CSS basics

Contents [hide]

Introduction

In the earlier tutorials of this course we talked about the content of web sites and how to structure content using HTML. This is very important as it means that we give our documents meaning and structure for other technologies to seamlessly integrate with, manipulate and otherwise affect. The most important web technology to discuss next is CSS (Cascading Style Sheets), which is used to style our HTML and influence its position on the page. In this article we'll introduce you to CSS — what it is, how to apply it to HTML, and what basic CSS syntax looks like.

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, colouring, etc. 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 (or rulesets), the exact syntax of which you'll learn more about below. These rules:

1. State what HTML elements should have styling added to them
2. Specify the "properties" (colour, size, font, etc.) of the HTML elements we want to manipulate

3. Give the values we want to give each property.

For example, a CSS rule might state:

> I want to find every `<h2>` element on a page, and change the colour of their text to green

or

> I want to find every `<p>` element with a class attribute 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. As we've discussed earlier in the course, 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, using actual element names, eg. `<body>`, or another identifier such as `class` attribute values. We’ll look at the different types of selectors available later on.
- 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 text colour, background colour, position on the page, font type, border colour and thickness and many other things.
- The values are the values that you want to set for each property of the selected elements. 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 this CSS is applied to 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.

Whitespace in CSS

Note that whitespace in CSS works in exactly the same way as it does in HTML — excess whitespace is completely ignored by the browser that renders the CSS, so you can add as much whitespace as you like. So

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

Is functionally identical to

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

As long as you include the necessary curly braces, colons and semi-colons to separate out different parts, you are ok.

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 are having on your web page; just comment them out, save your CSS, and reload the HTML.

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

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 are repeating information that is the same. To remedy this, 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;}
```

Basic types of selector

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

Element Selector

```
p {}
```

An element selector matches all the elements of that name on the page (`<p>` elements, in the case above).

Class Selector

```
.example {}
```

A 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.

ID Selector

```
#example {}
```

An 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

Combining Selectors

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

- `p.warning {}` matches all paragraphs with a `class` of `warning`.
- `div#example {}` matches the element with an `id` attribute of `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 we are using these to differentiate between the different warning styles:

- example-specificselectors.html
- specificselectors.css

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 we will look at the available types of shorthand.

For example, the `border` property is shorthand.

```
border: 2px solid black;
```

is equivalent to

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

Comparing individual and shorthand values

Consider the following margin rule:

```
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;
}
```

These types of property can take less than four values too, as follows:

1. Same value applied to all four sides — `margin: 2px;`
2. First value applied to the top and bottom, second to the left and right — `margin: 2px 5px;`.
3. First and third values applied to the top and bottom respectively, second value applied to the left and right — `margin: 2px 5px 1px;`.

There are a number of properties that work like this, including `padding`, `margin` and`outline`. You'll find more out about these later on.

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 in the following situations:

- **When 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.
- **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 with limited CSS skills (or spatial reasoning ability).** If you can get 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.** This requirement is often a signal of a poorly designed document or stylesheet, but those are hardly unheard of.

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 lets review the different options.

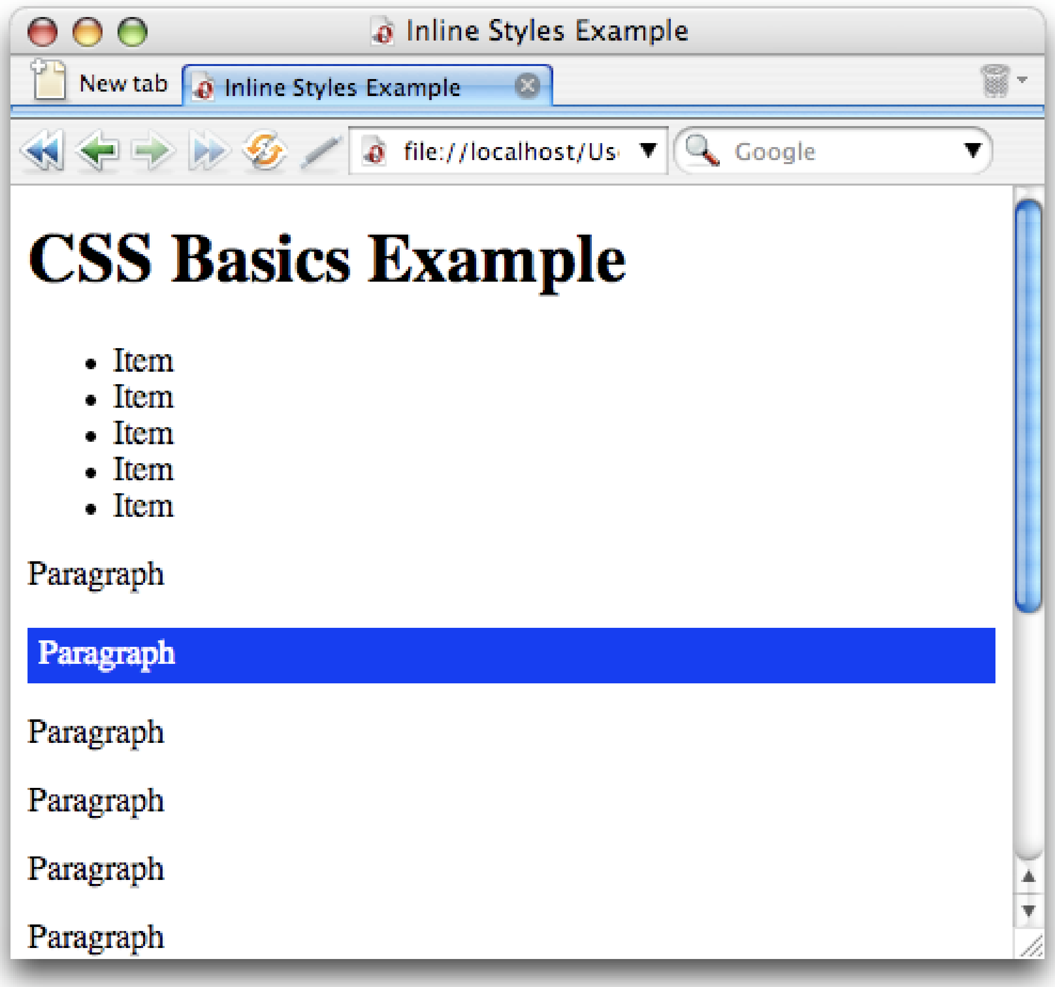
Inline styles

You can apply styles to a specific 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. Try viewing the source of this example.

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.



CSS Basics Example

- Item
- Item
- Item
- Item
- Item

Paragraph

Paragraph

Paragraph

Paragraph

Paragraph

Paragraph

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 styles defined in other style sheets or even embedded in the `<head>` of the document will be overridden by these styles.

The big problem with 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 the opposite — scattering presentation rules throughout the document.

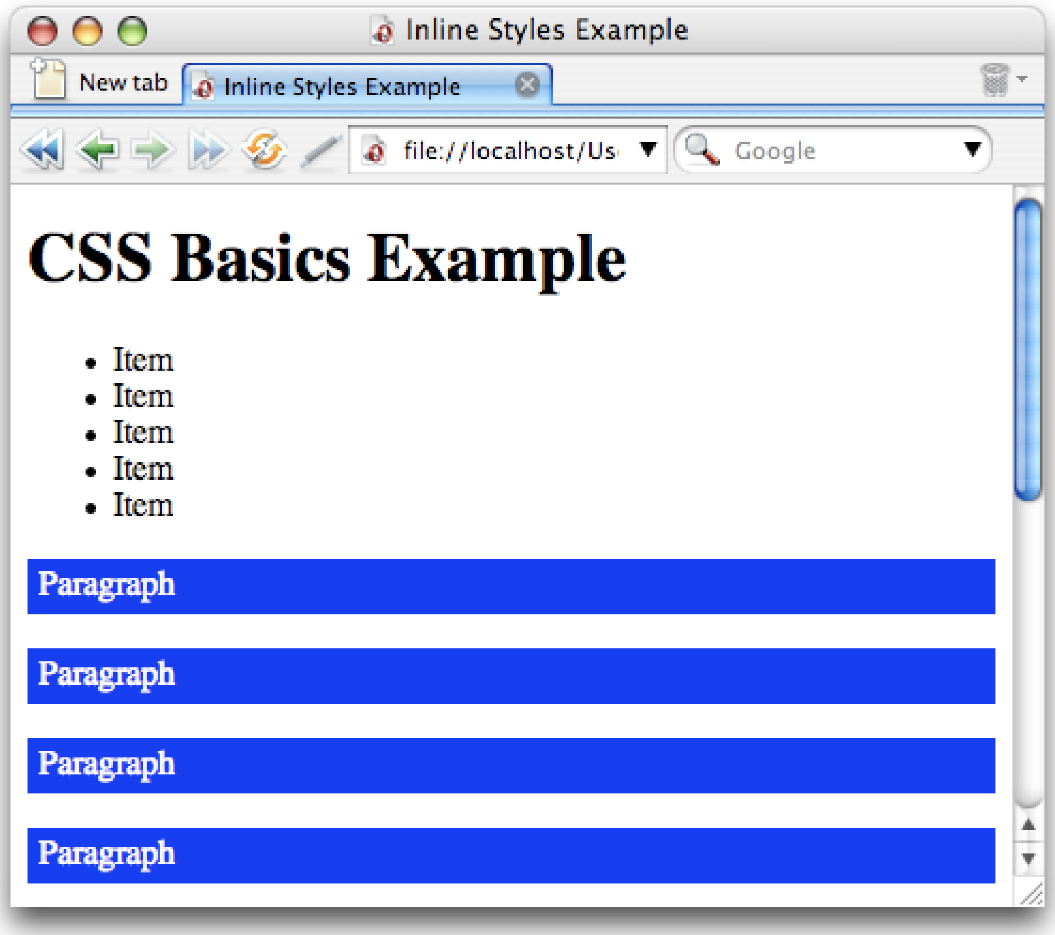
In addition to the maintenance issue you don't take advantage of the most powerful part of CSS: the cascade. We'll come back to the cascade in detail later, but for now all you need to know is that using the cascade means you define a look and feel in one place and the browser applies it to all the elements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



CSS Basics Example

- Item
- Item
- Item
- Item
- Item

Paragraph

Paragraph

Paragraph

Paragraph

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

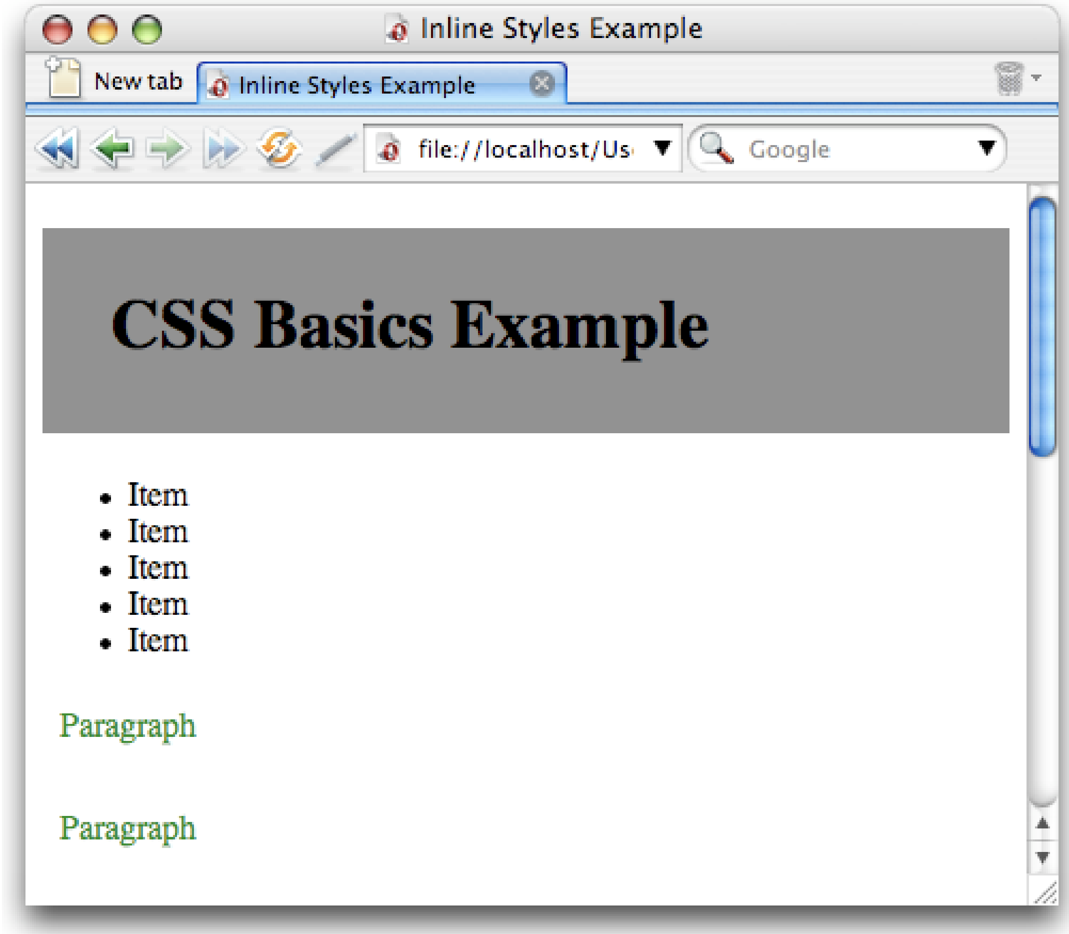
The benefit with embedded styles is that you don't need to add a `style` attribute to each paragraph — you can style them all with one single definition. This also means that if you need to change the look and feel of all paragraphs, you can do it in one single location, however this is still limited 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).



CSS Basics Example

- Item
- Item
- Item
- Item
- Item

Paragraph

Paragraph

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.

In the next tutorial we'll go deeper into the details of CSS, covering the cascade and specificity of selectors in detail.

Exercise Questions

- What are the benefits and problems of inline styles and how do you apply them to an element?
- What is a style rule? Describe the different components and syntax.
- Say you have a bunch of style rules, what do you need to do to make them into an external style sheet?
- What does the following CSS selector match: `ul#nav{}`?
- What is the benefit of grouping selectors?
- How can you define a print style sheet?