CSY2028
Web Programming

Tom Butler
thomas.butler@northampton.ac.uk

Week 1 - Introduction

- About me
- Module Overview
- Module Objectives
- HTML & CSS recap

About me

- Graduated from UoN in 2007
- Worked as a software developer since
- Been teaching here 2 years part time
- Working on my PhD into software development best practices
- Write infrequently online:
  - My blog http://r.je
  - Sitepoint.com

Module Overview

- Module structure
  - 20 Credits
  - 22 Weeks
    - 1 hour lecture + 1 hour practical
  - Assignments
    - One assignment due in January
    - One assignment due in April

Module Objectives

- Develop an understanding of web programming including
  - Development tools and techniques
  - Best practices
  - PHP Programming
- You should already have the following:
  - Basic programming skills
  - Basic website building skills (HTML)
  - Basic database skills (MySQL)

How do the sessions work?

- 1 hour lecture
  - Each lecture builds on the last
  - **If you miss a lecture make sure you read the notes and try the practical exercises**
  - Feel free to ask questions
  - Slides will be on NILE

How do the sessions work?

- 1 hour Practical
  - Exercises on NILE
    - Sometimes exercises will be flagged with a difficulty
    - Everyone must do the grade D-C exercises
    - Others are optional and a lot more difficult
    - If there is no grade associated with an exercise, everyone should do it
  - The exercises may take longer than an hour
  - You may work in small groups (**Except for assignments!)**
  - Ask questions

Topics Covered

1) Introduction + Basic HTML/CSS recap
2) HTML5 + Advanced CSS tips and tricks
3) Setting up a development environment for web programming
4) PHP - Introduction
5) PHP – Functions and Control Structures
6) PHP – Arrays and data structures
7) PHP – Handling User Input
8) Object-Oriented PHP Code
9) MySQL – Introduction + Connecting with PHP
10) MySQL – Manipulating databases with PHP
11) PHP, MySQL and Database Abstraction
12) PHP - Maintaining state across requests + user log ins

Provisional

Topics Covered

13) Revision session

14)Object Oriented PHP part 2

15)Object Oriented PHP part 3

16) URL rewriting with mod_rewrite, URL Routing, & Single point of entry

17) Separation of concerns + HTML Templates

18) Complex HTML forms + automation

19) MVC (Model-View-Controller)

20) Building reusable components

21) Deploying your website with Git

22) Javascript – jQuery

23) Javascript – Ajax

24) PHP – Unit Testing/Test Driven Development

Module Focus

- Learn how to build websites using PHP and MySQL
- Object-Oriented Programming in PHP
- Best Practices- Why should you structure your code in a certain way?
- Making code reusable, removing duplication of effort

Today's Lecture

- Today's lecture will cover:
  - HTML
  - CSS
  - Creating a basic web page

HTML

- Basic HTML rules:
  - All HTML files should start with **<!DOCTYPE html>** (This is the only doctype you should use, all others are now very out of date!)
  - All HTML files should have a **<html>** tag which wraps everything else
  - All HTML should have a **<body>** tag which wraps all the content
  - Most HTML files should have a **<head>** tag which contains *metadata*

HTML

- A basic HTML file looks like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
  </head>

  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

Nothing in <head>
appears on the page, but the info is
used elsewhere

HTML

- HTML is forgiving
  - You can make mistakes and the browser will try to fix it
- But you shouldn't rely on this behaviour!
  - Try to always write valid HTML
  - You can (and should!) check your HTML for errors using the W3C HTML Validator http://validator.w3.org/
    - Alternatively, most editors (e.g. Sublime) have plugins to do this inside the program which makes it trivial to do
  - If your page isn't displaying as you expect, it's probably got an error and the validator will help you find out how to fix it by telling you where to look (e.g. tag name or line number)

Useful HTML tags

- Link to another page or website
  - Click here to go to google
- Display an image:
  -
    - Always supply an alt attribute, this is used by screen readers and when the image cannot be loaded
- A paragraph of text
  - text
- Headings of various levels (h1 is the top "biggest" header)
  - <h1>Heading</h1>, <h2>Heading 2</h2>,..... <h6>Heading 6</h6>

CSS

- CSS is a separate language which is used to describe how a HTML file looks when it's opened in a browser
- CSS should be in its own file (or multiple files) rather than in the HTML file

Inline vs External Styles

- You can store CSS rules inside the HTML file
- Although this is discouraged
- You should store CSS in its own file:
  - This allows you to use the same CSS rules on multiple HTML files
  - OR present some HTML in more than one way using different CSS files

CSS

- External stylesheets are structured in the following way:

```
selector {
        rule-name: value;
        rule-name: value;
}
```

- The selector is a way of 'targetting' one or more HTML elements on the page and applying properties to it
- Each stylesheet can contain one or more selectors with rules

CSS Rules

- Each rule is used to declare the way the element should be presented. For example

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

- Targets any H1 element and sets the colour of the text to red
- *Notice the American spelling of colour without a u!*

External Stylsheets

- To attach a css file to a HTML file, save the file with a .css extension in the same directory as the HTML file and you can reference it in the HTML using a <link> tag inside the <head> tag

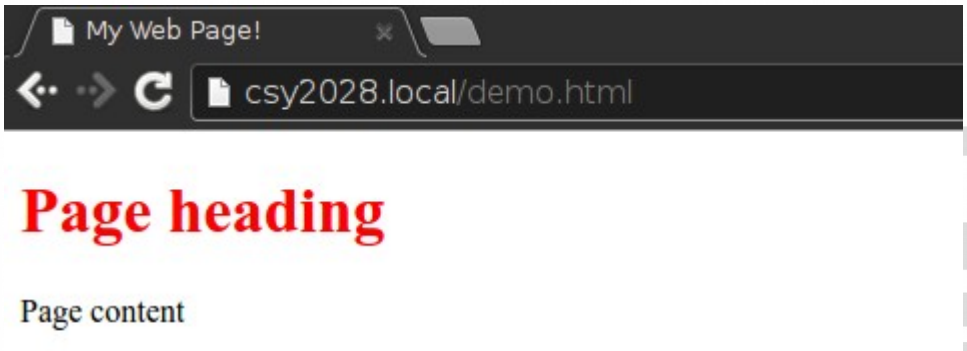
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

CSS

- Now, when you open up the page in a browser it will pull in both the HTML and CSS then apply the CSS rules to the HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

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

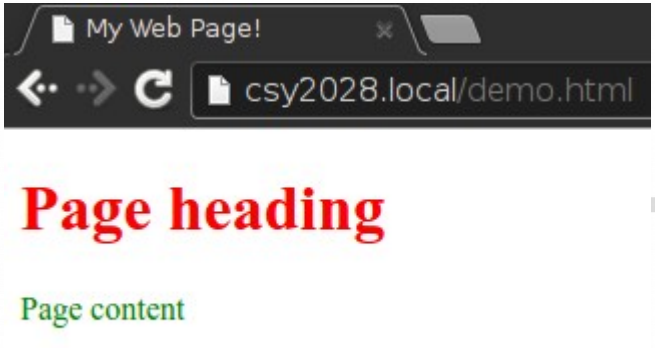


CSS

- You can add as many rules to a CSS file as you like.
- Each rule has a selector, an opening brace, some properties and a closing brace

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

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

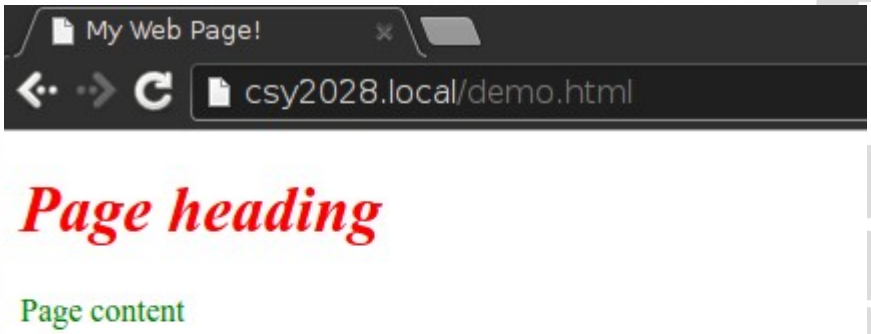


CSS

- Each rule can have one or more properties. For example:
- To change the font to italic you can use the property *font-style* with the value *italic*

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

```
h1 {
        color: red;
        font-style: italic;
}
p {
        color: green;
}
```



CSS Selectors

- There are three main css selectors
  - Element name
  - Class name
  - ID

Element Name selector

- The element name selector selects all elements with the same tag name

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Paragraph one</p>
    <p>Paragraph two</p>
  </body>
</html>
```

```
h1 {
        color: red;
        font-style: italic;
}
p {
        color: green;
}
```

Notice that both paragraphs are now green

Class name selector

- Any element can be given a *class* this is an HTML attribute like *src* in images or *href* in links
- The class name selector works in the format
  - .name
  - A dot followed by the name of the class you want to match
- This will match any element that has *class="name"*
- Note that the attribute does not include the dot in the HTML!

Class name selector

- The class name selector is a name prefixed with a dot and selects any element with a class attribute set to that value

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1 class="myclass">Page heading</h1>
    <p class="myclass">Paragraph one</p>
    <p>Paragraph two</p>
  </body>
</html>
```

```
.myclass {
        color: green;
}
```

Notice that both elements with the class "myclass" are green

ID Selector

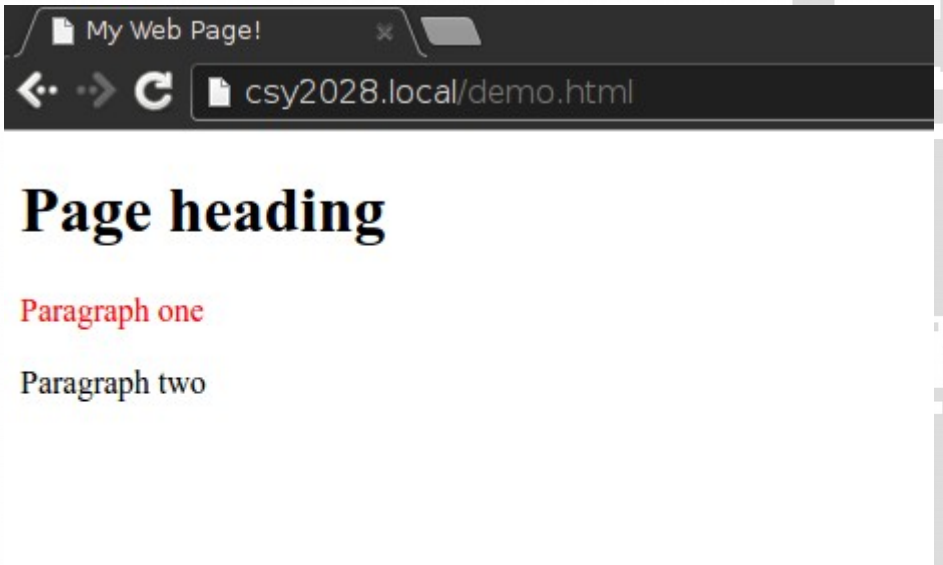
- The ID selector works very similarly to the class name selector
  - Uses a # prefix
  - #myid
  - Matches any element with id="myid" set as an attribute
- Note that the HTML does not contain the # symbol!
- IDs are different to class names. You can only use an ID once: Each ID should apply to a single element on the page

ID selector

- The ID name selector is a name prefixed with a hash and selects the element with that ID

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p id="myid">Paragraph one</p>
    <p>Paragraph two</p>
  </body>
</html>
```

```
#myid {
        color: red;
}
```



You may only have one element with each ID

Which should I use?

- In a lot of cases you can use a class name, or an element name or an ID to achieve exactly the same result!
- You should avoid IDs because you cannot easily reuse the style (if you want to make two elements red, you need to add two css rules and two IDs!)
- You should usually avoid element selectors as they're very imprecise
  - They will affect ALL elements of that type!
  - Sometimes, this is what you want
  - But if you want a specific style on a single part of the page, you might get unexpected results as the page grows and you add more elements

Combining Selectors

- HTML is a *nested* notation
  - (Some) Elements can exist inside (some) other elements
- You can use css to target nested elements. By combining a selector with a space you can target specific elements in the HTML DOM Tree
  - article h1 {font-weight: bold} – Sets any <h1> element inside an <article> element to bold
  - .myclass span {font-style: italic} – Sets any element inside an element with *class="myclass"* to italic
- This will target elements any *depth!*

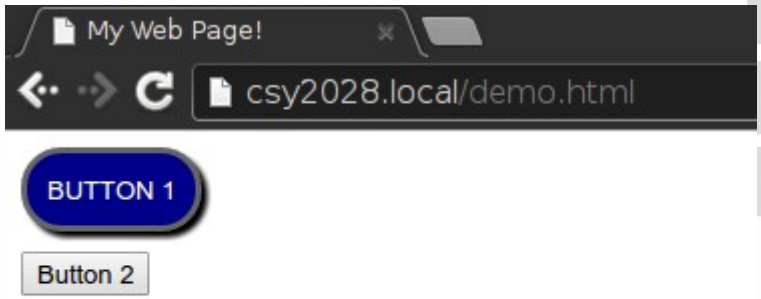
Combining selectors

- You can combine selectors to be more specific

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="myform">
      <input type="button" value="Button 1" />
    </div>

    <input type="button" value="Button 2" />
  </body>
</html>
```

```
.myform input {
  border-radius: 20px;
  border: 3px solid #666;
  box-shadow: 3px 3px 3px #000;
  margin-bottom: 10px;
  background-color: darkblue;
  padding: 10px;
  text-transform: uppercase;
  color: white;
}
```



Notice that only the button inside the element with the class *myform* has been styled

Combining selectors

- This will target elements of any *depth*

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="myform">
      <input type="button" value="Button 1" />
      <section>
        <form>
          <input type="button" value="Button 2" />
        </form>
      </section>
    </div>

    <input type="button" value="Button 3" />
  </body>
</html>
```

```
.myform input {
  border-radius: 20px;
  border: 3px solid #666;
  box-shadow: 3px 3px 3px #000;
  margin-bottom: 10px;
  background-color: darkblue;
  padding: 10px;
  text-transform: uppercase;
  color: white;
}
```

Notice that only the button inside the element with the class *myform*

Combining selecors

- You can use the *direct decedent* selector to filter to only elements that are only one level down
- The direct decedent selector uses the greater than symbol >

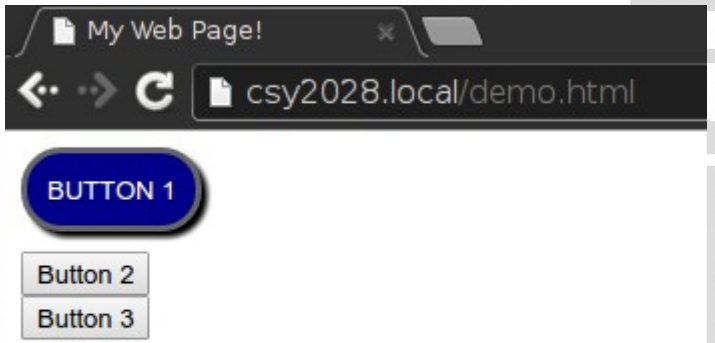
Combining selectors

- This will target elements only one level down

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="myform">
      <input type="button" value="Button 1" />
      <section>
        <form>
          <input type="button" value="Button 2" />
        </form>
      </section>
    </div>

    <input type="button" value="Button 3" />
  </body>
</html>
```

```
.myform > input {
  border-radius: 20px;
  border: 3px solid #666;
  box-shadow: 3px 3px 3px #000;
  margin-bottom: 10px;
  background-color: darkblue;
  padding: 10px;
  text-transform: uppercase;
  color: white;
}
```



Notice that only the button inside the element with the class *myform*

Combining selectors

- You can combine selectors into very complex expressions e.g.:
- `#myelement > .myclass article section header h1`
- However, this is generally discouraged
- We call this *tightly coupled* to the HTML: A slight change to the HTML will stop the h1 tag being styled
- As a rule of thumb you shouldn't need more than three levels of selector

Layouts

- CSS is used for describing how elements look on the screen:
  - Background colours, font colours, font sizes, font faces, spacing (paddings/margins)
- It's other use is describing how the page is laid out: *where* on the screen the element should appear and how large it is

Layouts

- Don't use tables for layouts!
- Back in 1999 when browser support for CSS was limited, tables were the best available choice for positioning elements on the screen
- Using tables for this purpose has been labelled *bad practice* since the early 2000s!
- Don't do it!
- I will mark you down for doing this!
- Further reading:
  - https://www.hotdesign.com/seybold/everything.html
  - http://phrogz.net/css/WhyTablesAreBadForLayout.html
  - http://webdesign.about.com/od/layout/a/aa111102a.htm
  - http://www.htmlgoodies.com/beyond/css/article.php/3642151

Layout properties

- Knowing how these 3 properties work will let you build almost any layout you want
  - Display
  - Float
  - Overflow

Display

- There are two main properties:
  - Inline
  - Block
- Inline elements will display on the same line as other inline elements
- Block elements will fill all the available width

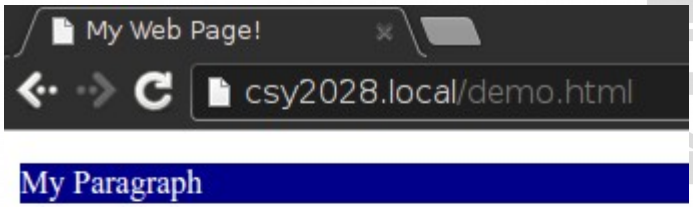
Block vs inline

- Every element has a default display property of block or inline.
- Example inline elements:
  - <img>
  -
  - <input>
- Example block elements:
  - <p>
  -
  - <h1>
- These are not fixed and can be easily overridden with CSS

Block elements

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>My Paragraph</p>
  </body>
</html>
```

```
p {
  background-color: darkblue;
  color: white;
  display: block;
}
```

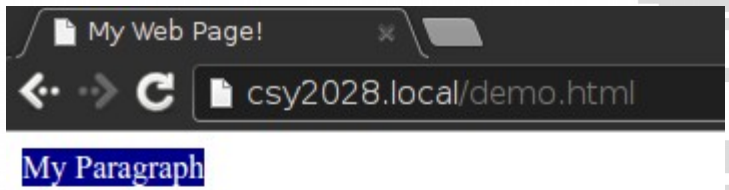


Notice that the blue background goes all the way across the page

Inline elements

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>My Paragraph</p>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: inline;
}
```



Notice that the blue background only extends to the width of the text

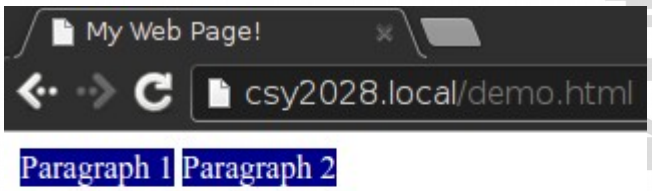
Block vs inline

- Block and inline affects how elements interact with each other
- Block elements will always be on their own line
- Inline elements will appear on the same line if there is enough space

Inline elements

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
  </body>
</html>
```

```
p {
  background-color: darkblue;
  color: white;
  display: inline;
}
```

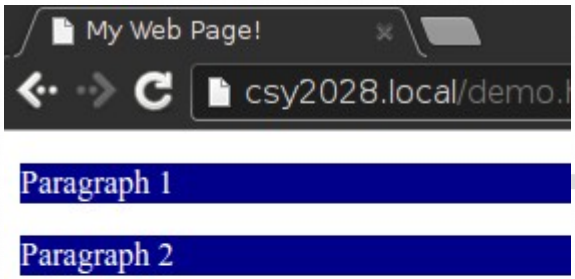


The elements appear on the same line

Block elements

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
}
```



The elements appear on the same line

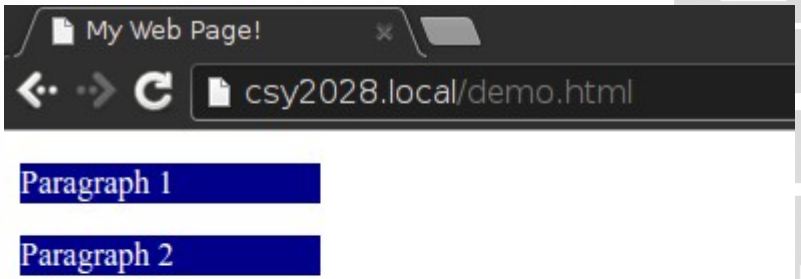
Block vs inline

- Only block elements can have a width and height!
- Setting a width and a height on inline elements will have no effect

Block elements

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
  </body>
</html>
```

```
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
}
```

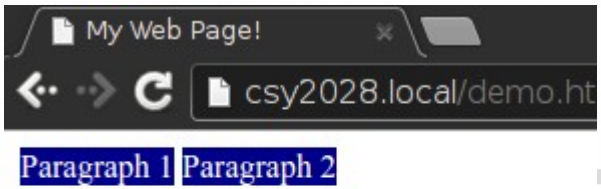


The elements appear on the same line but now don't span the entire width

Block elements

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
  </body>
</html>
```

```
p {
  background-color: darkblue;
  color: white;
  display: inline;
  width: 150px;
}
```



Float

- So how do you put elements with a fixed with/height on the same line?
- *Float* can be applied to *block* elements and have them appear on the same line but with fixed dimensions
- Float has three options:
  - None (default, no float)
  - Left
  - Right

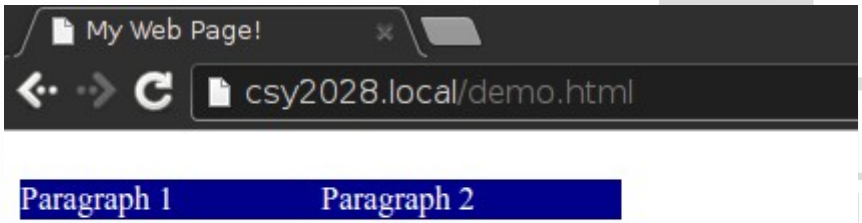
Float: left

- Float left will have the affect of stacking block elements left to right. When elements get too wide for the browser they will wrap to the next line

Float: left

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: left;
}
```



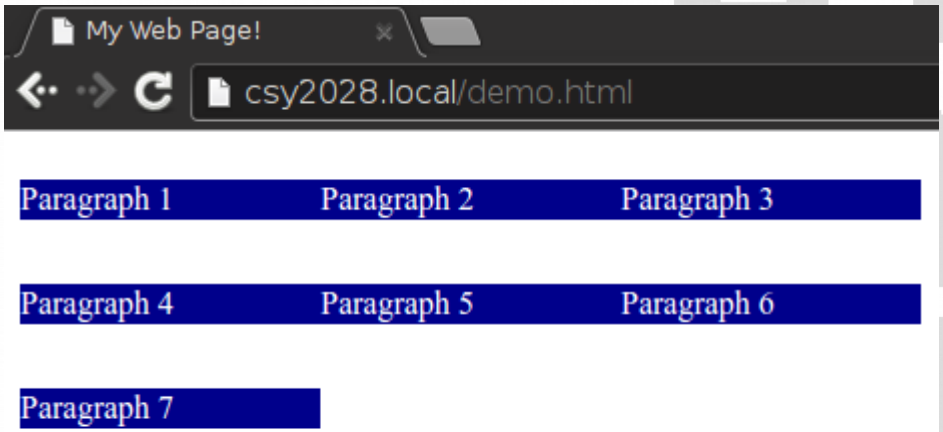
Float: left

- As more elements are added, they will be stacked left to right until there is no more room on the line

Float: left

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
     <p>Paragraph 3</p>
     <p>Paragraph 4</p>
     <p>Paragraph 5</p>
     <p>Paragraph 6</p>
     <p>Paragraph 7</p>
  </body>
</html>
```

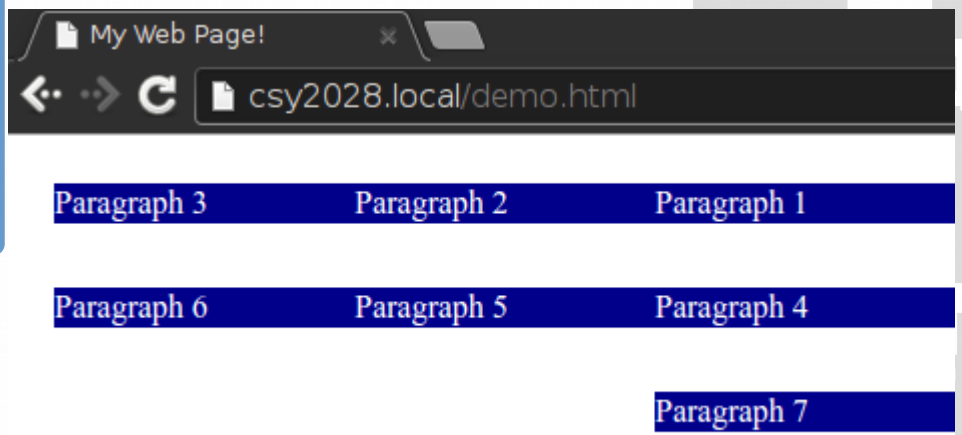
```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: left;
}
```



Float: right

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
     <p>Paragraph 3</p>
     <p>Paragraph 4</p>
     <p>Paragraph 5</p>
     <p>Paragraph 6</p>
     <p>Paragraph 7</p>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: right;
}
```



Float: right stacks the elements from right to left

Floats

- Floats have some unexpected results
- Usually when an element contains another, the background will be visible

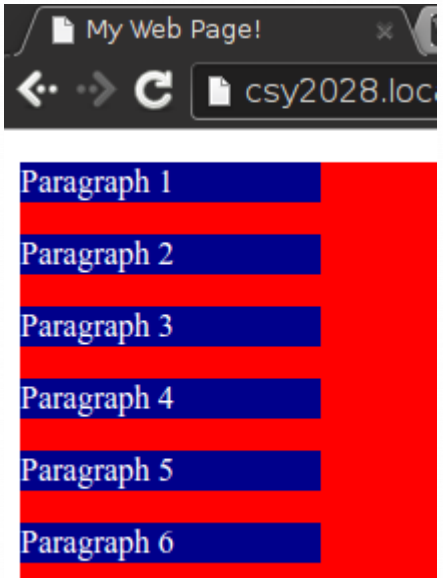
Float

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="red">
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
     <p>Paragraph 3</p>
     <p>Paragraph 4</p>
     <p>Paragraph 5</p>
     <p>Paragraph 6</p>
     <p>Paragraph 7</p>
    </div>
  </body>
</html>
```

```
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;

}

.red {
  background-color: red;
}
```



As you expect, the paragraphs sit in a container with a red background

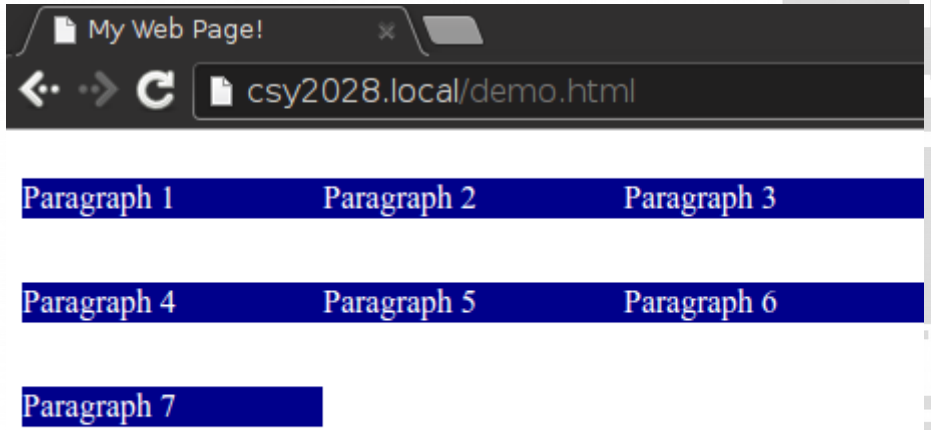
Float

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="red">
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
     <p>Paragraph 3</p>
     <p>Paragraph 4</p>
     <p>Paragraph 5</p>
     <p>Paragraph 6</p>
     <p>Paragraph 7</p>
    </div>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: left;

}

.red {
  background-color: red;
}
```



But when float: left is added,
The background disappears!

Overflow

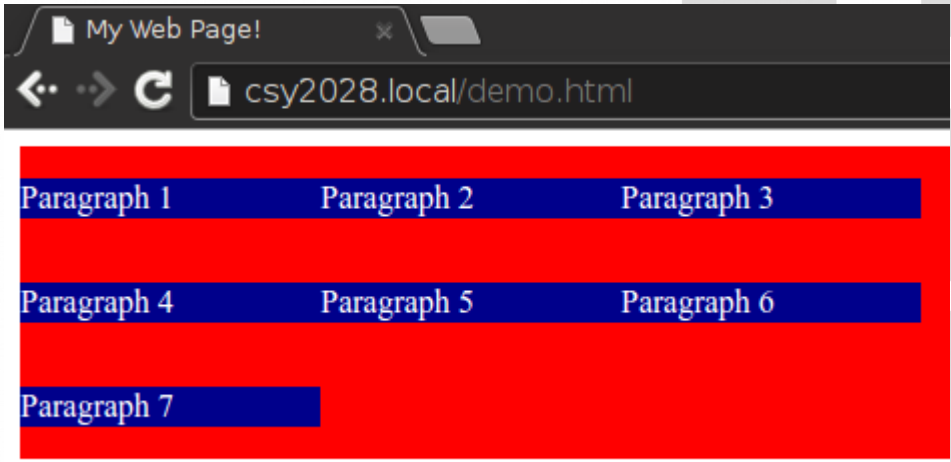
- The overflow property can be used to make the background appear
- Overflow sets whether the element should be stretched to fill floated (or otherwise moved) child elements
- By adding overflow: auto to the .red element the background will appear

Float

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="red">
     <p>Paragraph 1</p>
     <p>Paragraph 2</p>
     <p>Paragraph 3</p>
     <p>Paragraph 4</p>
     <p>Paragraph 5</p>
     <p>Paragraph 6</p>
     <p>Paragraph 7</p>
    </div>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: left;
}

.red {
  background-color: red;
  overflow: auto;
}
```



By adding overflow: auto,
The background appears

Clear: both

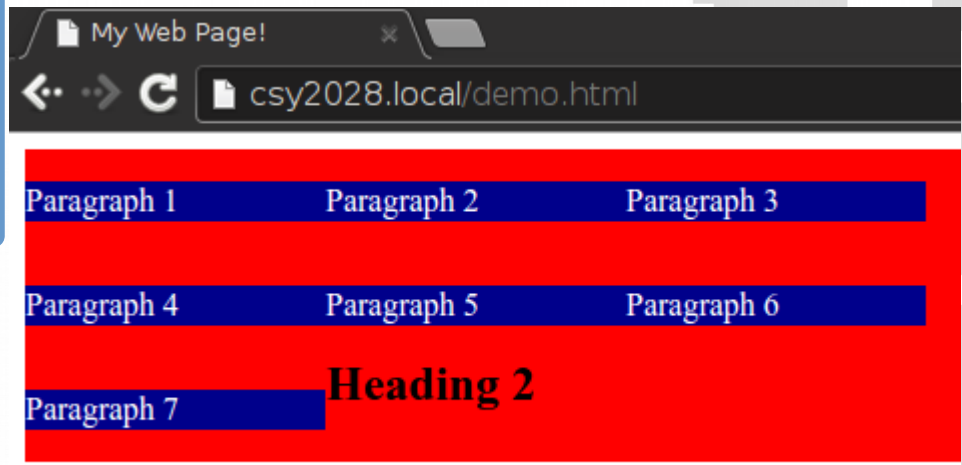
- When an element without a float is next to an element with a float, it will be floated anyway!
- To stop this, you can apply the style clear: both

Clear: both

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="red">
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
      <p>Paragraph 3</p>
      <p>Paragraph 4</p>
      <p>Paragraph 5</p>
      <p>Paragraph 6</p>
      <p>Paragraph 7</p>
      <h2>Heading 2</h2>
    </div>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: left;
}

.red {
  background-color: red;
  overflow: auto;
}
```



The H2 is floated even though it doesn't have float set

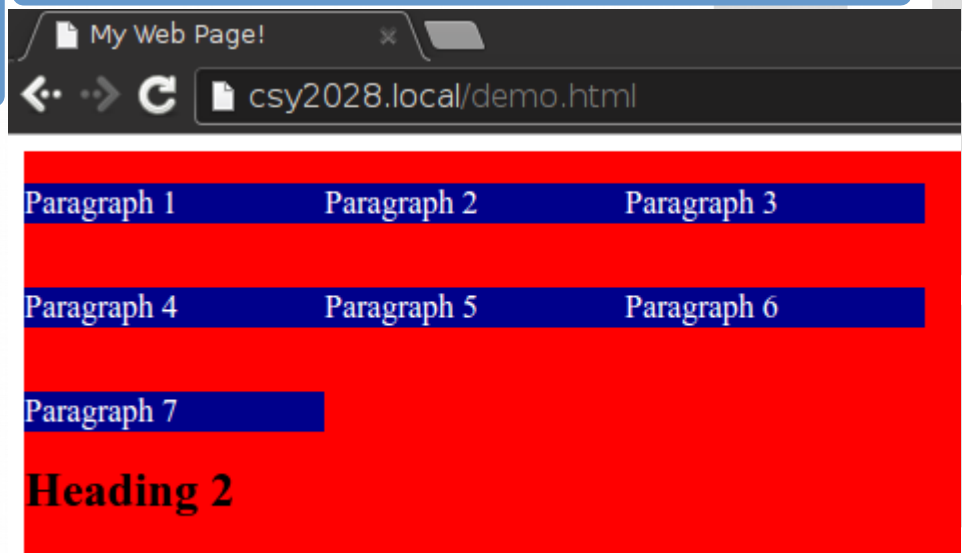
Clear: both

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="red">
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
      <p>Paragraph 3</p>
      <p>Paragraph 4</p>
      <p>Paragraph 5</p>
      <p>Paragraph 6</p>
      <p>Paragraph 7</p>
      <h2>Heading 2</h2>
    </div>
  </body>
</html>
```

```css
p {
  background-color: darkblue;
  color: white;
  display: block;
  width: 150px;
  float: left;
}

.red {
  background-color: red;
  overflow: auto;
}

h2 {
  clear: both;
}
```



By applying clear: both, the heading will not float