Cascading Style Sheets (CSS)

Cascading Style Sheets (**CSS**) is a style sheet language used for describing the presentation semantics (the look and formatting) of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML.

CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the layout, colors, and fonts.

Advantages

- define the look of your pages in one place rather than repeating yourself over and over again throughout your site.
- easily change the look of your pages even after they're created. Since the styles are defined in one place you can change the look of the entire site at once.
- define font sizes and similar attributes with the same accuracy as you have with a word processor - not being limited to just the seven different font sizes defined in HTML.
- position the content of your pages with pixel precision.
- redefine entire HTML tags. Say for example, if you wanted the bold tag to be red using a special font - this can be done easily with CSS.
- define customized styles for links - such as getting rid of the underline.
- define layers that can be positioned on top of each other (often used for menus that pop up).

Examples

```
<font color="#FF0000" face="Verdana, Arial,  Helvetica, sans-serif">
   <strong>This is  text</strong></font>
```

This is verbose and contributes to making your HTML messy. With CSS, you can create a custom style elsewhere and set all its properties, give it a unique name and then ‘tag’ your HTML to apply these stylistic properties:

```
<p class="myNewStyle">My CSS styled text</p>
```

And in between the tags at the top of your web page you would insert this CSS code that defines the style we just applied:

```
<style type="text/css">
.myNewStyle {
   font-family: Verdana, Arial, Helvetica, sans-serif;
   font-weight: bold;
```

```
    color: #FF0000;
}</style>
```

Syntax

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties.

A style sheet consists of a list of *rules*. Each rule or rule-set consists of one or more *selectors*, and a *declaration block*.

Selector

In CSS, *selectors* are used to declare which part of the markup a style applies to, a kind of match expression. Selectors may apply to:

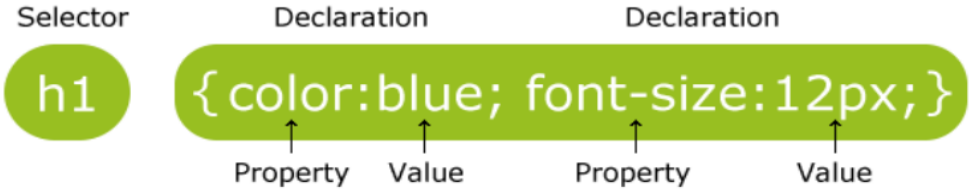
- all elements of a specific type, e.g. the second level headers h2
- to elements specified by attribute, in particular:
  - *id*: an identifier unique to the document
  - *class*
- to elements depending on how they are placed relative to, or nested within, others in the document tree.

Pseudo-classes are used in CSS selectors to permit formatting based on information that is outside the document tree. An often-used example of a pseudo-class is `:hover`, which identifies content only when the user 'points to' the visible element, usually by holding the mouse cursor over it. It is appended to a selector as in `a:hover` or `#elementid:hover`. A *pseudo-class* classifies document elements, such as `:link` or `:visited`, whereas a *pseudo-element* makes a selection that may consist of partial elements, such as `:first-line` or `:first-letter`.[3]

Selectors may be combined in many ways, especially in CSS 2.1, to achieve great specificity and flexibility.[4]

Declaration block

A declaration block consists of a list of *declarations* in braces. Each declaration itself consists of a *property*, a colon (`:`), and a *value*. If there are multiple declarations in a block, a semi-colon (`;`) must be inserted to separate each declaration.[5]



Why Use External CSS?

It keeps your website design and content separate.

It's much easier to reuse your CSS code if you have it in a separate file. Instead of typing the same CSS code on every web page you have, simply have many pages refer to a single CSS file with the "link" tag.

You can make drastic changes to your web pages with just a few changes in a single CSS file.

Selectors

*1. **

```
* {
  margin: 0;
  padding: 0;
}
```

The star symbol will target every single element on the page. Many developers will use this trick to zero out the `margin`s and `padding`. While this is certainly fine for quick tests, I'd advise you to never use this in production code. It adds too much *weight* on the browser, and is unnecessary.

2. #X

```
#container {
    width: 960px;
    margin: auto;
}
```

Prefixing the hash symbol to a selector allows us to target by `id`. This is easily the most common usage, however be cautious when using `id` selectors.

3. .X

```
.error {
  color: red;
}
```

This is a `class` selector. The difference between `id`s and `class`es is that, with the latter, you can target multiple elements. Use `class`es when you want your styling to apply to a group of elements. Alternatively, use `id`s to find a needle-in-a-haystack, and style only that specific element.

4. X Y

```
li a {
  text-decoration: none;
}
```

The next most comment selector is the `descendant` selector. When you need to be more specific with your selectors, you use these

5. X

```
a { color: red; }
ul { margin-left: 0; }
```

What if you want to target all elements on a page, according to their `type`, rather than an `id` or `class`name? Keep it simple, and use a type selector. If you need to target all unordered lists, use `ul {}`.

6. X:visited and X:link

view plaincopy to clipboardprint?

```
1. a:link { color: red; }
2. a:visted { color: purple; }
```

We use the `:link` pseudo-class to target all anchors tags which have yet to be clicked on.

Alternatively, we also have the `:visited` pseudo class, which, as you'd expected, allows us to apply specific styling to only the anchor tags on the page which *have* been clicked on, or *visited*.

7. X + Y

```
1. ul + p {
2.     color: red;
3. }
```

This is referred to as an adjacent selector. It will select *only* the element that is immediately preceded by the former element. In this case, only the first paragraph after each `ul` will have red text.

8. X > Y

```
1. div#container > ul {
2.   border: 1px solid black;
3. }
```

The difference between the standard `X Y` and `X > Y` is that the latter will only select direct children. For example, consider the following markup.

```
1. <div id="container">
2.     <ul>
3.       <li> List Item
4.         <ul>
5.           <li> Child </li>
6.         </ul>
7.       </li>
8.       <li> List Item </li>
9.       <li> List Item </li>
10.            <li> List Item </li>
11.          </ul>
12.        </div>
```

A selector of `#container > ul` will only target the `ul`s which are direct children of the `div` with

an `id` of `container`. It will not target, for instance, the `ul` that is a child of the first `li`.

For this reason, there are performance benefits in using the child combinator. In fact, it's recommended particularly when working with JavaScript-based CSS selector engine

9. X ~ Y

```
ul ~ p {
    color: red;
}
```

This sibling combinator is similar to `X + Y`, however, it's less strict. While an adjacent selector (`ul + p`) will only select the first element that is *immediately* preceded by the former selector, this one is more generalized. It will select, referring to our example above, any `p` elements, as long as they follow a `ul`.

10. X[title]

```
a[title] {
    color: green;
}
```

Referred to as an *attributes selector*, in our example above, this will only select the anchor tags that have a `title` attribute. Anchor tags which do not will not receive this particular styling. But, what if you need to be more specific? Well…

11. X[href="foo"]

view plaincopy to clipboardprint?

```
a[href="http://net.tutsplus.com"] {
  color: #1f6053; /* nettuts green */
}
```

The snippet above will style all anchor tags which link to *http://net.tutsplus.com*; they'll receive our branded green color. All other anchor tags will remain unaffected.

Selector	Example	Example description	CSS
.class	.intro	Selects all elements with class="intro"	1
#id	#firstname	Selects the element with id="firstname"	1
*	*	Selects all elements	2
element	p	Selects all <p> elements	1
element,element	div,p	Selects all elements and all <p> elements	1
element element	div p	Selects all <p> elements inside elements	1
element>element	div>p	Selects all <p> elements where the parent is a element	2
element+element	div+p	Selects all <p> elements that are placed immediately after elements	2
[*attribute*]	[target]	Selects all elements with a target attribute	2
[*attribute=value*]	[target=_blank]	Selects all elements with target="_blank"	2
[*attribute~=value*]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"	2
[*attribute|=value*]	[lang|=en]	Selects all elements with a lang attribute value starting with "en"	2
:link	a:link	Selects all unvisited links	1
:visited	a:visited	Selects all visited links	1
:active	a:active	Selects the active link	1
:hover	a:hover	Selects links on mouse over	1
:focus	input:focus	Selects the input element which has focus	2
:first-letter	p:first-letter	Selects the first letter of every <p> element	1
:first-line	p:first-line	Selects the first line of every <p> element	1
:first-child	p:first-child	Selects every <p> element that is the first child of its parent	2
:before	p:before	Insert content before the content of every <p> element	2
:after	p:after	Insert content after every <p> element	2
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute equal to "it" (Italian)	2
element1~element2	p~ul	Selects every element that are preceded by a <p> element	3
[*attribute^=value*]	a[src^="https"]	Selects every <a> element whose src attribute value begins with "https"	3
[*attribute$=value*]	a[src$=".pdf"]	Selects every <a> element whose src attribute value ends with ".pdf"	3
[*attribute*=value*]	a[src*="w3schools"]	Selects every <a> element whose src attribute value contains the substring "w3schools"	3
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent	3
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent	3

:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent	3
:only-child	p:only-child	Selects every <p> element that is the only child of its parent	3
:nth-child(*n*)	p:nth-child(2)	Selects every <p> element that is the second child of its parent	3
:nth-last-child(*n*)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child	3
:nth-of-type(*n*)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent	3
:nth-last-of-type(*n*)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child	3
:last-child	p:last-child	Selects every <p> element that is the last child of its parent	3
:root	:root	Selects the document's root element	3
:empty	p:empty	Selects every <p> element that has no children (including text nodes)	3
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)	3
:enabled	input:enabled	Selects every enabled <input> element	3
:disabled	input:disabled	Selects every disabled <input> element	3
:checked	input:checked	Selects every checked <input> element	3
:not(*selector*)	:not(p)	Selects every element that is not a <p> element	3
::selection	::selection	Selects the portion of an element that is selected by a user	3

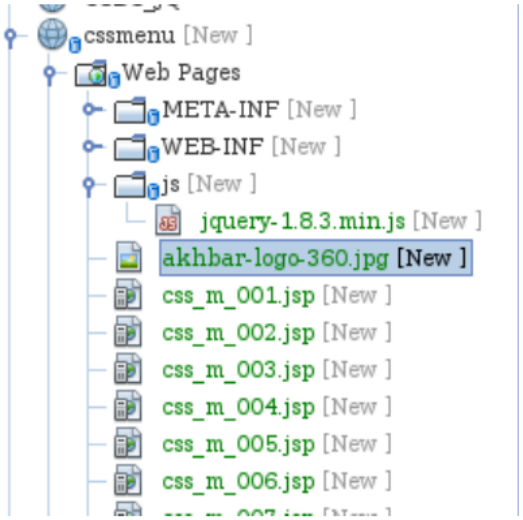
jQuery

jQuery is a multi-browser JavaScript library designed to simplify the client-side scripting of HTML. It was released in January 2006 at BarCamp NYC by John Resig. It is currently developed by a team of developers led by Dave Methvin. Used by over 55% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today.

jQuery is free, open source software, licensed under the MIT License.[4] jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. J

Remember to download the jquery library from
http://jquery.com/

and add to the project under the web directory in a directory called js before running the following examples.



Hello jQuery

We start with an empty html page:

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
  // we will add our javascript code here
</script>
</head>
<body>
  <!-- we will add our HTML content here -->
</body>
</html>
```

This page just loads the jquery.js library (make sure the URL points to where you stored your copy of jquery! This example assumes that you store it in the same directory as this example file). Two comments indicate where we will expand this template with code.

As almost everything we do when using jQuery reads or manipulates the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

To do this, we register a ready event for the document.

```
$(document).ready(function() {
  // do stuff when DOM is ready
});
```

Putting an alert into that function does not make much sense, as an alert does not require the DOM to be loaded. So lets try something a little more sophisticated: Show an alert when clicking a link.

Add the following to the `<body>`:

```
<a href="">Link</a>
```

Now update the `$(document).ready` handler:

```
$(document).ready(function() {
  $("a").click(function() {
    alert("Hello world!");
  });
});
```

This should show the alert as soon as you click on the link. You are ready now to copy and paste this script into your custom.js file. Then, open starterkit.html in the browser and click any link. You should see a pop-up window with "Hello world!" message regardless of what link was clicked.

Let's have a look at what we are doing: `$("a")` is a jQuery selector, in this case, it selects all **a** elements. `$` itself is an alias for the jQuery "class", therefore `$()` constructs a new jQuery object. The

`click()` function we call next is a method of the jQuery object. It binds a click event to all selected elements (in this case, a single anchor element) and executes the provided function when the event occurs.

This is similar to the following code:

```
<a href="" onclick="alert('Hello world')">Link</a>
```

The difference is quite obvious: We don't need to write an onclick for every single element. We have a clean separation of structure (HTML) and behavior (JS), just as we separate structure and presentation by using CSS.

With this in mind, we explore selectors and events a little further.

Find me: Using selectors and events

jQuery provides two approaches to select elements. The first uses a combination of CSS and XPath selectors passed as a string to the jQuery constructor (eg. `$("div > ul a")`). The second uses several methods of the jQuery object. Both approaches can be combined.

To try some of these selectors, we select and modify the first ordered list in our starterkit.

To get started, we want to select the list itself. The list has an ID "orderedlist". In classic JavaScript, you could select it by using `document.getElementById("orderedlist")`. With jQuery, we do it like this:

```
$(document).ready(function() {
  $("#orderedlist").addClass("red");
});
```

The starterkit provides a stylesheet with a class "red" that simply adds a red background. Therefore, when you reload the page in your browser, you should see that the first ordered list has a red background. The second list is not modified.

Now lets add some more classes to the child elements of this list.

```
$(document).ready(function() {
  $("#orderedlist > li").addClass("blue");
});
```

This selects all child `li`s of the element with the id orderedlist and adds the class "blue".

Now for something a little more sophisticated: We want to add and remove the class when the user hovers the li element, but only on the last element in the list.

```
$(document).ready(function() {
  $("#orderedlist li:last").hover(function() {
    $(this).addClass("green");
  },function(){
    $(this).removeClass("green");
```

```
  });
});
```

For every onxxx event available, like onclick, onchange, onsubmit, there is a jQuery equivalent. Some other events, like ready and hover, are provided as convenient methods for certain tasks.

With those selectors and events you can already do a lot of things, but there is more.

```
$(document).ready(function() {
  $("#orderedlist").find("li").each(function(i) {
    $(this).append( " BAM! " + i );
  });
});
```

`find()` allows you to further search the descendants of the already selected elements, therefore `$("#orderedlist").find("li")` is mostly the same as `$("#orderedlist li")`.

`each()` iterates over every element and allows further processing. Most methods, like `addClass()`, use `each()` themselves.

In this example, `append()` is used to append some text to it and set it as text to the end of each element.

Another task you often face is to call methods on DOM elements that are not covered by jQuery. Think of a form you would like to reset after you submitted it successfully via AJAX.

```
$(document).ready(function() {
  // use this to reset a single form
  $("#reset").click(function() {
    $("form")[0].reset();
  });
});
```

This code selects the first form element and calls `reset()` on it. In case you had more than one form, you could also do this:

```
$(document).ready(function() {
  // use this to reset several forms at once
  $("#reset").click(function() {
    $("form").each(function() {
      this.reset();
    });
  });
});
```

This would select all forms within your document, iterate over them and call `reset()` for each. Note that in an `.each()` function, `this` refers to the actual element. Also note that, since the `reset()` function belongs to the form element and not to the jQuery object, we cannot simply call `$("form").reset()` to reset all the forms on the page.

An additional challenge is to select only certain elements from a group of similar or identical ones. jQuery provides `filter()` and `not()` for this. While `filter()` reduces the selection to the elements that fit the filter expression, `not()` does the contrary and removes all elements that fit the expression. Think of an unordered list where you want to select all li elements that have no ul children.

```
$(document).ready(function() {
  $("li").not(":has(ul)").css("border", "1px solid black");
});
```

This selects all li elements that have a ul element as a child and removes all elements from the selection. Therefore all li elements get a border, except the one that has a child ul.

The `[expression]` syntax is taken from XPath and can be used to filter by attributes. Maybe you want to select all anchors that have a name attribute:

```
$(document).ready(function() {
  $("a[name]").css("background", "#eee" );
});
```

This adds a background color to all anchor elements with a name attribute.

More often than selecting anchors by name, you might need to select anchors by their "href" attribute. This can be a problem as browsers behave quite inconsistently when returning what they think the "href" value is (Note: This problem was fixed recently in jQuery, available in any versions after 1.1.1). To match only a part of the value, we can use the contains select "*=" instead of an equals ("="):

```
$(document).ready(function() {
  $("a[href*='/content/gallery']").click(function() {
    // do something with all links that point somewhere to /content/gallery
  });
});
```

```
<html>
  <head>
    <script type="text/javascript" src="js/jquery-1.8.3.min.js"></script>
    <script>
      $(document).ready(function(){
        $("p").click(function(){
          $(this).hide();
        });
      });
    </script>
  </head>
  <body>
    <p>If you click on me, I will disappear.</p>
```

```
    <p>If you click on me, I will disappear.</p>
    <p>If you click on me, I will disappear.</p>
  </body>
</html>



<html>
  <head>
    <script type="text/javascript" src="js/jquery-1.8.3.min.js"></script>
    <!DOCTYPE html>
  <script>
    $(document).ready(function(){
      $("button").click(function(){
        $("p").toggle();
      });
    });
  </script>
</head>
<body  onload="" >
  <button>Toggle</button>
  <p>This is a paragraph with little content.</p>
  <p>This is another small paragraph.</p>
</body>
</html>



<html>
  <head>
    <script type="text/javascript" src="js/jquery-1.8.3.min.js"></script>
    <!DOCTYPE html>
  <html>
    <head>
      <script src="jquery.js"></script>
      <script>
        $(document).ready(function(){
          $("#button").click(function(){
            $("#div1").fadeToggle();
            $("#div2").fadeToggle("slow");
            $("#div3").fadeToggle(3000);
```

```
            });
        });
    </script>
  </head>
  <body>
    <p>Demonstrate fadeToggle() with different speed parameters.</p>
    <button  ID="button">Click to fade in/out boxes</button>
    <button  >Click to fade in/out boxes</button>
    <button>Click to fade in/out boxes</button>      <br>
    <div id="div1" style="width:80px;height:80px;background-color:red;"></div>      <br>
    <div id="div2" style="width:80px;height:80px;background-color:green;"></div>  <br>
    <div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
  </body>
</html>
```