

CSS basics

What is CSS?

Whilst HTML structures the document and tells browsers what a certain element's function is (it is a link to another page? Is it a heading?), CSS gives the browser instructions on how to display a certain element—styling, spacing and positioning. If HTML is the struts and bricks that make up the structure of a house, CSS is the plaster and paint to decorate it. This is done using a system of rules, the exact syntax of which you'll learn more about below. These rules state what HTML elements should have styling added to them, and then within each rule list the properties (eg colour, size, font, etc.) of those HTML elements they want to manipulate, and what values they want to change them to. For example, a CSS rule might state “I want to find every `h2` element and colour them all green”, or “I want to find every paragraph with a class name of `author-name`, colour their backgrounds in red, make the text inside them twice the size of normal paragraph text, and add 10 pixels of spacing around each one. CSS is not a programming language like JavaScript and it is not a markup language like HTML—actually there is nothing that can be compared to it. Technologies that defined interfaces before web development always mixed presentation and structure. This is not a clever thing to do in an environment that changes as often as the web, which is why CSS was invented.

Defining style rules

Without further ado, let's have a look at a CSS code example, and then dissect it:

```
selector {  
  property1:value;  
  property2:value;  
  property3:value;  
}
```

The pertinent parts are as follows:

- The selector identifies the HTML elements that the rule will be applied to, identified by the actual element name, eg `body`, or by other means such as `class` attribute values—we'll get on to these later.
- The curly braces contain the property/value pairs, which are separated from each other by semi-colons; the properties are separated from their respective values by colons,
- The properties define what you want to do to the element(s) you have selected. These come in wide varieties, which can affect element colour, background colour, position, margins, padding, font type, and many other things.
- The values are the values that you want to change each property of the selected elements to. The values are dependent on the property, for example properties that affect colour can take hexadecimal colours, like `#336699`, RGB values like `rgb(12,134,22)` or colour names like `red`, `green` or `blue`. Properties that affect position, margins, width, height etc can be measured in pixels, ems, percentages, centimeters or other such units.

Now let's look at a specific example:

```
p {  
  margin:5px;  
  font-family:arial;  
  color:blue;
```

```
}
```

The HTML element this rule selects is `p`—every `p` in the HTML document(s) that use this CSS will have this rule applied to it, unless they have more specific rules also applied to them, in which case the more specific rule(s) will overwrite this rule. The properties affected by this rule are the margins around the paragraphs, the font of the text inside the paragraphs, and the colour of that text. The margins are set at 5 pixels, the font is set as Arial, and the colour of the text is set as blue.

We will come back to all of these specifics later—the main goal of this tutorial is to cover the basics of CSS and not the nitty-gritty details.

A whole set of these rules goes together to form a style sheet. This is the most basic syntax of CSS there is. There is more, but not much—probably the coolest thing about CSS is its simplicity.

CSS comments

One thing to know early on is how to comment in CSS. You add comments by enclosing them in `/*` and `*/`. Comments can span several lines, and the browser will ignore these lines:

```
/* These are basic element selectors */
selector{
  property1:value;
  property2:value;
  property3:value;
}
```

You can add comments either between rules or inside the property block—for example in the following CSS the 2nd and 3rd properties are enclosed inside comment delimiters, so they will be ignored by the browser. This is useful when you are checking out what effect certain parts of your CSS is having on your web page; just comment them out, save your CSS, and reload the HTML.

```
selector{
  property1:value;
  /*
  property2:value;
  property3:value;
  */
}
```

Unlike other languages, CSS only has block level comments—single line comments do not exist. You can of course constrain the comment to a single line if you wish, but you still need to include the opening and closing comment delimiters (`/*` and `*/`).

Grouping selectors

You can also group different selectors. Say you want to apply the same style to `h1` and `p`—you could write the following CSS:

```
h1 {color:red}
p {color:red}
```

This however is not ideal, as you repeat information that is the same. Therefore you can shorten the CSS by grouping the selectors together with a comma—the rules within the brackets are applied to both selectors:

```
h1, p {color:red}
```

There are several different selectors, each matching a different part of the markup. The three most basic ones that you'll encounter most often are as follows:

`p {}`: element selector

matches all the elements of that name on the page (p elements, in the case above)

`.example {}`: class selector

matches all elements that have a `class` attribute with the value specified, so the above would match `<p class="example">`, `<li class="example">` or `<div class="example">`, or any other element with a `class` of `example`. Note that class selectors don't test for any specific element name

`#example {}`: id selector

matches all elements that have an `id` attribute with the value specified, so the above would match `<p id="example">`, `<li id="example">` or `<div id="example">`, or any other element with an `id` of `example`. Note that ID selectors don't test for any element name, and you can only have one of each ID per HTML document—they are unique to each page.

You can see the above selectors in action in the following examples. Notice that when you open the example in a browser the `warning` style gets applied to both the list item and the paragraph (if the bullet disappears it's because you are setting a white text colour on a white background).

- [example-selectors.html](#)
- [selectors.css](#)

You can join some selectors to define even more specific rules:

```
p.warning {}
```

matches all paragraphs with the `class` of `warning`

```
div#example {}
```

matches the element with the `id` attribute `example`, but only when it is a `div`

```
p.info, li.highlight {}
```

matches paragraphs with a `class` of `info` and list items with a `class` of `highlight`

In the following example I use these to differentiate between the different warning styles:

- [example-specificselectors.html](#)
- [specificselectors.css](#)

Advanced CSS selectors

In the above section, I introduced you to the most basic of CSS selectors, element, class and id selectors. With these selectors you can accomplish a lot, but this certainly isn't the be all and end all of selectors—there are other selectors that allow you to select elements to style based on more specific criteria:

- Universal selectors: universal selectors can be used to select every element on the page.
- Attribute selectors: as their name suggests, attribute selectors allow you to select elements based on their attributes.
- Child selectors: if you want to select specific elements that are children of other specific elements, use this selector.
- Descendent selectors: if you want to select specific elements that are descendents of other specific elements (not just direct children, but further down in the tree as well), you can use this selector type.
- Adjacent sibling selectors: if you want to select just specific elements that follow other specific

elements, use these selectors.

- Pseudo-classes: these allow you to style elements based not on what the elements are, but on more esoteric factors such as the states of links (eg if they are being hovered over, or have been visited already).
- Pseudo-elements: these allow you to style specific parts of elements, rather than the whole element (eg the first letter within that element); they also allow you to insert content before or after specific elements.

You will see references to some of the more complicated selectors as you progress through the rest of the curriculum, but don't worry if you don't understand them all immediately—you will get there as you gain more experience in styling web pages! It is best to start off easy with the three basic selectors mentioned in the above section, then move on to the others as you gain more confidence.

Universal selectors

Put simply, universal selectors select every element on a page to apply styles to. For example, the following rule says that every element on the page should be given a solid 1 pixel black border:

```
* {  
  border: 1px solid #000000;  
}
```

Attribute selectors

Attribute selectors allow you to select elements based on attributes they contain. For example, you can select every `img` element with an `alt` attribute with the following selector:

```
img[alt] {  
  border: 1px solid #000000;  
}
```

Note the square brackets.

Using the above selector, you could perhaps choose to put a black border around any images that have an `alt` attribute, and style other images with a bright red border—useful in accessibility testing.

But attributes instantly get more useful when you consider that you can select by *attribute value*, not just attribute names. The following rule gives all images with an `src` attribute value of `alert.gif`:

```
img[src="alert.gif"] {  
  border: 1px solid #000000;  
}
```

You might not think this is hugely useful, but again, it can be useful for debugging purposes. Far more useful is the ability to select for specific parts of attributes, for example file extensions. And this is on the way—CSS 3 actually introduces three new types of attribute selector that can select based on text strings in attribute values (at the beginning, end, or anywhere within the value). [Read Christopher Schmitt's article on CSS 3 attribute selectors.](#)

Child selectors

You can use a child selector to select just specific elements that are children of other specific elements. For example, the following rule will turn the text of `strong` elements that are children of `h3` elements

blue, but no other strong elements: p

```
h3 > strong {
  color: blue;
}
```

Child selectors are not supported in IE 6 or below.

Descendent selectors

Descendent selectors are very similar to child selectors, except that child selectors only select **direct** descendents; descendent selectors select suitable elements anywhere in the element hierarchy, not just direct descendents. Let's look at what this means more carefully. Consider the following HTML snippet:

```
<div>
  <em>hello</em>
  <p>In this paragraph I will say
    <em>goodbye</em>.
  </p>
</div>
```

In this snippet, the `div` element is the parent of all the others. It has two children, an `em` and a `p`. The `p` element has a single child element, another `em`.

You could use a child selector to select just the `em` immediately inside the `div`, like so:

```
div > em {
  ...
}
```

If you instead used a descendent selector, as follows:

```
div em {
  ...
}
```

Both of the `em` elements would be selected.

Adjacent sibling selectors

These selectors allow you to select a specific element that comes directly after another specific element, on the same level in the element hierarchy. For example, if you wanted to select all `p` elements that come immediately after `h2` elements, but no other `p` elements, you could use the following rule:

```
h2 + p {
  ...
}
```

Adjacent sibling selectors are not supported in IE 6 or below.

Pseudo-classes

Pseudo-classes are used to provide styles not for elements, but for various states of elements. The most common usage you'll come across is styling link states, so I'll look at these first. The following list gives you the different pseudo-classes, and a description of the link state they select:

- `:link` —the normal, default state of links, just as you first found them.
- `:visited` —links that you have already visited in the browser you are currently using.
- `:focus` —links (or form fields, or anything else) that currently have the keyboard cursor within them.
- `:hover` —links that are currently being hovered over by the mouse pointer.
- `:active` —a link that is currently being clicked on.

The following CSS rules make it so that by default, links are blue (the default in most browsers anyway). When hovered over, the default link underline disappears. We want the same effect when the link is focussed via the keyboard, so we duplicate the `:hover` rule with `:focus`. When a link has already been visited, it turns grey. Finally, when a link is active, it is bolded, as an extra clue that something is about to happen.

```
a:link{
  color: blue;
}
a:visited{
  color: gray;
}
a:hover a:focus{
  text-decoration: none;
}
a:active{
  font-weight: bold;
}
```

Take care if you don't specify these rules in the same order as they are shown in above, otherwise they might not work as you expect. This is due to the way specificity causes later rules in the stylesheet to override earlier rules. You'll [learn more about specificity in the next article](#).

The `:focus` pseudo-class is also useful as a usability aid in forms. For example, you can highlight the input field that has the active blinking cursor inside it with a rule like this:

```
input:focus {
  border: 2px solid black;
  background color: lightgray;
}
```

Next, I'll have a look at `:first-child`—this pseudo-class selects any instance of the element that is the first child element of its parent, so for example, the following rule selects the first list item (bulleted or numbered) in any list and makes its text bold:

```
li:first-child {
  font-weight: bold;
}
```

Lastly, I'll briefly mention the `:lang` pseudo-class, which selects elements whose languages have been set to the specified language using the `lang` attribute. For example, the following element:

```
<p lang="en-US">A paragraph of American text, gee whiz!</p>
```

Could be selected using the following:

```
p:lang(en-US) {
  ...
}
```

Pseudo-elements

Pseudo elements have two purposes. First of all, you can use them to select the first letter or first line of text inside a given element. To create a drop cap easily at the start of every paragraph of your document, you could use the following rule:

```
p:first-letter {
  font-weight: bold;
  font-size: 300%;
  background-color: red;
}
```

the first letter of every paragraph will now be bolded, 300% bigger than the rest of the paragraph, and have a red background.

To make the first line of every paragraph bold, you could use the following rule:

```
p:first-line {
  font-weight: bold;
}
```

The second use of pseudo-elements is generating content via CSS, which is more complicated. You can use the `:before` or `:after` pseudo-elements to specify that content should be inserted before or after the element you are selecting. You then specify *what it is* that you want to insert. As a simple example, you can use the following rule to insert a decorative images after every link on the page:

```
a:after{
  content: " " url(flower.gif);
}
```

You can also use the `attr()` function to insert the values of attributes of the elements after the element. For example, you could insert the target of every link in your document in brackets after them using the following:

```
a:after{
  content: "(" attr(href) " ";
}
```

Rules like this are great for print stylesheets, which are stylesheets you can write and which are automatically applied when a user prints a page. The advantage for the user is that you can hide all the navigation that a user can't follow in a printout, and use the technique above so the reader can see the URLs referenced on a page.

You can also insert incremented numerical values after repeating elements (such as bullets or paragraphs) using the `counter()` function—this is explained in much more detail in the [dev.opera.com article on CSS counters](https://dev.opera.com/article/on-css-counters/).

These selectors are not supported in IE 6 or below. Also note that you shouldn't insert important information with CSS, or that content will be unavailable to assistive technologies or if a user chooses not to use your stylesheet. The golden rule is that CSS is for styling; HTML is for important content.

CSS shorthand

Another thing you'll come across regularly in this course is CSS shorthand. It is possible to combine several related CSS properties together into one property to save time and effort on your part. In this section I will look at the available types of shorthand.

I've already used shorthand in this section without mentioning it. The `border: 2px solid black;` rule is shorthand for separately specifying `border-width: 2px;`, `border-style: solid;` and `border-color: black;`.

Comparing individual and shorthand values

Consider the following margin rule (padding and border shorthand works in the same way):

```
div.foo {
  margin-top: 1em;
  margin-right: 1.5em;
  margin-bottom: 2em;
  margin-left: 2.5em;
}
```

Such a rule could also be written as:

```
div.foo {
  margin: 1em 1.5em 2em 2.5em;
}
```

Providing less than four values for a shorthand property

A shorthand value can take less than four values according to the list below. The results are ordered by the number of values provided:

1. Same value applied to all four sides, for example `margin: 2px;`
2. First value applied to the top and bottom, second to the left and right, for example `margin: 2px 5px;`
3. First and third values applied to the top and bottom respectively, second value applied to the left and right, for example `margin: 2px 5px 1px;`
4. Values applied to the top, right, bottom, and left respective to CSS source order, as seen above.

Generally, the wisest course is to provide all four values to shorthand properties, for reasons of legibility. This advice also applies to the `padding` shorthand property.

Making the choice to use a single property or a shorthand value

Shorthand `margin` and `padding` properties tend to get the greatest share of use, though there are situations in which the shorthand properties are best avoided, or at least considered carefully, such as the following:

- **Only a single margin needs to be set.** In a situation where only one property needs to be set, the act of simultaneously setting multiple properties usually violates the KISS (Keep It Simple, Stupid) Principle, explained in the Glossary.
- **The selector to which your properties apply is subject to many edge cases.** When this happens—which it will, sooner or later—the inevitable heap of shorthand values can become hard to follow when it comes time to repair or alter your layout.
- **The stylesheet you're writing will be maintained by people whose skills (or spatial reasoning ability) are deficient.** If you can count on them to read this article you may not need to worry about this scenario, but it's best not to make any assumptions.
- **You need to supplant a value, to account for an edge case.** While this requirement is often a signal of a poorly designed document or stylesheet... those are hardly unheard of, either.

Shorthand reference

1. Border shorthand for different properties: Already explained at the very start of this section. One extra point to mention is that you can even set border properties values just for a single border of the element it is applied to like so:

```
border-left-width: 2px;
border-left-style: solid;
border-left-color: black;
```

2. Margin, padding and border shorthand for same properties: These all act in the same way; shown as seen above in the [Comparing individual and shorthand values](#) section.
3. Font shorthand: You can specify the font size, weight, style, family and line height using one line shorthand. For example, consider the following CSS:

```
font-size: 1.5em;
line-height: 200%;
font-weight: bold;
font-style: italic;
font-family: Georgia, "Times New Roman", serif;
```

You could specify all of this using the following line:

```
font: 1.5em/200% bold italic Georgia,"Times New Roman",serif;
```

4. Background shorthand: you can specify background colour, background image, image repeat and image position with one line of CSS. Take the following:

```
background-color: #000;
background-image: url(image.gif);
background-repeat: no-repeat;
background-position: top left;
```

This can all be represented using the following shorthand:

```
background: #000 url(image.gif) no-repeat top left;
```

5. List shorthand: Again, a similar story with list properties allows you to put the property values for list bullet type, position and image on a single line. Take the following CSS:

```
list-style-type: circle;
list-style-position: inside;
list-style-image: url(bullet.gif);
```

This is the equivalent of:

```
list-style: circle inside url(bullet.gif);
```

Note that #000 is a shorthand hexadecimal colour value; it is equivalent to the longhand #000000, seen earlier on. Let's look at a more complicated example too; #6c9 is the same as #66cc99.

Applying CSS to HTML

There are three ways to apply CSS to an HTML document: inline styles, embedded styles and external style sheets. Unless you have a very good reason to use one of the first two always go for the third option. The reason for this will become obvious to you soon, but first review the different options.

Inline styles

You can apply styles to an element using a `style` attribute, like so:

```
<p style="background:blue; color:white; padding:5px;">Paragraph</p>
```

Inside this attribute you list all the CSS properties and their values (each property/value pair is separated from the others by a semi-colon, and each property is separated from its value within each pair by a colon.) This is how you define styles in CSS. [Try viewing the source of this example](#) (right/Ctrl + click > Source in Opera).

If you open this example in a browser you will see that the paragraph with the `style` attribute is blue with white text and has a different size to the others, as shown in Figure 1.

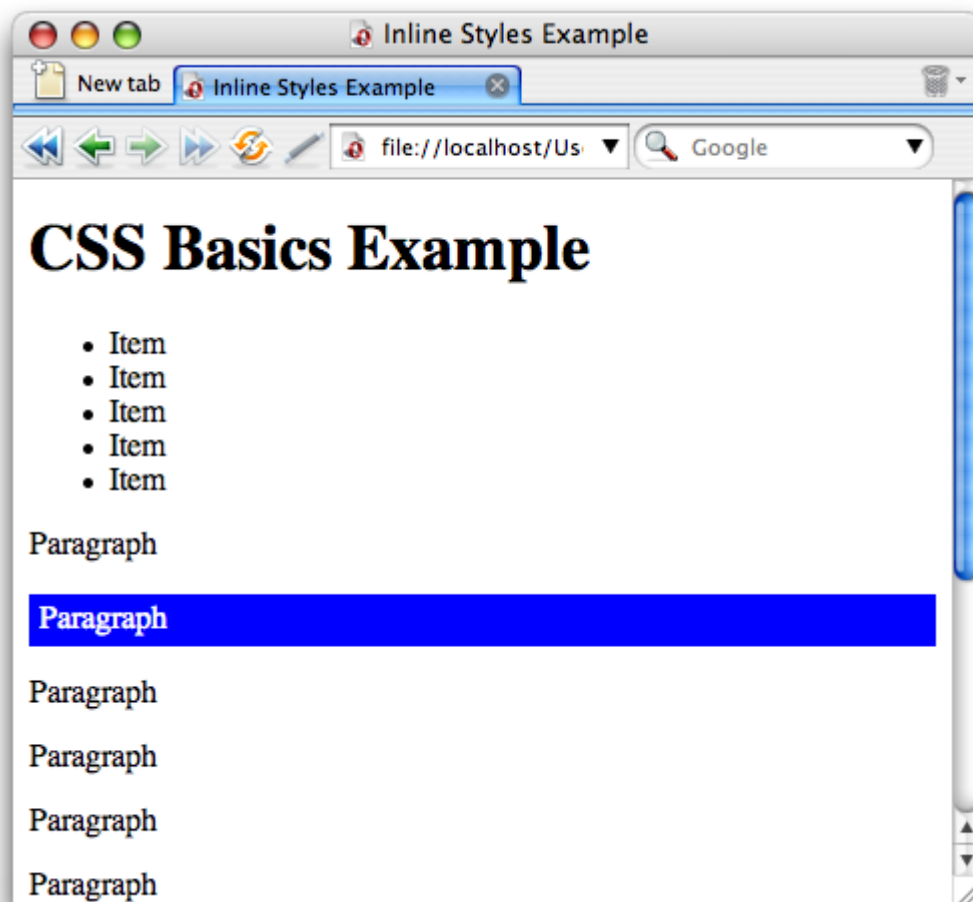


Figure 1: Opera shows the paragraph with the applied inline styles differently to the others.

The benefit of inline styles is that the browser will be forced to use these settings. Any other styles defined in other style sheets or even embedded in the `head` of the document will be overridden by these styles.

The big problem of inline styles is that they make maintenance a lot harder than it should be. Using CSS is all about separating the presentation of the document from the structure, but inline styles are doing just that—scattering presentation rules throughout the document.

In addition to the maintenance issue you don't take advantage of the most powerful part of CSS: the cascade. We'll come back to the cascade in detail in the next article, but for now all you need to know is that using the cascade means you define a look and feel in one place and the browser applies it to all the elements that match a certain rule.

Embedded styles

Embedded styles are placed in the head of the document inside a `style` element, [as in this example](#):

```
<style type="text/css" media="screen">
  p {
    color:white;
    background:blue;
    padding:5px;
  }
</style>
```

If you open the above link in a browser you'll see that the defined styles get applied to all the paragraphs in the document, as shown in Figure 2. Also try looking at the example page's source to see the CSS inside the head.

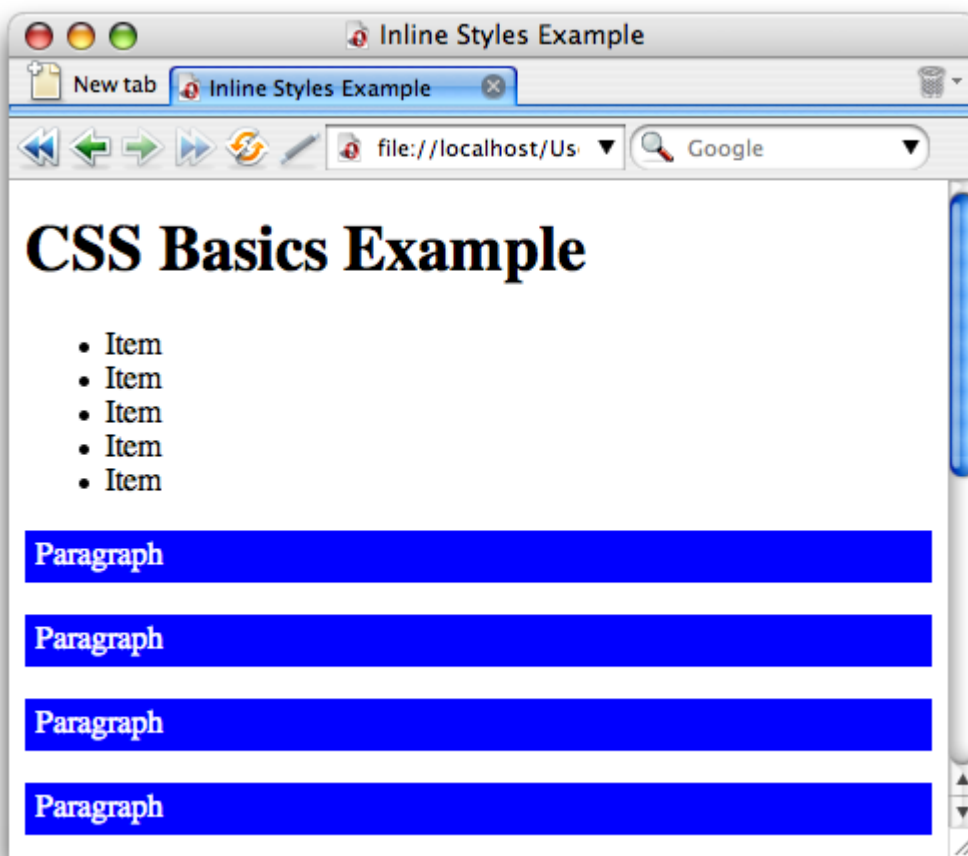


Figure 2: Opera shows all paragraphs with the styles defined in the embedded style sheet.

The benefit with embedded styles is that you don't need to add a `style` attribute to each paragraph—you can style them all with one single definition. This also means that if you need to change the look and feel of all paragraphs, you can do it in one single location, however this is still limited to one document—what if you want to define the look of paragraphs for a whole site in one go? Enter external style sheets.

External style sheets

External style sheets means putting all your CSS definitions in their own file, saving it with a file extension of CSS, and then applying it to your HTML documents using a `link` element inside the document head. View source of our [example page](#), and note that the `head` contains a `link` element that references this [external CSS file](#), and verify that the styles defined in the external CSS file are applied to the HTML. Let's have a closer look at that `link` element:

```
<link rel="stylesheet" href="styles.css" type="text/css" media="screen">
```

We've talked about the `link` element before in this course. Just to recap: the `href` attribute points to the CSS file, the `media` attribute defines which media should get these styles applied to it (`screen` in this case as we don't want a printout to look like this) and the `type` defines what the linked resource is (a file extension is not enough to determine that).

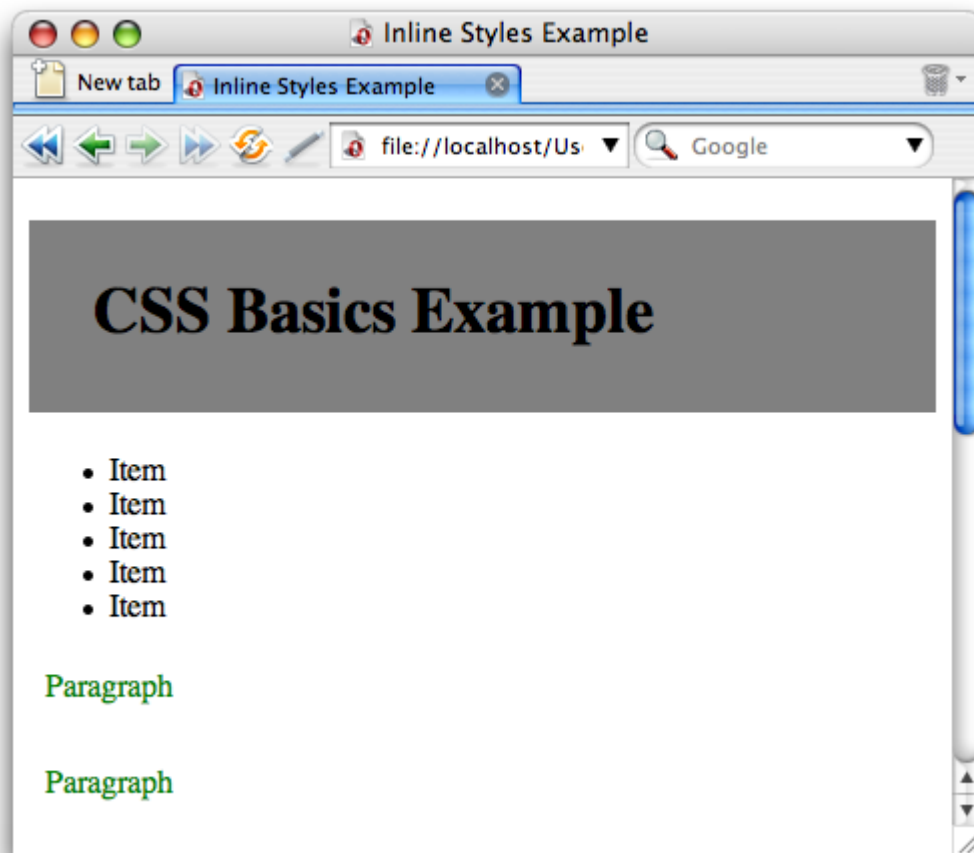


Figure 3: Opera shows the styles defined in the external style sheet when it is linked with a link element.

This is the best of all worlds: you keep your look and feel definitions all in one single file, which means that you can make site-wide changes by changing one file, and the browser can load that once and then cache it for all other documents that reference it, meaning less to download.

@importing stylesheets

There is actually another way to import external style sheets into HTML files - the `@import` property. This is inserted into an embedded style sheet, in the same way as the embedded CSS shown above. The syntax looks as follows:

```
<style type="text/css" media="screen">  
  @import url("styles.css");
```

```
...other import statements or CSS styles could go here...  
</style>
```

You'll sometimes see import statements without the brackets, but it does the same thing. Another thing to be aware of is that `@import` should always be first in an embedded style sheet. Finally, you can specify that the imported style sheet be applied only to certain types of media by including the media type at the end of the import statement (this works in every browser except IE6 and below). The following does the same thing as the previous code example:

```
<style type="text/css">  
  @import url("styles.css") screen;  
  ...other import statements or CSS styles could go here...  
</style>
```

The first question you'll be asking is "why on earth do I need another way to apply external style sheets to my HTML documents?" Well, you don't really - I am mainly including information on `@import` here for the sake of completeness. There are a few minor advantages/disadvantages of using `@import` over `link` elements, but they are *very minor*, so it's really up to you which way you go. `link` elements are the recognised best way to do things these days.

- Older browsers don't recognise `@import` at all, so completely ignore it (Netscape 4 and older, and IE 4 and older if you omit the brackets from around the filename). You can therefore use an `@import` statement to hide styles from old buggy browsers that would use them incorrectly. You could put your up-to-date styles in an external stylesheet and import them with `@import`, then provide some really basic styles that will not cause IE/Netscape 4 to choke in the embedded stylesheet. This is useful, but you'll very rarely need to ensure IE/Netscape 4 compatibility these days!
- As mentioned before, IE6 doesn't support putting the media type at the end of the `@import` line, so they are not a good way to go if you want to insert multiple stylesheets for different media.
- You could argue that the code for multiple `@import` statements is smaller than the code for multiple `link` elements, but this is pretty negligible.

Summary

In this tutorial you learnt about applying CSS to HTML documents, either as inline styles using `style` attributes, embedded styles in a `style` element in the document head or as external files in their own document. You also learnt that the latter—linking an external style sheet using a `link` element—makes the most sense in terms of maintenance and caching. We then talked about the basic syntax of CSS and explained comments, different selector types, and grouping of selectors.