Basic Cascading Style Sheets (CSS) and JavaScript

PGCertInfoTech

Programming with Web Technologies

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Auckland ICT Graduate School

CSS (Cascading Style Sheets)

- CSS is a language for describing presentational characteristics of elements in a document (commonly an HTML web page)
- The idea is to separate description of document content/structure from description of layout/style
- HTML code defines content/structure
- CSS code defines layout/style
- A file containing CSS code is commonly referred to as a **stylesheet**

Separation of Content and Presentation

- The idea is to separate description of document content/structure from description of layout/style

This is powerful and flexible:

- We can take the same HTML and present it in different ways using different CSS stylesheets
- It is easy to modify styles across multiple pages by using stylesheets rather than defining presentation in the HTML
- Ideally your HTML documents should be understandable by a person (or program) if CSS is turned off.
- This is known as 'separation of content and presentation'

Divs and *Spans* are our Friends!

Commonly we use the HTML elements *div* and *span* so that we can apply CSS styles to a group of elements in one go:

- *div* is used to group together block level elements
- *span* is usually used for in-line elements
- look at the HTML source of pages you visit
  - divs, divs and more divs!

But not restricted to these elements:

- Can be applied to any element
- Paragraph <p> is another frequently used element that CSS gets applied to

CSS

3 main ways of using CSS:

- Inline
  - Using a style attribute
- Internal
  - in the head of the document
  - Using class attribute and element selectors
- External
  - In a separate CSS file
  - Using class attribute and element selectors

Inline CSS

Using the style attribute, format:

- css-property: value;

For example:

- some text
- some text

Available CSS 3 properties:

- http://www.w3schools.com/cssref/
- Not every property applies to every HTML element

Inline CSS

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            Inline CSS
        </title>
    </head>
    <body>
        <h1 style="color:red"> Inline CSS Example </h1>

        <p style="font-family:sans-serif; font-size:12pt; color:green;">
         This is going to be a
         new paragraph <br /> for testing CSS properties.
        </p>
    </body>
</html>
```

Inline CSS Example

This is going to be a new paragraph
for testing CSS properties.

Note the US-style spelling (color not colour)

Some Common CSS properties

- color - specifying text color.
- font-family - specifying font type.
- font-size - specifying font size.
- text-decoration - specifying text decorations, such as underline.
- font-style - specifying font styling, such as italics.
- font-weight - specifying font weight, such as bold.
- width - specifying the width of an element.
- height - specifying the height of an element.
- background - specifying the background.
- border - specifying a border.
- text-shadow - specifying a shadow for our text.
- float - specifying the float of an element, such as left or right.
- position - specifying the position of an element, such as absolute or relative.

CSS Fonts

- font-family, font-size, font-style, font-weight, font-variant

Font-family has 5 basic (or generic) options:

- serif, sans-serif, monospace, cursive and fantasy
- cursive and fantasy not used much

- A font-family can specify a number of fonts:

```
font-family: Times new roman, Georgia, serif;
```

- The browser searches left to right to find a font that it has

http://www.w3schools.com/css/css_font.asp

CSS Fonts

- font-size:`medium|xx-small|x-small|small|large|x-large|xx-large|smaller|larger|length;`
- font-style: `normal|italic|oblique;`
- font-weight: `normal|bold|bolder|lighter|number;`
- font-variant: `normal|small-caps;`

CSS Selectors

- Inline CSS specifies which content the properties apply to by using the `style` attribute
  - This leads to lots of repetition and is not efficient
  - OK for simple small cases or one-off situations

- Selectors enable more (and more powerful) CSS:
  - e.g., Text in every Paragraph element should be in 12pt serif orange

CSS Selectors

- The general pattern is:

```
selector { property1: value1; … propertyN: valueN; }
```

- Where **`selector`** is any of the HTML elements in the **`body`** of a document
  - Can't style `title` or `meta` elements

```
P {font-family: serif; font-size: 12pt; color: orange;}
```

- **`selector`** can also be a list of elements:

```
H1, H2, H3 { font-weight: bold;}
```

Internal CSS

- We can't specify these rules in `style` attributes in `body`
  - We specify these in `style` element in `head`

```
<style type="text/css">
    h1 { color: red; }
</style>
```

- All text in `H1` elements in *this* HTML page will be red
  - *Unless* there are other (Internal) rules or inline style attributes

External CSS

- Internal CSS still requires duplication in multiple documents
- Even better is to define common CSS in a dedicated external CSS file and then link it to the HTML documents in which it is used

- In the `head` of a document:

```
<link href="mystyle.css" rel="stylesheet" type="text/css">
```

- `mystyle.css` has no HTML in it – just CSS rules

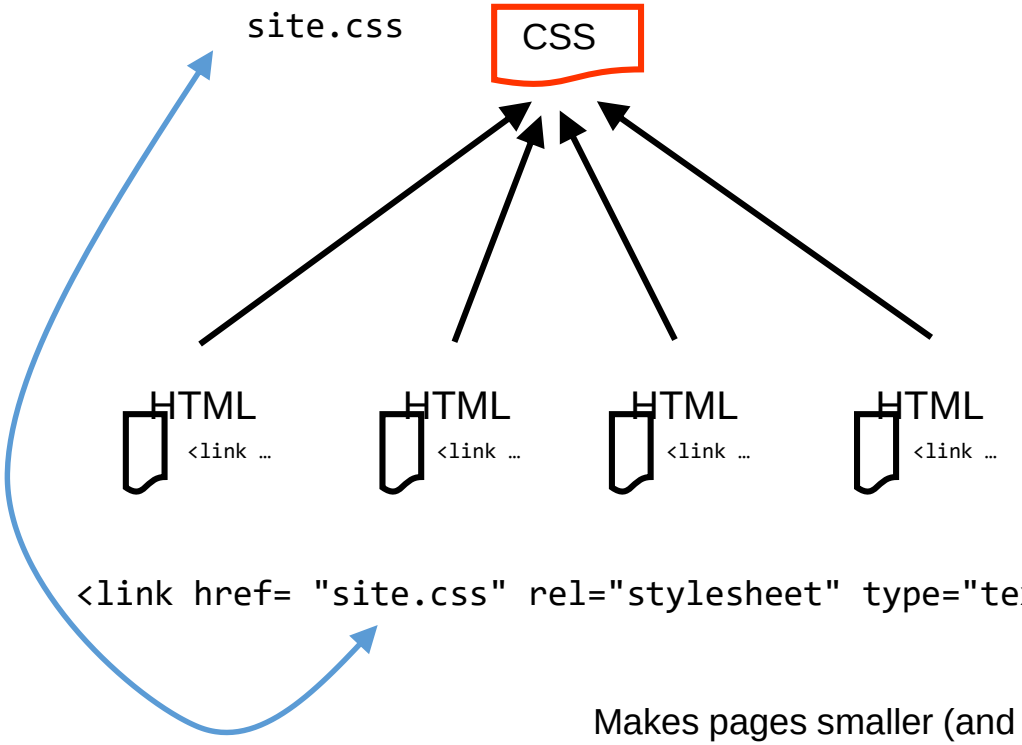
External CSS

- Easy to use the same styles across multiple pages
- To change a style modify rules in one place (the CSS file) not multiple HTML pages
  - easy maintenance
- Once a browser has downloaded a CSS file it can reuse it until it changes on the server
  - uses less bandwidth

- Can create multiple CSS files for a site to help manage the CSS rule set (rather than having all rules in a single file)
- Perhaps organise by function (general.css, forms.css) or page groups (holidays.css, pgcert-infotech.css)

External CSS → Consistency

1 file to control *many* pages



`.css` files contain only CSS
No HTML, no JavaScript

```
<link href= "site.css" rel="stylesheet" type="text/css">
```

Makes pages smaller (and faster)
Easy to maintain – just edit in one place

Spot the errors

```
h1 {    font-family: Georgia; serif;}
h1 {    font-style: italic;
        font-size: 1.5em:
        font-weight: 1400;}
p {     font-family: 'Droid Serif', Georgia, serif; }
```

Errors fixed

```
h1 {    font-family: Georgia, serif;}
h1 {    font-style: italic;
        font-size: 1.5em;
        font-weight: 900;}
p {     font-family: 'Droid Serif', Georgia, serif; }
```

(1)

(2)

(3)

JavaScript

- JavaScript (JS) most commonly used to write code that runs in a web browser as part of a web page
- Browsers contain a JS engine that interprets and executes JS code
- But JS can be used in many other places
  - server-side application coding
  - desktop widgets/gadgets
  - mobile applications
  - browser extensions
  - etc.

JavaScript: History

- The name 'JavaScript' is actually a trademark of *Oracle*
- A programming language developed in 1994-5 at Netscape Corp and used in their *Netscape Navigator* web browser
- Originally called **Mocha**, then **LiveScript**, and then **JavaScript**
- The Java language was new and hot at the time – the name **JavaScript** possibly an attempt to cash in on that
- Java and JavaScript are **NOT** formally related

JavaScript

- The non-trademarked name is **ECMAScript**
- ECMA is a standards organisation – JavaScript was submitted to it and the standard for the language was published as ECMAScript
  - *ECMA = European Computer Manufacturers Association*

- So technically, most of the time the use of 'JavaScript' is incorrect
  - it should be 'ECMAScript'

- The term 'ECMAScript' is free to use
- The term 'JavaScript' may be licensed for use
  - But in practice it isn't …
  - and 'JavaScript' is used almost all of the time

`http://en.wikipedia.org/wiki/List_of_ECMAScript_engines`

Why do we need programs in our web pages?

- Two main reasons:
  - to represent and process data **without** sending it to the server, processing it on the server, and sending back the result to the browser
  - to dynamically read/modify the content and/or presentation of the page
- For example, the user modifies the number of some item in their shopping cart (e.g. 20 blank DVDs rather than 10)
  - compute the new cost for DVDs based on the quantity and unit price
  - compute the new total price for all items in the cart
  - display the new cost for DVDs
  - display the new total price
- This involves both data processing and page modification – but achieving it via the server would be expensive in terms of time and bandwidth

Why do we need programs in our web pages?

- Web applications have matured significantly over the last few years
- Moving to a user experience comparable to desktop applications
- Supported in part by ability to write large scale and efficient programs in JS for execution in the browser
- Helped by improvements in JS engines

For example:

- Hotmail (and similar)
- Google Maps/Calendar/Docs/Drive/Sites (and similar)
- and other 'Software as a Service' solutions

Core JS features

Similar to many other languages:

- statements delimited by ; (semi-colon)
- selection with if-else
- selection with switch
- repetition with for and while
- variables for basic (e.g. integer) or complex (e.g. structured) data
- arrays
- functions/procedures with parameters and return values
- objects which store data and have operations (functions/procedures) to operate on that data
- numeric and string operations

Different in:

- how data types are dealt with (loose typing)
- the way in which object-orientation is achieved

How do we include JS code in HTML?

- The inclusion of JS code in a web page is similar to the inclusion of CSS

It can be done:

- Inline
- Internal
- External

The syntax and features are same in each case

Inline JS

- JavaScript works in response to events on the webpage
  - e.g., onclick event

```
<button type="button" onclick="alert('clicked!')">
Click me!</button>
```

- Several other types of events, e.g.
  - `focus`, `blur` (loss of focus)
  - `onmouseover` (cursor movement)
  - `onload` (page load)

http://www.w3schools.com/JS/js_htmldom_events.asp

Internal JS

```
<script type="text/javascript">
    alert("Welcome to this webpage!");
</script>
```

- JavaScript code goes inside the script element
  - The <script> element can be placed both inside the head , or in the body of the HTML document or both
  - When the browser reads the <script> element, it starts executing the code inside it (unless the code is inside a function)

External JS

```
<script type="text/javascript" src="URL_of_the_JS_file"></script>
```

- If the `src` attribute is present the script element must be empty
- The "type" attribute is required in HTML 4, but optional in HTML5
- In this case the script behaves as if it was located exactly where the <script> tag is

Displaying content on a webpage using JS

- alert() method
- document.write() method
- innerHTML
- console.log() [writes to the browser console]
- document.getElementByID("id").textContent

The <noscript> element

- Is JavaScript absolutely necessary? What if someone has it turned off?
- The `noscript` element allows you to specify some alternative content
  - Such as “this site requires JavaScript to be turned on”
- The `noscript` element can be placed either in head or in body

```
<noscript>
        Your browser has JavaScript turned off. Please turn
        Javascript on to continue using this webpage!
</noscript>
```