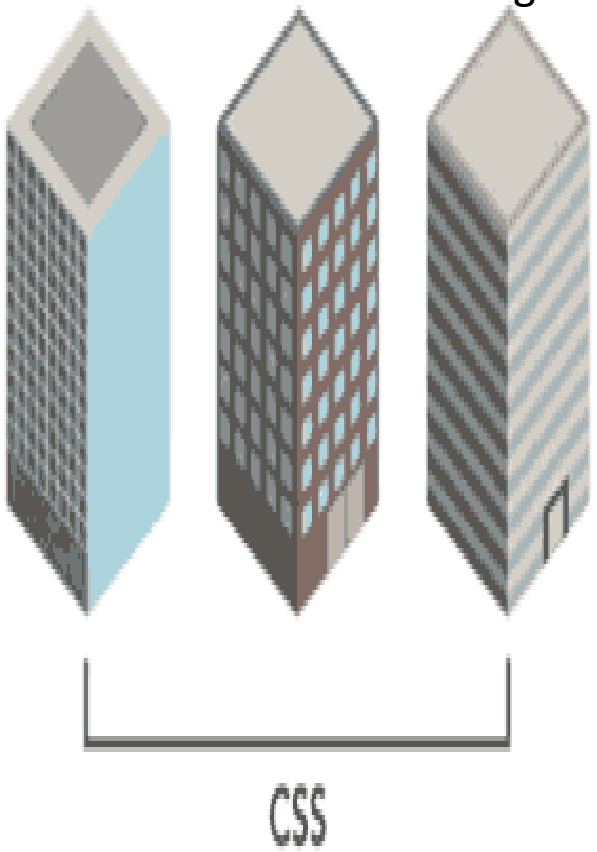
When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers.

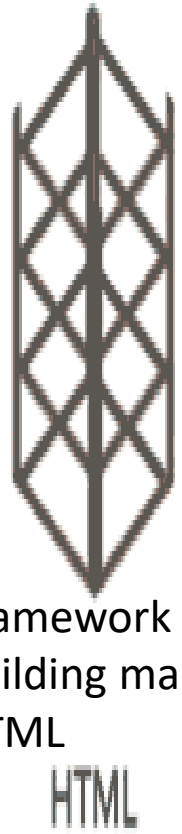
Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.

To solve this problem, the World Wide Web Consortium (W3C) created CSS.
CSS removed the style formatting from the HTML page!

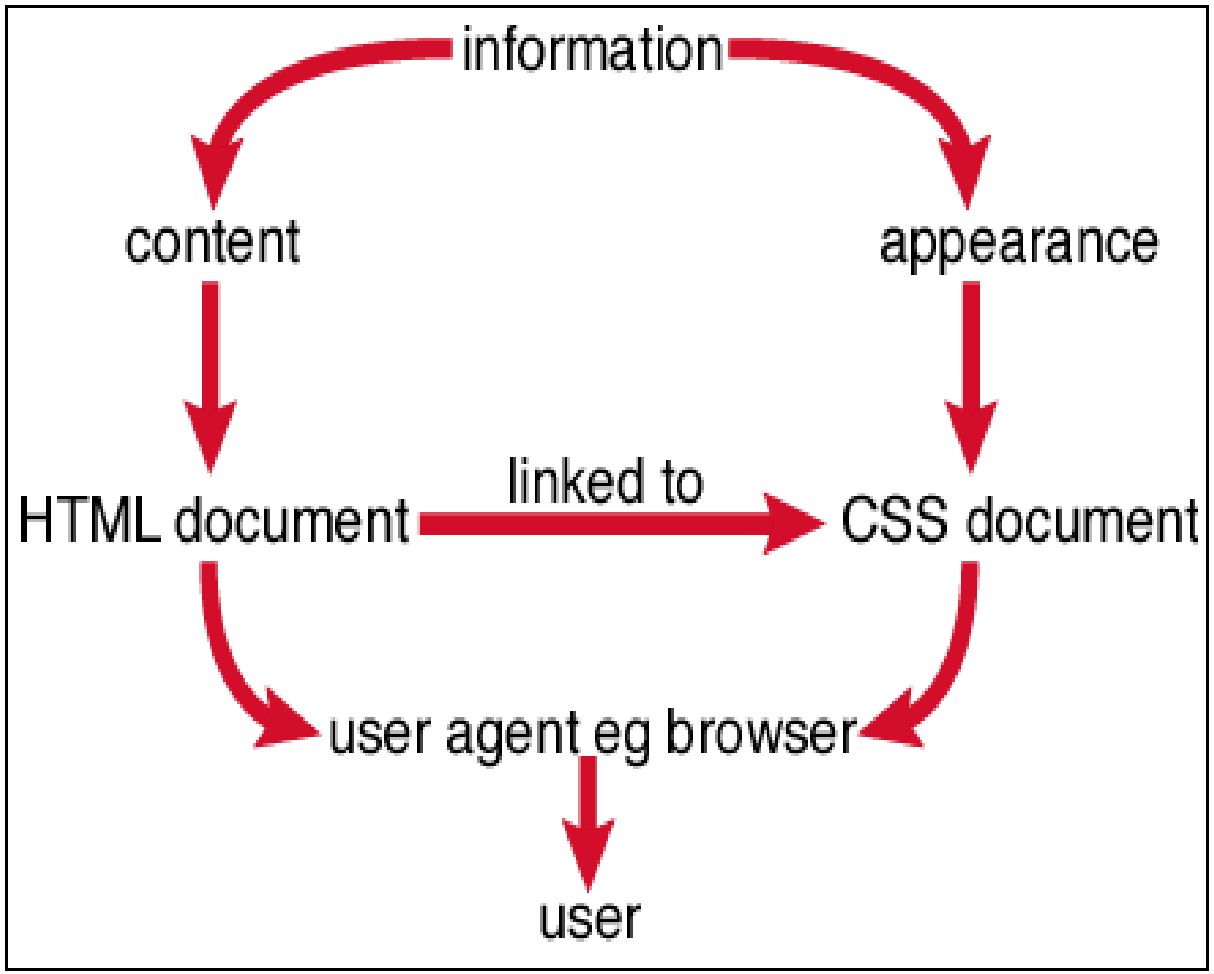
How CSS Works…

CSS beautifies and decorates the building





Framework of a building made in HTML

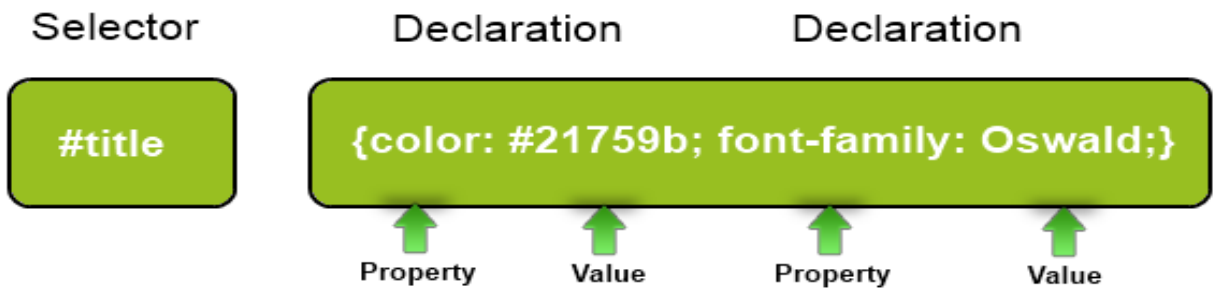


Building blocks of a CSS syntax…

For example, if you don't like the default look of the <h1>, <h2>, and other heading tags, you can assign new styles to override the default settings for the font family and size used, or whether bold or italics should be set, and many more properties too.

One way you can add styling to a web page is by inserting the required statements into the head of a web page between the <head> and </head> tags. So, to change the style of the <h1> tag, you might use the following code (I'll explain the syntax later):

```
<style>
  h1 {
Color : red;
 font-size : 3em;
font-family : Arial; }
 </style>
```



Importing a Style Sheet

When you wish to style a whole site, rather than a single page, a better way to manage style sheets is to move them completely out of your web pages to separate files, and then import the ones you need. This lets you apply different style sheets for different layouts (such as web and print), without changing the HTML.

There are a couple of different ways you can achieve this, the first of which is by using the CSS `@import` directive like this:

```
<style>
  @import url('styles.css');
</style>
```

This statement tells the browser to fetch a style sheet with the name *styles.css*. The `@import` command is quite flexible in that you can create style sheets that themselves pull in other style sheets, and so on. You need to just make sure that there are no `<style>` or `</style>` tags in any of your external style sheets, or they will not work.

Importing CSS from Within HTML

You can also include a style sheet with the HTML `<link>` tag like this:

```
<link rel='stylesheet' type='text/css' href='styles.css'>
```

This has the exact same effect as the `@import` directive, except that `<link>` is an HTML-only tag and is not a valid style directive, so it cannot be used from within one style sheet to pull in another, and also cannot be placed within a pair of `<style> ... </style>` tags.

Just as you can use multiple `@import` directives within your CSS to include multiple external style sheets, you can also use as many `<link>` elements as you like in your HTML.

Using IDs

A better solution for setting the style of an element is to assign an ID to it in the HTML, like this:

```
<div id='welcome'>Hello there</div>
```

This states that the contents of the `<div>` with the ID `welcome` should have applied to them the style defined in the `welcome` style setting. The matching CSS statement for this might look like the following

```
#welcome { font-style:italic; color:blue; }
```

Note the use of the # symbol, which specifies that only the ID with the name `welcome` should be styled with this statement.

Using Classes

If you would like to apply the same style to many elements, you do not have to give each one a different ID because you can specify a class to manage them all, like this:

```
<div class='welcome'>Hello</div>
```

This states that the contents of this element (and any others that use the class) should have applied to them the style defined in the `welcome` class. Once a class is applied you can use the following rule, either in the page header or within an external style sheet for setting the styles for the class:

```
.welcome { font-style:italic; color:blue; }
```

Instead of the # symbol, which is reserved for IDs, class statements are prefaced with a . (period).

Using Semicolons

In CSS, semicolons are used to separate multiple CSS statements on the same line. But if there is only one statement in a rule (or in an inline style setting within an HTML tag), you can omit the semicolon, as you can for the final statement in a group.

However, to avoid hard-to-find CSS errors, you may prefer to always use a semicolon after every CSS setting. You can then copy and paste them, and otherwise modify properties, without worrying about removing semicolons where they aren't strictly necessary or having to add them where they are required.

CSS Rules

Each statement in a CSS rule starts with a *selector*, which is the item to which the rule will be applied. For example, in this assignment, `h1` is the selector being given a font size 240% larger than the default:

```
h1 { font-size:240%; }
```

`font-size` is a *property*. Providing a value of `240%` to the `font-size` property of the selector ensures that the contents of all `<h1> ... </h1>` pairs of tags will be displayed at a font size that is 240% of the default size. All changes in rules must be within the `{` and `}` symbols that follow the selector. In `font-size:240%;` the part before the `:` (colon) is the property, while the remainder is the value applied to it.

Last comes a `;` (semicolon) to end the statement. In this instance, because `font-size` is the last property in the rule, the semicolon is not required (but it would be if another assignment were to follow).

Multiple Assignments

You can create multiple style declarations in a couple of different ways. First, you can concatenate them on the same line, like this:

```
h1 { font-size:240%; color:blue; }
```

This adds a second assignment that changes the color of all <h1> headings to blue. You can also place the assignments one per line, like the following:

```
h1 { font-size:240%;
color:blue; }
```

Or you can space out the assignments a little more, so that they line up below each other in a column at the colons, like this:

```
h1 {
  font-size:240%;
  color    :blue;
}
```

This way, you can easily see where each new set of rules begins, because the selector is always in the first column, and the assignments that follow are neatly lined up with all property values starting at the same horizontal offset. In the preceding examples, the final semicolon is unnecessary, but should you ever want to concatenate any such groups of statements into a single line, it is very quick to do with all semicolons already in place.

You can specify the same selector as many times as you want, and CSS combines all the properties. So the previous example could also be specified as:

```
h1 { font-size: 240%; }
h1 { color    : blue; }
```

There is no right or wrong way to lay out your CSS, but I recommend that you at least try to keep each block of CSS consistent with itself, so that other people can take it in at a glance.

What if you specified the same property to the same selector twice?

```
h1 { color : red; }
h1 { color : blue; }
```

The last value specified—in this case, blue—would apply. In a single file, repeating the same property for the same selector would be pointless, but such repetition happens frequently in real-life web pages when multiple style sheets are applied. It's one of the valuable features of CSS, and where the term *cascading* comes from.

External Style Sheets

The next types of styles are those assigned in an external style sheet. These settings will override any assigned either by the user or by the browser. External style sheets are the recommended way to create your styles because you can produce different style sheets for different purposes such as styling for general web use, for viewing on a mobile browser with a smaller screen, for printing purposes, and so on. Just apply the one needed for each type of media when you create the web page.

Internal Styles

Then there are internal styles, which you create within `<style>` ... `</style>` tags, and which take precedence over all the preceding style types. At this point, though, you are beginning to break the separation between styling and content, as any external style sheets loaded in at the same time will have a lower precedence.

Inline Styles

Finally, inline styles are where you assign a property directly to an element. They have the highest precedence of any style type, and are used like this:

```
<a href="http://google.com" style="color:green;">Visit Google</a>
```

In this example, the link specified will be displayed in green, regardless of any default or other color settings applied by any other type of style sheet, whether directly to this link or generically for all links.

When you use this type of styling, you are breaking the separation between layout and content; therefore, it is recommended that you do so only when you have a very good reason.

CSS Selectors

The means by which you access one or more elements is called *selection*, and the part of a CSS rule that does this is known as a *selector*. As you might expect, there are many varieties of selector.

The Type Selector

The type selector works on types of HTML elements such as `<p>` or `<i>`. For example, the following rule will ensure that all text within `<p> ... </p>` tags is fully justified:

```
p { text-align:justify; }
```

The Descendant Selector

Descendant selectors let you apply styles to elements that are contained within other elements. For example, the following rule sets all text within `<b>` ... `</b>` tags to red, but only if they occur within `<p>` ... `</p>` tags (like this: `<p><b>Hello</b> there</p>`):

```
p b { color:red; }
```

Descendant selectors can continue nesting indefinitely, so the following is a perfectly valid rule to make the text blue within bold text, inside a list element of an unordered list:

```
ul li b { color:blue; }
```

As a practical example, suppose you want to use a different numbering system for an ordered list that is nested within another ordered list. You can achieve this in the following way, which will replace the default numeric numbering (starting from `1`) with lowercase letters (starting from `a`):

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      ol ol { list-style-type:lower-alpha; }
    </style>
  </head>
  <body>
    <ol>
      <li>One</li>
      <li>Two</li>
      <li>Three
        <ol>
          <li>One</li>
          <li>Two</li>
          <li>Three</li>
        </ol>
      </li>
    </ol>
  </body>
</html>
```

The result of loading this HTML into a web browser is as follows, in which you can see that the second list elements display differently:

```
1. One
2. Two
3. Three
  a. One
  b. Two
  c. Three
```

The Child Selector

The child selector is similar to the descendant selector but is more restrictive about when the style will be applied, by selecting only those elements that are direct children of another element. For example, the following code uses a descendant selector that will change any bold text within a paragraph to red, even if the bold text is itself within italics (like this `<p><i><b>Hello</b> there</i></p>`):

```
p b { color:red; }
```

In this instance, the word `Hello` displays in red. However, when this more general type of behavior is not required, a child selector can be used to narrow the scope of the selector. For example, the following child selector will set bold text to red only if the element is a direct child of a paragraph, and is not itself contained within another element:

```
p > b { color:red; }
```

Now the word `Hello` will not change color because it is not a direct child of the paragraph.

For a practical example, suppose you wish to embolden only those `<li>` elements that are direct children of `<ol>` elements. You can achieve this as follows, where the `<li>` elements that are direct children of `<ul>` elements do not get emboldened:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      ol > li { font-weight:bold; }
    </style>
  </head>
  <body>
    <ol>
      <li>One</li>
      <li>Two</li>
      <li>Three</li>
    </ol>
    <ul>
      <li>One</li>
      <li>Two</li>
      <li>Three</li>
    </ul>
  </body>
</html>
```

The result of loading this HTML into a browser will be as follows:

1. **One**
2. **Two**
3. **Three**

- One
- Two
- Three

The ID Selector

If you give an element an ID name (like this: `<div id='mydiv'>`) then you can directly access it from CSS in the following way, which changes all the text in the element to italic:

```
#mydiv { font-style:italic; }
```

IDs can be used only once within a document, so only the first occurrence found will receive the new property value assigned by a CSS rule. But in CSS you can directly reference any IDs that have the same name, as long as they occur within different element types, like this:

```
<div id='myid'>Hello</div> <span id='myid'>Hello</span>
```

The Class Selector

When there are a number of elements in a page that you want to share the same styling, you can assign them all the same class name (like this: `<span class='myclass'>`); then, create a single rule to modify all those elements at once, as in the following rule, which creates a 10-pixel left margin offset for all elements using the class:

```
.myclass { margin-left:10px; }
```

In modern browsers, you can have HTML elements use more than one class by separating the class names with spaces, like this: `<span class='class1 class2 class3'>`. Remember, though, that some very old browsers only allow a single class name in a `class` argument.

You can narrow the scope of action of a class by specifying the types of elements to which it should apply. For example, the following rule applies the setting only to paragraphs that use the class `main`:

```
p.main { text-indent:30px; }
```

In this example, only paragraphs using the class `main` (like this: `<p class="main">`) will receive the new property value. Any other element types that may try to use the class (such as `<div class="main">`) will not be affected by this rule.

Because IDs normally apply only to unique elements, the following rule will apply an underline to only the first occurrence of `myid`:

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

However, you can ensure that CSS applies the rule to both occurrences like this:

```
span#myid { text-decoration:underline; }
div#myid  { text-decoration:underline; }
```

Or more succinctly like this (see "Selecting by Group" on page 435):

```
span#myid, div#myid { text-decoration:underline; }
```

I don't recommend using this form of selection because any JavaScript that also must access these elements cannot easily do so because the commonly used `getElementByID()` function will return only the first occurrence. To reference any other instances, a program would have to search through the whole list of elements in the document—a trickier task to undertake. So it's generally better to always use unique ID names.

The Difference Between Div and Span Elements

Both and elements are types of containers, but with some different qualities. By default, a element has infinite width (at least to the browser edge), which you can see by applying a border to one, like this:

```
<div style="border:1px solid green;">Hello</div>
```

A element, however, is only as wide as the text it contains. Therefore, the following line of HTML creates a border only around the word Hello, which does not extend to the righthand edge of the browser.

```
<span style="border:1px solid green;">Hello</span>
```

Also, elements follow text or other objects as they wrap around, and can therefore have a complicated border. For example, in Example 19-2, I used CSS to make the background of all elements yellow, to make all elements cyan, and to add a border to both, before then creating a few example and sections.

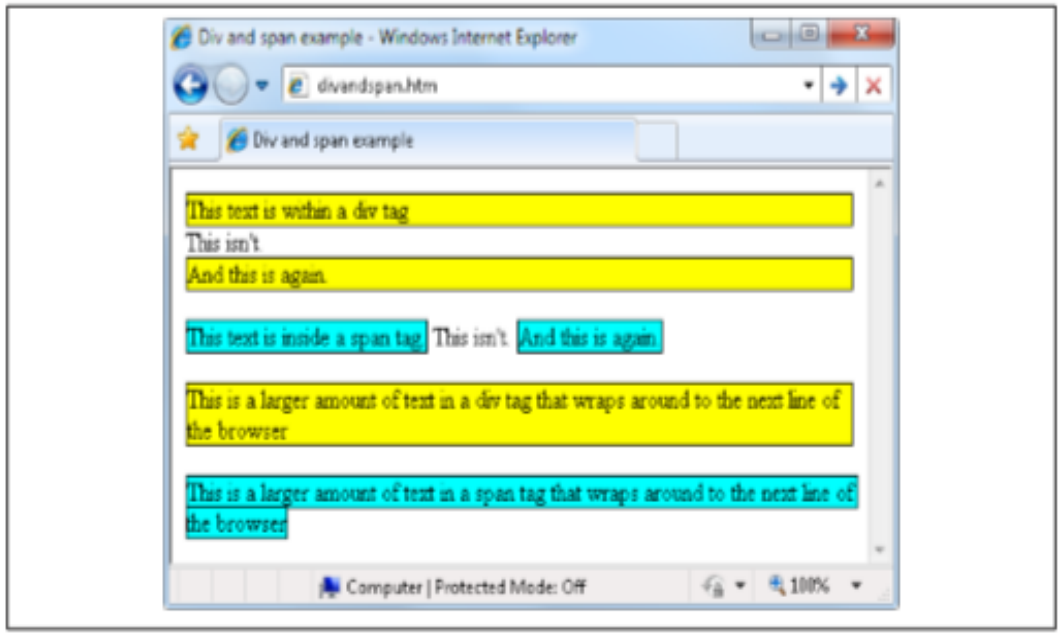


Figure 19-4. A variety of elements of differing width

The figure also shows how elements keep to themselves and take up only the space required to hold their content, without forcing subsequent content to appear below them.

For example, in the bottom two examples of the figure, you can also see that when elements wrap around the screen edge they retain a rectangular shape, whereas elements simply follow the flow of the text (or other contents) they contain.

Because tags can only be rectangular, they are better suited for containing objects such as images, boxouts, quotations, and so on, while tags are best used for holding text or other attributes that are placed one after another inline, and which should flow from left to right (or right to left in some languages).

Example 19-2. Div and span example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Div and span example</title>
    <style>
      div, span { border          :1px solid black; }
      div       { background-color:yellow;          }
      span      { background-color:cyan;            }
    </style>
  </head>
  <body>
    <div>This text is within a div tag</div>
    This isn't. <div>And this is again.</div><br>

    <span>This text is inside a span tag.</span>
    This isn't. <span>And this is again.</span><br><br>

    <div>This is a larger amount of text in a div that wraps around
    to the next line of the browser</div><br>

    <span>This is a larger amount of text in a span that wraps around
    to the next line of the browser</span>
  </body>
</html>
```

Figure 19-4 shows what this example looks like in a web browser. Although it is printed only in shades of gray in this book, the figure clearly shows how elements extend to the righthand edge of a browser, and force the following content to appear at the start of the first available position below them.

Measurements

CSS supports an impressive range of units of measurement, enabling you to tailor your web pages precisely to specific values, or by relative dimensions. The ones I generally use (and believe you will also find the most useful) are pixels, points, ems, and percent, but here's the complete list:

Pixels

The size of a pixel varies according to the dimensions and pixel depth of the user's monitor. One pixel equals the width/height of a single dot on the screen, and so this measurement is best suited to monitors. For example:

```
.classname { margin:5px; }
```

Points

A point is equivalent in size to 1/72 of an inch. The measurement comes from a print design background and is best suited for that medium, but is also commonly used on monitors. For example:

```
.classname { font-size:14pt; }
```

Inches

An inch is the equivalent of 72 points and is also a measurement type best suited for print. For example:

```
.classname { width:3in; }
```

Centimeters

Centimeters are another unit of measurement best suited for print. One centimeter is a little over 28 points. For example:

```
.classname { height:2cm; }
```

Millimeters

A millimeter is 1/10 of a centimeter (or almost 3 points). Millimeters are another measure best suited to print. For example:

```
.classname { font-size:5mm; }
```

Picas

A pica is another print typographic measurement, which is equivalent to 12 points. For example:

```
.classname { font-size:1pc; }
```

Ems

An em is equal to the current font size and is therefore one of the more useful measurements for CSS because it is used to describe relative dimensions. For example:

```
.classname { font-size:2em; }
```

Exs

An ex is also related to the current font size; it is equivalent to the height of a lowercase letter *x*. This is a less popular unit of measurement that is most often used as a good approximation for helping to set the width of a box that will contain some text. For example:

```
.classname { width:20ex; }
```

Percent

This unit is related to the em in that it is exactly 100 times greater (when used on a font). Whereas 1 em equals the current font size, the same size is 100 in percent. When not relating to a font, this unit is relative to the size of the container of the property being accessed. For example:

```
.classname { height:120%; }
```

Figure 19-5 shows each of these measurement types in turn being used to display text in almost identical sizes.

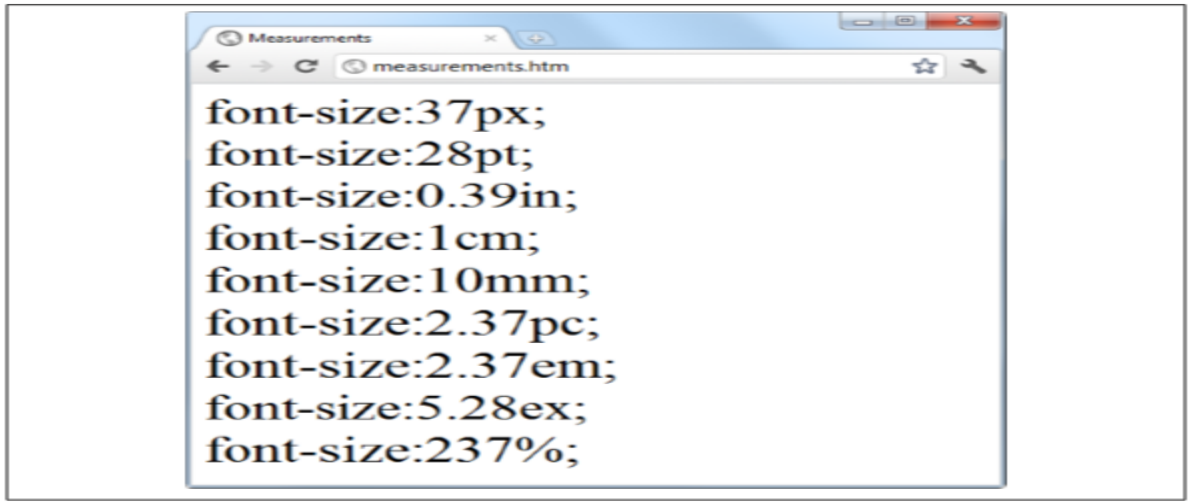


Figure 19-5. Different measurements that display almost the same

That’s all for beginners.
Thank You!

Remember:

Web Development = Team Work

HTML, CSS, JavaScript, PHP, MySQL etc. work together as a team to make an awesome webpage. Just Like GDG!!!

Happy Coding!!