

ICTWEB409

Develop cascading style sheets

Learner Guide



© Copyright, 2016 by North Coast TAFEnow

Date last saved: 27 January 2016 by Sharon Lehman	Version: 1	# of Pages = 67
CHEC IT teachers – Content writer and course adviser TAFEnow Resource Development Team – Instructional and graphic design		

Copyright of this material is reserved to the Crown in the right of the State of New South Wales.

Reproduction or transmittal in whole, or in part, other than in accordance with the provisions of the Copyright Act, is prohibited without written authority of North Coast TAFEnow.

Disclaimer: In compiling the information contained within, and accessed through, this document ("Information") DEC has used its best endeavours to ensure that the Information is correct and current at the time of publication but takes no responsibility for any error, omission or defect therein. To the extent permitted by law, DEC and its employees, agents and consultants exclude all liability for any loss or damage (including indirect, special or consequential loss or damage) arising from the use of, or reliance on, the Information whether or not caused by any negligent act or omission. If any law prohibits the exclusion of such liability, DEC limits its liability to the extent permitted by law, to the re-supply of the Information.

Third party sites/links disclaimer: This document may contain website contains links to third party sites. DEC is not responsible for the condition or the content of those sites as they are not under DEC's control. The link(s) are provided solely for your convenience and do not indicate, expressly or impliedly, any endorsement of the site(s) or the products or services provided there. You access those sites and use their products and services solely at your own risk.

Contents

Getting Started	i
About this unit	i
Elements and performance criteria.....	i
Icon Legends.....	ii
Topic 1 – Define styles	1
Identify and document HTML that can be controlled by the CSS	2
Multiple styles will cascade into one	3
Cascading order.....	7
The CSS structure	9
Summary	14
Topic 2 – Produce CSS	15
CSS positioning.....	15
CSS display property.....	18
Positioning properties.....	21
Floating.....	27
CSS selector relationships.....	32
Creating CSS in a document lesson	34
Topic 3 - Validate CSS	61

Getting Started

About this unit

This unit describes the skills and knowledge required to develop cascading style sheets (CSS) that are attached to a markup language document, in order to externally define, and control, styles to enhance and achieve commonality between web documents.








It applies to individuals who are required to layout and ensure consistency of appearance between web pages.

Elements and performance criteria

Elements define the essential outcomes of a unit of competency. The Performance Criteria specify the level of performance required to demonstrate achievement of the Element. They are also called Essential Outcomes.

Follow this link to find the essential outcomes needed to demonstrate competency in this Unit: <http://training.gov.au/Training/Details/ICTWEB409>

Icon Legends

	<p>Learning Activities</p> <p>Learning activities are the tasks and exercises that assist you in gaining a clear understanding of the content in this workbook. It is important for you to undertake these activities, as they will enhance your learning.</p> <p>Activities can be used to prepare you for assessments. Refer to the assessments before you commence so that you are aware which activities will assist you in completing your assessments.</p>
	<p>Case Studies</p> <p>Case studies help you to develop advanced analytical and problem-solving skills; they allow you to explore possible options and/or solutions to complex issues and situations and to subsequently apply this knowledge and these newly acquired skills to your workplace and life.</p>
	<p>Discussions/Live chat</p> <p>Whether you discuss your learning in an online forum or in a face-to-face environment discussions allow you to create and consolidate new meaningful knowledge.</p>
	<p>Readings (Required and suggested)</p> <p>The required reading is referred to throughout this Learner Guide. You will need the required text for readings and activities.</p> <p>The suggested reading is quoted in the Learner Guide, however you do not need a copy of this text to complete the learning. The suggested reading provides supplementary information that may assist you in completing the unit.</p>
	<p>Reference</p> <p>A reference will refer you to a piece of information that will assist you with understanding the information in the Learner Guide or required text. References may be in the required text, another textbook on the internet.</p>
	<p>Self-check</p> <p>A self-check is an activity that allows you to assess your own learning progress. It is an opportunity to determine the levels of your learning and to identify areas for improvement.</p>
	<p>Work Flow</p> <p>Shows a logical series of processes for completing tasks.</p>



Topic 1 – Define styles

Markup languages are designed for the processing, definition and presentation of text. The language specifies code for formatting, both the layout and style, within a text file. The code used to specify the formatting are called tags. HTML is an example of a widely known and used markup language

The idea and terminology evolved from the "marking up" of manuscripts, i.e. the revision instructions by editors, traditionally written with a blue pencil on authors' manuscripts.

A well-known example of a markup language in widespread use today is HyperText Markup Language (HTML), one of the document formats of the World Wide Web. HTML is mostly an instance of SGML, Standard Generalised Markup Language (though, strictly, it does not comply with all the rules of SGML) and follows many of the markup conventions used in the publishing industry.

Markup is normally excluded from the version of the text which is displayed for end use. Some markup languages, like HTML have presentation rules, meaning their specification prescribes how the structured data is to be presented, but other markup languages, like XML, have no predefined rules.



ADDITIONAL RESOURCES

RECOMMENDED 1

For more information on document markup languages visit:

- > http://en.wikipedia.org/wiki/Markup_language
- > http://en.wikipedia.org/wiki/List_of_document_markup_languages

Identify and document HTML that can be controlled by the CSS

Cascading Style Sheets (CSS) is a style sheet language used to describe the look and formatting of a document written in a markup language. The most common application of CSS is to style web pages written in HTML and XHTML, but the language can also be applied to any kind of XML document, including plain XML, SVG and XUL.

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, colours, and fonts. This separation can:

- > improve content accessibility,
- > provide more flexibility and control in the specification of presentation characteristics,
- > enable multiple pages to share formatting and
- > reduce complexity and repetition in the structural content (such as tableless page design).

CSS can also allow the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices. It can also be used to allow the web page to display differently depending on the screen size or device on which it is being viewed.

While the author of a HTML document typically links that document to a CSS style sheet, readers can use a different style sheet, perhaps one on their own computer, to override the one the author has specified.

CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities or weights are calculated and assigned to rules, so that the results are predictable.

Multiple styles will cascade into one

You can use style sheets in 3 ways:

- 1 Inline
- 2 Internal
- 3 External

Let's look at the times when you would use each of these CSS techniques.

Inline

An **inline** style is applied directly to an HTML tag. There may be times when you need to overwrite a specific style in only one area of a page. This could apply to highlighting text or changing the look of a piece of text.

Below are a couple of examples of how something could be done without using CSS and by using inline CSS.

Table 1

How you want something to look	Without CSS	With inline CSS
A paragraph where the text is bold and italicised	<pre><p><i>Words</i></p></pre>	<pre><p style="font:bold italic;">Words</p></pre>

How you want something to look	Without CSS	With inline CSS
Put an image on the right side of the window	<pre> <table width="100%" border="0" cellspacing="5" cellpadding="5"> <tr> <td width="48%">&nbsp;</td> <td width="52%" align="right"></td> </tr> </table> </pre>	<pre> </pre>

Using the inline method ONLY applies to that instance where the `style=" "` has been applied and does not carry over to other tags of the same kind on that page.

Internal

You can define styles in the **HEAD** section of an HTML document and then apply those styles to the various headings, images and other page elements throughout the document. This is sometimes known as an **embedded** style sheet. Internal CSS is often used when you want one page to have a unique style. This is more efficient than using inline styles (but still not the best way - as you will see further on).

Embedded style sheets are created by inserting the **style** element into the **head** section of a HTML document. CSS code for setting a page's default font by redefining the <body> tag would look like this.

```
<style type="text/css">
body {
    font-family: Verdana, Geneva, sans-serif;
    font-size: 12px;
}
</style>
```

External

By using an external style sheet, a separate document that holds CSS style information, you can make sure that text appearance is consistent across multiple pages. Not only does a consistent appearance look more professional, but usability experts say that site visitors find what they're looking for more easily on sites where text appearance is consistent.

The code for linking from a HTML page to an external style sheet is placed between the **HEAD** tags of the page and will look something like this:

```
<link rel="stylesheet" href="styles/style.css">
```

This line basically says "insert a link from this page to the style sheet called "styles.css" inside the "styles" folder. Notice that the style sheet has a filename extension of ".css". A CSS document can be created in a simple text editor (such as Notepad) and needs to be saved with the correct ".css" extension.

If you insert this same style sheet link into the head section of every one of the HTML pages in your website, you will be able to update the appearance of your pages from a single central document, "styles.css".

Here is an extract from an external style sheet:

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

```

body {
    font-size: 62.5%;
    font-family: Verdana, Arial, Helvetica, sans-serif;
}

h1, h2, h3, h4 {
    font-family: Georgia, Times New Roman, Times, serif;
}

p, li, th, td {
    font-size: 1.3em;
}

li p, li li, th p, th li, td p, td li {
    font-size: 100%;
}

img, table {
    border: 0;
}

table {
    width: 100%;
}

```

Notice that this sample covers the paragraph `<p>` tag and heading levels 1, 2 and 3 (`<h1>`, `<h2>`, `<h3>`). The body rule ensures that no matter what text is on screen (e.g. table cells, lists, etc.) it will appear in the same font. Also note that there are three possible fonts listed in each font family declaration. This is done in case the user's machine does not have the first font installed.

Multiple external style sheets can be referenced inside a single HTML document. This can be done two ways:

- 1 Creating multiple `<link>` commands as shown above in the external example

```
<link rel="stylesheet" href="styles/style.css">

<link rel="gridlayout" href="styles/grid.css">

<link rel="typography" href="styles/typography.css">
```

- 2 Creating a single linked style sheet that contains multiple **@import** commands on each line shown below. When using this method you shouldn't include any other CSS in the document only the links.

```
/* reset styling */
@import url(reset.css);

/* Grid Layout*/
@import url(grid.css);

/* Basic Typography */
@import url(typography.css);
```

Cascading order

What style will be used when there is more than one style specified for an HTML element?

Generally speaking all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:

- 1 Browser default
- 2 External style sheet
- 3 Internal style sheet (in the head section)
- 4 Inline style (inside an HTML element)

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the `<head>` tag, which overrides a style defined in an external style sheet, or in a browser (the default value).

Note: If the link to the external style sheet link is placed after internal CSS in the HTML `<head>`, the external style sheet will override the internal style sheet!

CSS can control **ALMOST ANY** part of a markup language document from the default font, to how an image can be displayed, such as transparency, size, borders. Anything on an HTML document can be moved, changed or manipulated somehow using CSS. However, CSS cannot control nor have any effect on tags shown in Table 2.

Table 2

Tag	What it does
<code><head></code>	Defines information about the document
<code><title></code>	Defines the title of a document
<code><base /></code>	Defines a default address or a default target for all links on a page
<code><link /></code>	Defines the relationship between a document and an external resource
<code><meta /></code>	Defines metadata about an HTML document
<code><script></code>	Defines a client-side script
<code><style></code>	Defines style information for a document

The CSS structure

A CSS document contains a list of CSS rules. A CSS rule has three main parts:

- 1 a **selector**,
- 2 multiple or single **properties** and
- 3 a corresponding **value**.

The rule structure is explained below:

Syntax ----> **Selector** {[**property**: **value(s)**;]}

The standard layout when entering CSS is:

```
Selector {  
    Property1: value;  
    Property2: value;  
}
```

For example:

```
body {  
    font-size: 62.5%;  
    font-family: Verdana, Arial, Helvetica, sans-serif;  
}
```

Selectors

The selector determines which elements the property/value pairs apply to. Some features of selectors are listed in Table 3.

Table 3

Selector name	Selector	Example	Features
Universal	*	*	Applies a style to every element
Id	#	#footer	Author-defined scope label which must be unique on a page.
Class	.	.link	Author-defined label to restrict scope of style application. Multiple classes may be applied to a single element by a space-separated list
Tag	< >	<p>	Every HTML tag can take styles except <head>
Multiple	,	h1,h2,h3	Comma-separated list of selectors
Compound		div#foote r	A tag with a class or id restricting it
Contextual		.link p or div .intro	Space-separated list of selectors specifying nesting of elements and applying to any descendant. NB. div .intro is different from div.intro.

Selectors can also be used with modifiers, for example:

- > pseudo-elements -> :first-line, :first-letter, :before, :after
- > pseudo-classes -> :link, :visited, :active, :hover, :focus, :first-child, :lang
- > child -> restricts scope to first level rather than any descendent. i.e. div > p
- > adjacent -> applies to the second of two neighbouring elements. i.e. div + p
- > attribute -> restricts scope to elements with a particular attribute. i.e. div[lang] or div[lang="de"].

Property

The property is a characteristic, e.g. color or margin, which we wish to control. Note:

- > Shortcut properties take multiple values to reduce redundancy
- > Some properties (e.g. font-family) may take a list of values

Value

Each property has defined values. They may be categorised as:

- > Keyword - a name selected from a list of possible names. i.e. `auto`
- > Length - measured in `em`, `ex`, `px`, `pt`, `pc`, `cm`, `mm`, or `in`. Some properties such as `padding`, may be negative.
- > Percentage - % of current value. i.e. `width: 75%;`
- > URL - parentheses-enclosed web address, either absolute or relative to the location of the CSS file. i.e. `url(grid.css);`
- > Color - color name (not reliable except for basic 16, e.g. `white`) or RGB value expressed as:
 - > Hexadecimal - `#1a3b5c` or `#78f` (= `#7788ff`)
 - > Decimal - `rgb(120, 12, 150)`
 - > Percentage - `rgb(45%, 60%, 89%)`

When creating a CSS rule there are a couple of things to note:

- > A class overrides a tag
- > A more specific rule overrides a less specific one. This generally relates to compound selectors. i.e. `h4.subheading` overrides `h4`.

Inheritance

Most properties inherit into descendants. If you specify `body { font-family: Verdana, sans-serif; }` all text in the document will follow that rule unless overridden by a more specific rule, such as `h1 { font-family: Georgia, serif; }`.

Inheritance eliminates the need for repeated specifications of many properties within class rules. In fact, many classes can simply disappear because higher level rules prevail. For instance, many people add a class to every paragraph within a `div` when the class could be applied to the `div` instead.

Normally `font-size` inherits, but it does not inherit into table headers or cells and has to be specified separately or combined using code such as:

```
p, th, td {  
    font-size: 1.3em ;  
}
```

Comments

Comments within CSS use the syntax:

```
/* comment */
```

Comments may cross multiple lines and they must be closed. Failure to close a comment has unpredictable consequences.

```
<!--Comments in a document can also look like this -->
```

!important

Properties can be labelled “`!important`” so as to alter the normal cascade. This is especially useful for user-defined style sheets. An unstyled page will receive styles from a user-defined style sheet, but a page with styles will apply those because they appear later. A user who wants particular features should mark their style sheet

```
color: #ff0000 !important;
```

or equivalent. This is often used where there are browser issues. Careless use of this feature can make a page unreadable or have unpredictable effects.

Whenever possible, redefine existing tags instead of creating new CSS styles. The existing tags provide a useful hierarchy for both you and the users of your site, and they also make it easier for people using screen readers to use the site.



(a) Research the following links:

- > <http://www.cssbasics.com/>
- > <http://www.w3schools.com/css/default.asp>
- > <http://www.adobe.com/devnet/dreamweaver/css.html>

(b) For the following CSS properties give an example of the usage of each property:

- i. `text` : - there are 13 properties
- ii. `font` : - there are 6 properties
- iii. `margin` : - there are 5 properties
- iv. `padding` : - there are 5 properties
- v. `border` : - there are 20 properties

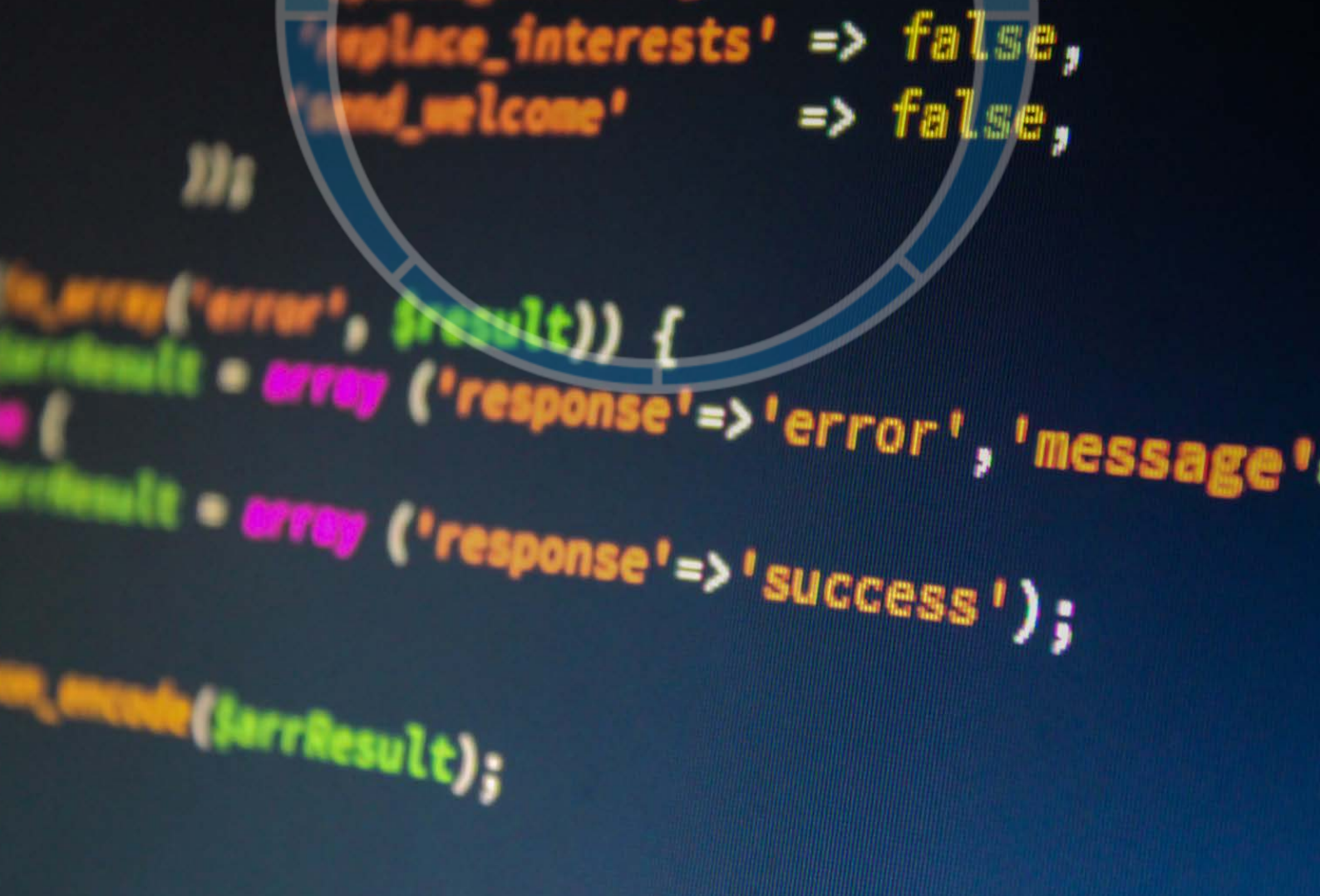
(c) Explain what the following CSS code means:

- i. `padding: 25px 50px 75px 100px;`
- ii. `margin: 25px 50px 75px;`

Summary

In this topic you have learned how to identify and document appropriate styles that are to be controlled by the CSS including:

- > What the term cascading means and the cascading order
- > The syntax of a CSS rule
- > What the cascading order is
- > What the term inheritance is
- > The 3 ways to use style sheets
- > How to comment in a style sheet
- > What !important is and how it should be used
- > How to express colour in a style sheet
- > How to import multiple style sheets using a single style sheet
- > What the 8 kinds of selector types are.



Topic 2 – Produce CSS

CSS positioning

To use CSS for layout effectively, it helps to know how it's used to position page content. This learner guide will give you an overview of the methods and rules that govern visual rendering in the CSS2 specification. It also points out some things to watch out for.

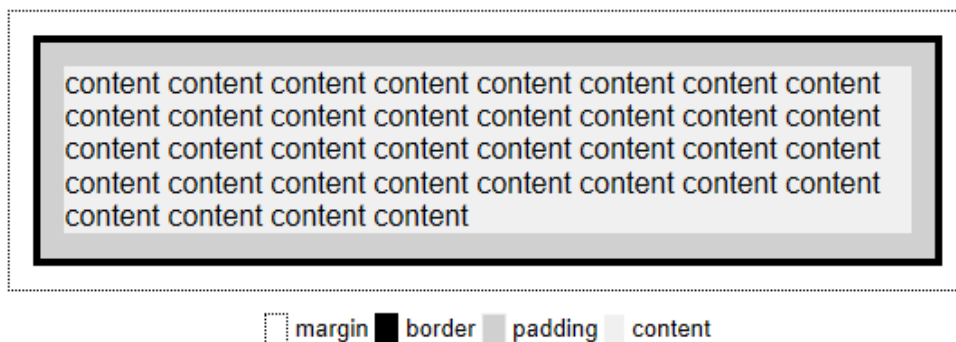
Although the specification applies to any device for displaying web pages, this article focuses on how it works in browsers. Many details are left out for the sake of simplicity.

It is important to remember that a given browser may not support a given feature or may even implement it incorrectly. Also, there is some leeway provided within the standards where individual browsers are free to deal with situations as they please.

The Box Model

To understand positioning in CSS you must first understand the box model. For display purposes, every element in a document is considered to be a rectangular box which has a content area surrounded by padding, a border and margins. Figure 1 shows these various parts.

Figure 1 – how an element is perceived



Margins are always transparent. Borders come in various styles. Background settings for an element apply to the area just inside the borders which includes both the padding and content areas. For the purposes of Figure 1 however, the padding area is shown in a slightly darker colour just to show how it is used.

When referring to boxes throughout these notes, the term "margin edge," "border edge", etc. means the outer boundary of the corresponding box area as shown above.

Margins, borders and padding are all optional but for the purposes of calculating positions and sizes they are given a default width of zero if not specified. Different widths can be set for each individual side (top, right, bottom and left) if desired. Margins can even have negative values.

The width and height of each box is equal to the width and height of the outer margin box. Note that this is not necessarily the same as the width and height of the content area.

A box can contain any number of other boxes, creating a hierarchy of boxes that corresponds to the nesting of page elements. The browser window serves as the root element for this hierarchy.

Box Types

There are two basic types of boxes:

- 1 Block
- 2 Inline

Block boxes are generated by elements such as `<p>`, `<div>` or `<table>`.

Inline boxes are generated by tags such as ``, `<i>` or `` and actual content like text and images.

The box type may also be set using the `display` property. Setting a value of **block** on an inline element, for example, will cause it to be treated as a block element.

If you set the display to **none**, no box is created. The browser acts as if the element did not exist. Likewise, any nested elements are ignored as well, even if they specifically declare some other display value.

There are other types of boxes which apply to special elements like lists and tables. However, these are treated as block or inline boxes for positioning purposes. These other box types won't be covered in this learner guide.

Containing Blocks

Block boxes act as a **containing block** for any boxes within them. For example, in this code:

```
<div>
  This is a DIV.
  <p>This is a paragraph.</p>
</div>
```

The `<div>` element establishes a containing block for both the first string of text and the `<p>` element. The `<p>` element in turn creates a containing block for the second text string.

This containing block is used in determining both the position of the boxes within it and in some cases, the dimensions of those boxes. For example, if an element has a style setting of `width:50%;` the width of the element will be set to half the width of its containing block.

For any element that is not absolutely positioned, the containing block is considered to be the content edge of its most recent, block-level ancestor. If none exists, the browser window serves as the containing block. Absolutely positioned elements are discussed in an upcoming section.

CSS display property

To use CSS for layout effectively, it helps to know how it's used to position page content. The display property specifies if/how an element is displayed, and the visibility property specifies if an element should be visible or hidden. Let's look at five display properties:

- > block
- > inline
- > none
- > inline-block
- > list-item

display: block; means that the element is displayed as a block, as paragraphs and headers have always been. A block has some whitespace above and below it and tolerates no HTML elements next to it, except when ordered otherwise (by adding a float declaration to another element, for instance) as shown in Figure 2.

Figure 2 - divs stacked using the display: block property

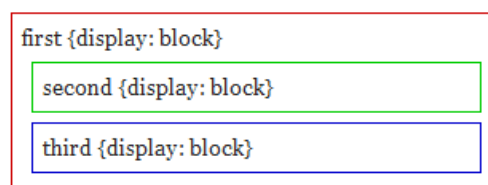


Figure 3 - display:block divs and display:inline divs together

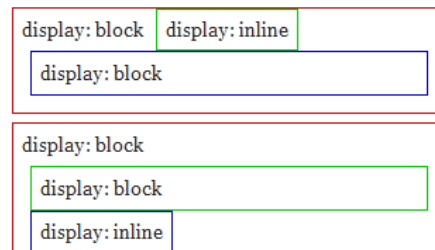


Figure 4 – Three divs, one with `display:none`; shown

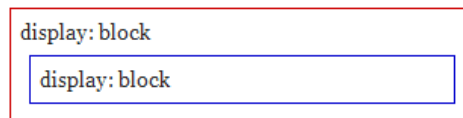
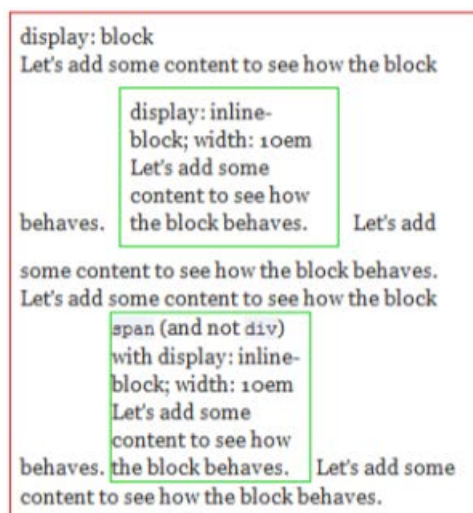


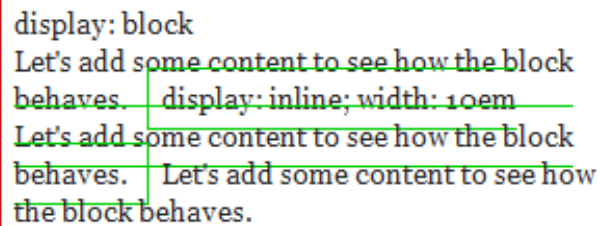
Figure 5 – display:inline-block div in the middle of a paragraph



The real use of this value is when you want to give an inline element a width. In some circumstances some browsers don't allow a width on a real inline element, but if you switch to `display: inline-block` you are allowed to set a width.

Here's the same example but using only `display: inline`. Here, the inner element does not form a block at all but gets its dimensions from the outer block and the way the text wraps.

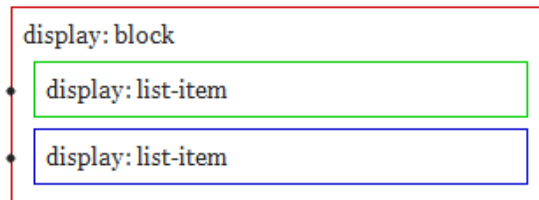
Figure 6 - `display: inline` is used and is just slotted in after the 1st paragraph



The diagram shows a red rectangular box representing a block element with the label `display: block`. Inside this box, there is a green rectangular box representing an inline element with the label `display: inline; width: 10em`. The text "Let's add some content to see how the block behaves." is written twice, once before and once after the green box, demonstrating how the inline element is slotted into the flow of the block.

`display: list-item` means a block box is created for the content and the list item marker, for example, `` or ``.

Figure 7 - `display: list-item` being used



The diagram shows a red rectangular box representing a block element with the label `display: block`. Inside this box, there are two blue rectangular boxes, each representing a list item with the label `display: list-item`. Each blue box is preceded by a black dot, representing a list marker, illustrating how `display: list-item` creates a block box for the content and adds a list item marker.

There are other values in the `display` property. Have a look for examples of:

- > `display: run-in`
- > `display: table`
- > `display: table-row`
- > `display: table-cell`

Positioning properties

Let's now talk about how to position elements on the webpage. Before you position items to the left, right, top or bottom, the properties won't work unless you set the position property correctly. There are four positioning properties to look at initially:

- 1 static
- 2 relative
- 3 absolute
- 4 fixed

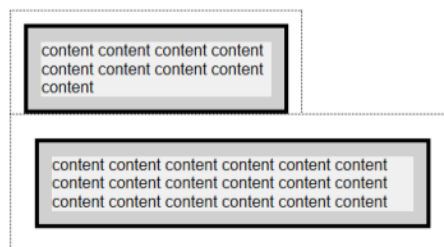
Static positioning

If you don't specify the position for an element, it will default to static positioning and the positioning parameters top, right, bottom, left do not apply. The HTML elements will be positioned according to the normal flow of the page.

In this scheme, block boxes flow vertically starting at the top of their containing block with each placed directly below the preceding one. Inline boxes flow horizontally from left to right.

You should note that vertical margins are collapsed in the static flow. That is, instead of adding the bottom margin of a box to the top margin of the one below it, only the larger of the two values is used, as illustrated in Figure 8.

Figure 8 - Static flow showing bottom & top margin usage



Relative Positioning

Relative positioning tells the element where to sit relative to where it would be sitting if it was just left as the default, static positioning, i.e. relative to its normal position.

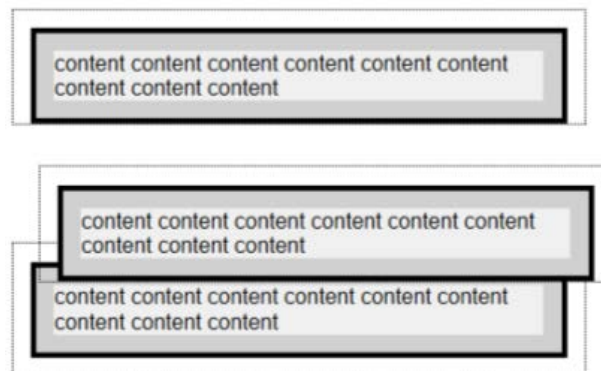
Let's look at an example:

```
h1 {  
    position: relative ;  
    left: 10px ;  
}
```

In the above example, the h1 heading does not move 10px left from any particular item. It moves 10px left of where it would have been positioned if the positioning was left as default (i.e. static).

When an element specifies `position: relative;` it is initially positioned following the static flow rules. Surrounding boxes are positioned accordingly. Then, the box is moved according to its offset properties as shown in Figure 9.

Figure 9 - Relative positioning of content boxes



Note that the surrounding boxes above are positioned normally (i.e. static), including the collapsing of vertical margins, and that the repositioned box may overlap other boxes.

The offset values are specified using a combination of the `top`, `right`, `left` and `bottom` style properties. The value of each is interpreted as the distance the box's corresponding outer edge should be moved with respect to its original position in the normal flow.

Note that opposing offsets are constrained. For example, if you specify both `left` and `right` and the value of one is not the exact negative of the other, the `right` setting will be ignored. A specific `width` setting may also cause an offset to be ignored. The same is true of the `top`, `bottom` and `height` properties.

In practice, you'll probably want to specify only one of `left` and `right` and one of `top` and `bottom`.

Absolute Positioning

When the positioning of an element is set to absolute, it takes the element out of the normal flow of the document and positions it in relation to the first parent element that doesn't have `position: static;`. The web page and other elements behave as though the `position: absolute;` element is not there.

There is not element other than static elements, the `position: absolute;` element is contained within the `<html>` block. Absolute positioned elements are always treated as block-level elements. As such, they establish a new containing block for any descendants, i.e. any elements contained within the absolutely positioned element.

The position of an absolute positioned element is determined by its offset values: `top`, `right`, `bottom` and `left`. These values work in much the same way as with relatively positioned elements. But unlike relative positioning, where the offsets are measured from the element's position in the normal flow, an absolutely positioned element is offset from its container block.

Elements set as `position: absolute;` can overlap other elements.

Fixed Positioning

Fixed positioning is a special case of absolute positioning. A fixed positioned element is anchored to the browser window and will not move, even if the screen is scrolled. This is handy for toolbars or banners. Just like in the absolute position, the element is removed from the normal flow, the other elements act as though the `position: fixed;` element does not exist and the fixed position element can overlap other elements.

It is important to note that some browsers have difficulty displaying fixed positioned elements correctly.

Container Block for Absolutely Positioned Elements

The containing block of an absolutely positioned element is defined a little differently than it is for other elements. The containing block for an absolutely positioned element is established by its closest, positioned ancestor. That is, the nearest element outside it that has a **position** of **absolute**, **relative** or **fixed**. If there is no such ancestor element, the initial containing block (the browser window) is used.

Recall that non-absolutely positioned elements used the containing block of their closest, block-level ancestor. But for absolute elements the containing element can be an inline element.

Additionally, if that containing element is a block-level element, the padding edge of that element forms the container block, not the content edge. In other words, the offsets are measured from just inside the border of the containing element.

If the containing element is an inline-level element, it gets a little more complicated. Since an inline element may generate several line boxes, the container box is defined as the area bordered by the top and left content edges of the first box within that element and the bottom and right content edges of the last box it contains.

For example, consider a relatively positioned **span** that contains text taking up two lines, some of which is bolded.

Figure 10 – two line span with bold text



Due to the inconsistencies noted previously in how browsers handle relatively positioned elements, nesting absolute elements inside relative elements may also cause unexpected results and be worth avoiding.

While margins are honoured with absolutely positioned elements (position offsets are measured from the margin edge of the element's box) they are otherwise meaningless since absolutely positioned elements do not participate in the normal flow.

An absolutely positioned element establishes a containing block for any elements within it. Descendent elements follow the same positioning rules they normally would, just offset by the position of the containing element.

They can even contain other absolutely positioned elements. These descendants are likewise removed from the normal flow within the containing block, so they can appear outside of the bounds of the containing element.

Descendant Positioning

Relatively positioned elements may or may not establish a new containing block for positioned (relative or absolute) descendant elements. They follow the same rules as non-positioned elements.

If the relatively positioned element is a block element, it establishes a new containing block. Positioned elements within it will use the "offset" position of that element as a base for positioning. In other words, the offsets of descendant elements are compounded.

If the relatively positioned element is an inline element, its offsets are not combined with the offsets of its positioned descendants. Instead they are based on the same containing block used by the relatively positioned element.

The original standards specification declared that relatively positioned elements always created a new containing block. However, later corrections to the standard changed this to state that such boxes follow the rules of non-positioned elements.

Some browsers incorporate this correction but others do not. Due to these disparities, you may wish to avoid such situations or always use block elements for relative positioning.

Stacking Order

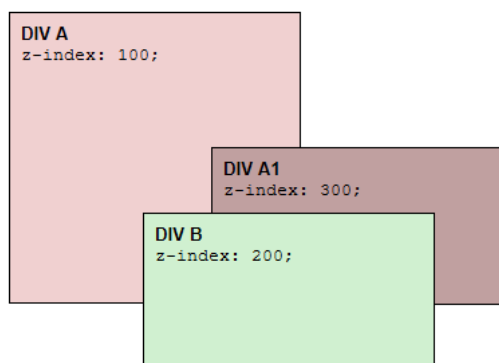
As explained earlier, absolutely positioned elements may overlap non-positioned elements and each other. Two things determine when one appears in front of or behind another, the **stacking context** and the **z-index** property.

Each absolutely positioned element belongs to a stacking context. The initial containing block generates an initial stacking context. Absolutely positioned elements within the same stacking context are displayed according to their **z-index** value.

An element with a higher **z-index** appears in front of an element with a lower **z-index**. When two elements have the same value (or if neither has been assigned a value) the source order is used.

But the staking context is also important. The element that is defined earlier in the source is displayed behind the one defined later. Let's look at Figure 11.

Figure 11 – Stacking DIVs



In the example above, note the stacking order of the absolutely positioned divs and the z-index values. DIV A1 is contained within DIV A, while DIV B is independent. For this reason, DIV B appears on top of DIV A1, even though A1 has a higher z-index value. That's because it's containing element, DIV A, has a lower z-index value than DIV B. Their z-index value applies only to elements in the same local stacking context. Otherwise the z-index value of the containing element is used.

A local stacking context is created when an absolutely positioned element is given a z-index value other than auto. Any absolutely positioned elements nested within the element participate in that local stacking context.

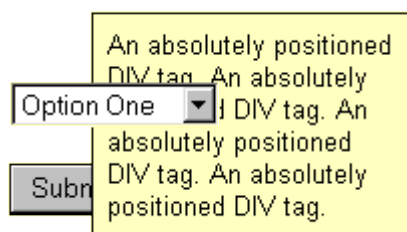
Negative values can be specified for z-index. An element with a negative value is displayed behind any with an undefined or positive z-index value in the same stacking context.

Some browsers do not properly handle negative z-index values.

Overlapping Form Controls and Plug-ins

Some elements cause display problems when overlapped by positioned elements as shown in Figure 12. This can occur when elements overlap form controls, applets or plug-in displays like Flash.

Figure 12 – Overlapping form controls



Overlapping display problems occur because browsers may let other programs handle the display of these elements, like a plug-in or the operating system. Even setting the **z-index** will not help.

These other programs control that space and will simply draw on top of whatever the browser renders.

Which elements “bleed through” depend not only on the browser software you are using but also on the version of browser. Your only recourse is to try to avoid overlapping such elements. In some cases, you might know or be able to dynamically detect an overlap and hide the offending element from view by setting its style via scripting using settings like **visibility:hidden** or **display:none**.

This information has only dealt with positioning. Styling, form controls and dynamic content changes via scripting can complicate matters even more.

Even when browsers follow the standards, there is plenty of room for error. All software has bugs and specifications can't cover every possible situation. Test, test and retest a small version in multiple browsers before going into full scale production.

Floating

Floating is one way of telling the element son the page where you would like them to sit. Floating is achieved by setting the `float` property on an element's style to either `left` or `right` or `inherit`. Let's look at a simple example:

```
img {  
    float: right ;  
}
```

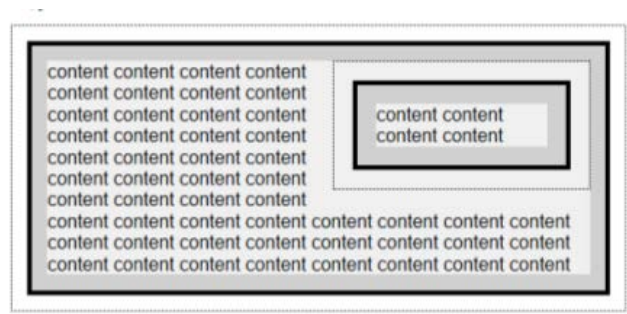
Special rules apply to floated elements. When specified, the box is positioned vertically as it would be within the normal flow, its top aligned with the top of the current line box. But horizontally, it is shifted as far to the right or left of its containing block as possible, within that block's padding (just like other content). Surrounding inline content is then allowed to flow around the opposite side.

Let's look at another example. Consider the following code:

```
<p>  
    <span style="float:right; width:40%;">content...  
    </span>  
    content content content content content content content  
    content...  
</p>
```

Figure 13 shows the result of the code above, where a floated element is defined as part of inline text. Margin, border and padding styles have been omitted for demonstration purposes.

Figure 13 - Using float



There are a few things to note regarding floated boxes. For one, the box being floated should have a width defined for it, either explicitly or implicitly. Otherwise, it will fill its containing block horizontally, just like non-floated content, leaving no room for other content to flow around it. As such, floated boxes are always treated as block boxes, even if they are defined using inline elements.

Second, unlike boxes in the normal flow, the vertical margins of a floated box are not collapsed with the margins of boxes either above or below it.

Finally, a floated box can overlap block-level boxes adjacent to it in the normal flow.

Overlapping floats

Here's an example where the floated element has much more text than its containing block (see Figure 14). Another paragraph has been added to the previous code to demonstrate.

```
<p>

    <span style="float:right; width:40%;"> content
    content content content...

</span>

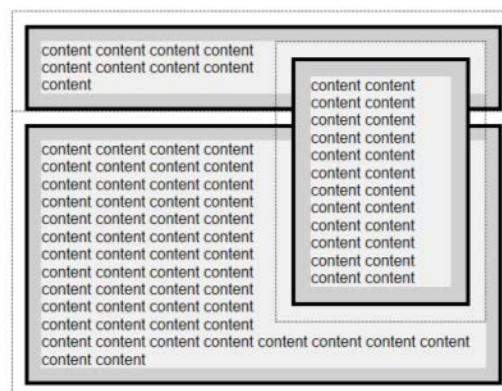
content content content content...
</p>

<p>

content content content content content content content
content...

</p>
```

Figure 14 - Overlapping floated boxes



Note that the floated box overlaps the borders of both its parent box and one following it (although the text in the second block flows around the float). This can be avoided using the `clear` property.

Setting `clear:right;` on the second box moves it down to just below the floated box. Here's an example where the `clear:right;` style has been set.

```
<p>

    <span style="float:right; width:40%;"> content
    content content content content content content
    content content content...

</span>

content content content content...

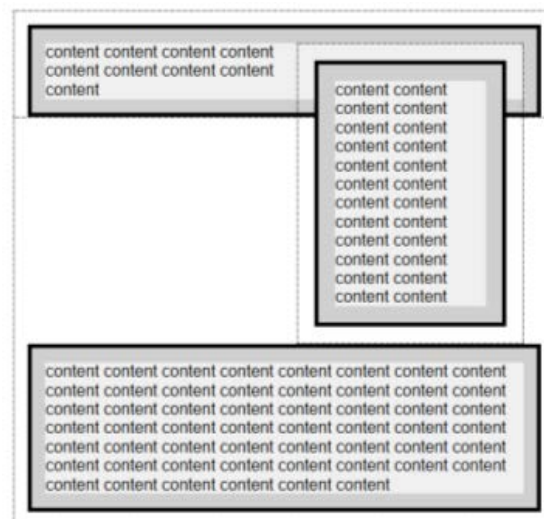
</p>

<p style="clear:right;">

content content content content content content content
content...

</p>
```

Figure 15 - Using the clear setting



This causes the top margin of the second box to be increased just enough to place the second paragraph's border and content just below the floated box.

The float still overlaps the top box (the paragraph that contains it) but that's because the example was purposely set up to do that, using a large amount of content in the floated element compared to the surrounding content. Normally this would be avoided by placing the floated outside of the paragraph and letting both paragraphs flow around it.

```
<span style="float:right; width:30%;"> content content  
content content content content content content content...  
</span>  
<p>content content content content...</p>  
<p>content content content content...</p>
```

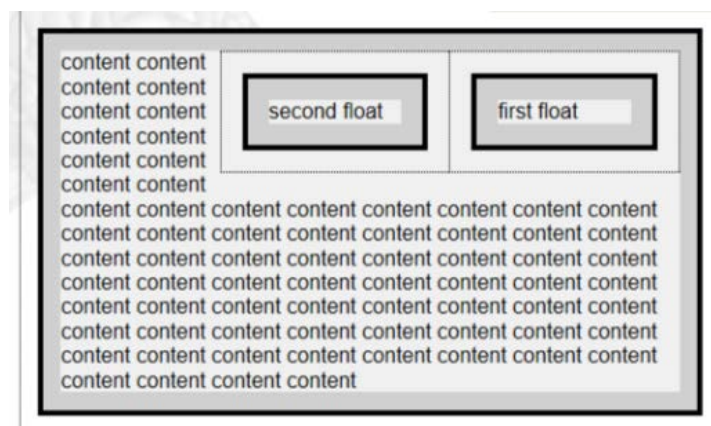
The `clear` property can be set to one of `left`, `right`, `both`, `none` or `inherit`. It only has meaning for block-level elements.

Adjacent Floats

When two or more adjacent elements are floated, their tops are positioned on the same line (side by side) if there is sufficient horizontal space to accommodate them. If not, the latter element(s) are moved down to a position where there is sufficient space, always aligned with a line box.

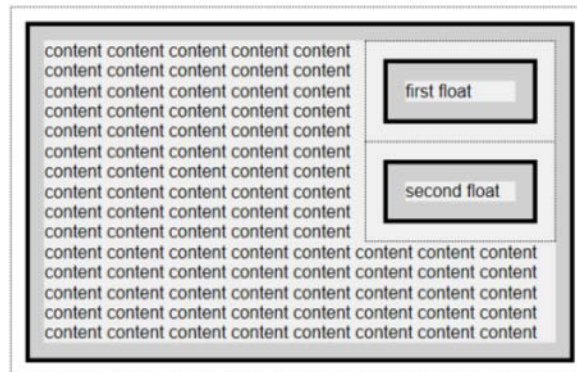
It should be noted that this downward shift may also occur for a single floated element if it's too wide to fit within its initial position with adjacent content. Figure 11 shows the effect of placing two right-floated elements next to each other. Note how the first floated element appears farthest to the right.

Figure 16 - Adjacent floats



The `clear` property can be used on a floated element to force it below adjacent floats (recall that floated elements are treated as block-level elements for positioning). Just remember that vertical margins are not collapsed for floated boxes. Figure 17 shows the same code but with `clear:right` set for the second floated element.

Figure 17 - Adjacent floats with clear setting used



CSS selector relationships

The child element

A CSS child selector applies to the elements that are children of another element. A child element is an element that is the immediate or direct descendant of another element. For example all the `` elements in an unordered list are children of the ``. But any **anchors** on the `` elements are not children of the ``.

```
<ul>
    <li><a href="#">...</a></li>
</ul>
```

The descendant element

A CSS descendant selector applies to the elements that are inside another element. For example an unordered list has a `` tag with `` tags as descendants. In the following HTML:

```
<ul>
    <li><a href="#">this is a link</a></li>
</ul>
```

The `` tags are descendants of the `` tag. The `<a>` tag is a descendant of both the `` (child descendant) and `` (grandchild descendant) tags.

Look at the example below.

```
<div id="1">
  <p id="2">
    <p id="3">Woot</p>
    <p id="4">Meh</p>
  </p>
  <div id="5">
    <p id="6">Bazinga</p>
  </div>
</div>
```

A CSS rule written as `div p` would target ids 2, 3, 4 and 6 because all the `<p>` tags are contained within `<div>` at some level - they are all descendants of `div id="1"`.

A CSS rule written as `div > p` would target only 2 and 6 because `<p id="2">` is a child of `<div id="1">` and `<p id="6">` is a child of `<div id="5">`. The other two `<p>` elements have `<p id="2">` as their parent.

Children are also descendants, but are a special case being the first level of descendants.

A CSS rule written as `div:first-child` would target `<p id="2">` only.

A CSS rule written as `div:first-child p` would target `<p id="3">` and `<p id="4">` and any other `<p>` descendants in there if there were any.



ADDITIONAL RESOURCES

RECOMMENDED 2

For more information and tutorials on CSS visit:

- > <http://www.cssbasics.com>
- > <http://www.w3schools.com/css/default.asp>
- > http://css-discuss.incutio.com/wiki/Main_Page

Creating CSS in a document lesson

In this part of the learner guide we are going to work through a project building a website using CSS and suggest you follow along and build it at the same time. This will allow you to practice your CSS skills and help you to prepare for your assessment.

Resources required

You will need web building software like Adobe Dreamweaver, Notepad++, Text Wrangler (for Mac users) or you can simply use the Notepad tool on your computer. This lesson uses Dreamweaver as an example.

You will also need the following files provided in your TAFEnow online course resources:

- > [ICTWEB409_site_css.pdf](#)
- > [ICTWEB409_lesson_images.zip](#)

Finding where you went wrong and how to fix it is an essential skill with CSS as you could spent a lot of time trying to fix the wrong style. If you get stuck at any time use the [ICTWEB409_site_css.pdf](#) as a reference guide to the CSS that is being built as we go along. Try not to just copy and paste it because then you won't develop the skills you need to be a web developer. Use the document to check against your CSS document to see where you might have gone wrong. Understanding CSS logic is 90% of the challenge.

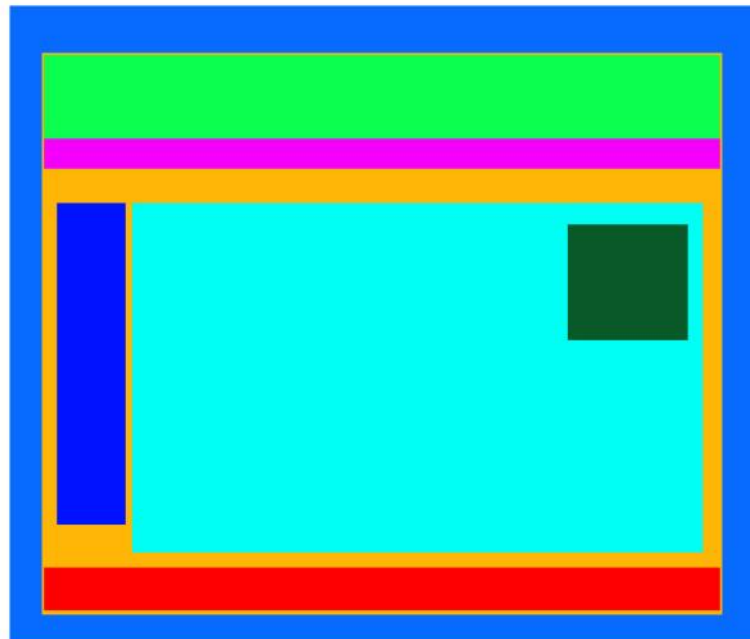
Step 1 – Set up a local root folder

- 1 Create a folder on your computer's hard drive called **NatureSnapWeb** or something similar.
- 2 Within this folder, create two subfolders called **images** and **resources**.
- 3 Download the two files listed above ([ICTWEB409_site_css.pdf](#) and [ICTWEB409_lesson_images.zip](#)) into the **resources** folder.
- 4 Unzip the **ICTWEB409_lesson_images.zip** file and move all the unzipped files into the **images** folder.

Step 3 – Setting up the DIVs

You will make a web page using the following design shown in Figure 18.

Figure 18 - The CSS design



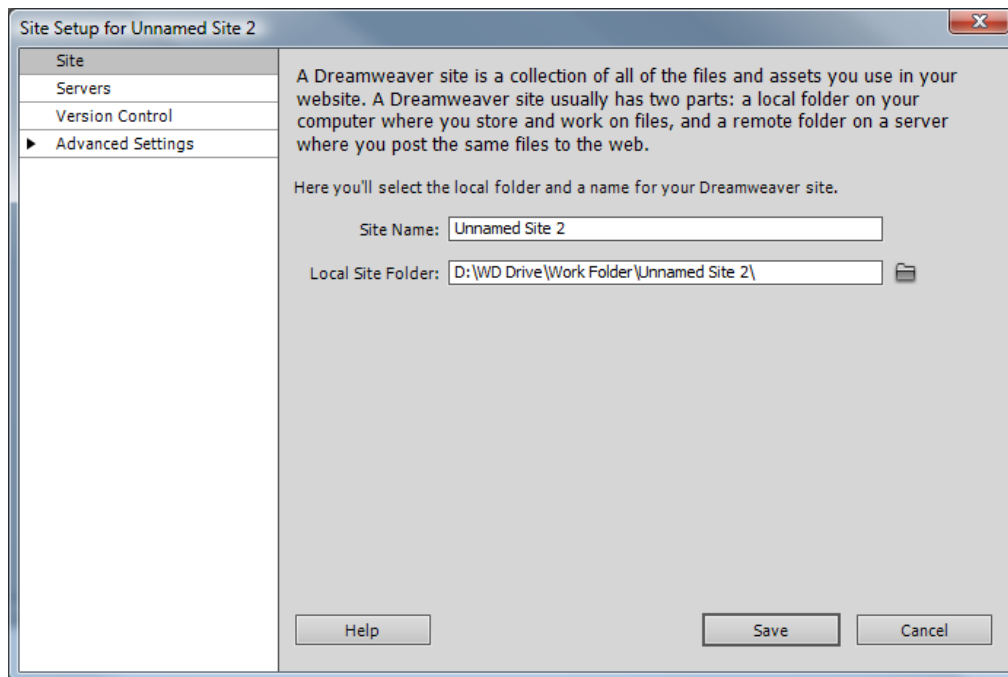
The design is layout as follows:

- > The outside medium blue is the left over available browser space
- > The light green is the banner space
- > The pink is the page menu
- > The orange is the “wrapper” for the whole page content
- > The dark blue is the secondary menu
- > The light blue is the page content area
- > The dark green is a floated image
- > The red at the bottom is the footer

- 1 Open Dreamweaver.
- 2 From the menu bar, select Site > New Site.

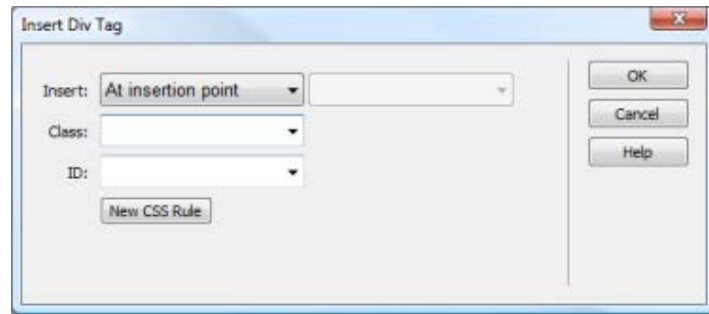
The Site Definition dialog box opens (Figure 19). Make sure the Site line is selected.

Figure 19 - Site Definition dialog box – SITE section



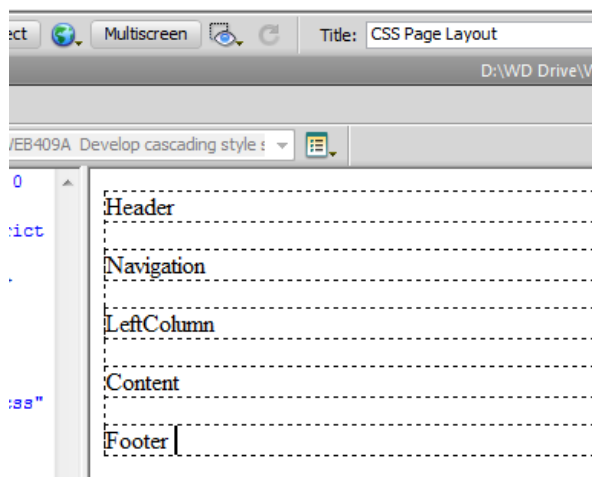
- 3 Delete the temporary name, Unnamed Site 1 and rename it to the name of the site you are creating (in this case use **Nature Snap Photography**).
- 4 In the Local Site Folder change from the current location to the location of folder you created in Step 1. You can do this by clicking on the folder icon and navigating **inside** the selected folder and clicking on select.
- 5 Click Save and the window closes.
- 6 Create a new blank HTML page called **index.html**.
- 7 With your **index.html** page opened, click in the design area and then select the layout tab on the quick menu bar and select **insert DIV tag**. The Insert DIV tag dialog box appears as shown in Figure 20.

Figure 20 - The Insert DIV tag dialog box



- 8 In the ID box type in **wrapper**, our overall box. Press OK.
- 9 Click inside the DIV you just made and click the insert DIV again.
- 10 This time click on the insert drop down and select **after start of tag** and in the drop down next to it select **<DIV id="wrapper">**. In the ID name it **header**.
- 11 Click Insert DIV again and this time select **after tag** "header" and give it the ID of **navigation**.
- 12 Repeat Step 11 for "left column", "content" and "footer" DIVs as well.
The design area should now look like Figure 21.

Figure 21 - Design view with no CSS attached

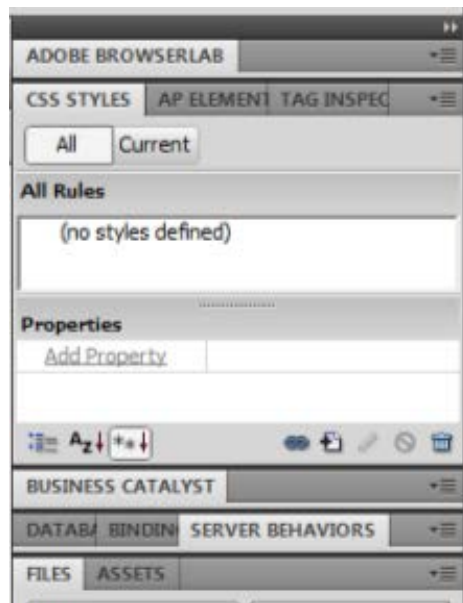


Step 4 – Applying some CSS

Now that the DIVs are in the wrapper they can be positioned and resized.

To create a style you have to use the CSS Styles panel on the top right of the screen or by going **Window > CSS Styles** or by pressing **Shift+F11**. How it looks is shown below in Figure 22.

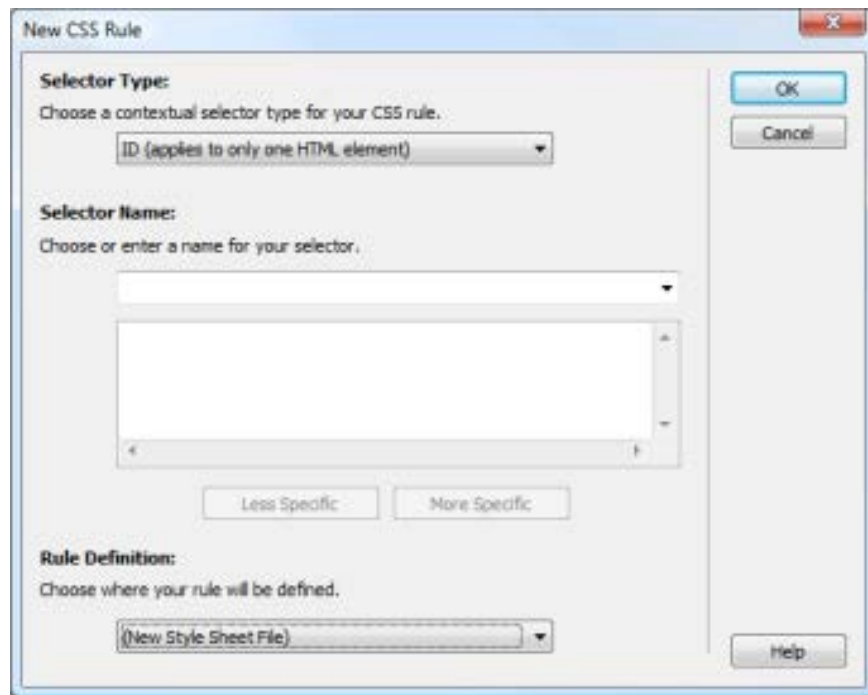
Figure 22 - The CSS Panel



You are going to reset all the padding and margins to the WHOLE document so that any changes that you make are your own and not something built into the browser.

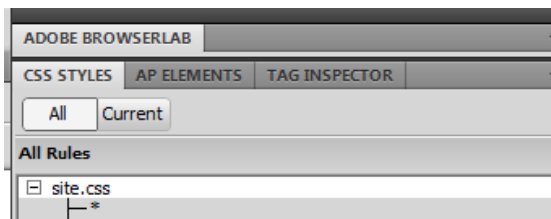
- 1 Click on the **New CSS Rule** button.
- 2 The New CSS Rule dialog opens up (Figure 23). Now you are going to define a wildcard to go at the top of the document to set the margins and padding.
- 3 In the contextual selector select **ID**.
- 4 In the selector name enter an *****.
- 5 In the Rule definition select **New Style Sheet File**. This creates a new linked external stylesheet.

Figure 23 - The CSS dialog box



- 6 Click OK.
- 7 When asked to save the document, create a folder called **CSS** and save it as **site.css**.
- 8 The CSS rule definition box opens up. There are many sections in this dialog box that you will use – some more than others – but you can make almost all of the CSS rules that are on the CSS cheat sheet.
- 9 For this rule select **Box**. The Box section controls the height and width of DIVs & other page elements, whether or not they are floated to the left and right of a page, and the padding and margins etc.
- 10 Ensure that **Same for all** is selected on both the padding and margin, put a **0** in the box and make sure the unit used is **px**. It is important to do this part for all sites that you work on. It clears all padding and margins on the page before you start.
- 11 Click on **OK**.
- 12 Your CSS panel should now look like Figure 24.

Figure 24 - CSS panel showing the attached stylesheet and style



This shows the newly created stylesheet linked to your document and the style that you just created.

Step 5 – Wrappers, margins and putting things in their place

Once you have reset the margins and paddings, the next step is to float the wrapper out into the middle of the screen.

When researching CSS and margins you would have come across code similar to this:

```
margin: 5px 20px;
```

which means the top and bottom margins are 5px and right and left margins are 20px.

There is another value that can be used besides **px** to set a distance and that is **auto**. Auto calculates the available window space minus any widths or settings and gives it the remaining value.

- 1 Create an ID called **wrapper**, (make sure that wrapper matches the same spelling, including case, of the DIV made earlier) and set:
 - > the top and bottom margin to **0** and
 - > the left and right one to **auto** and
 - > set the width to **960px**.

If you click apply you will see the page rearrange and be centred on the page.

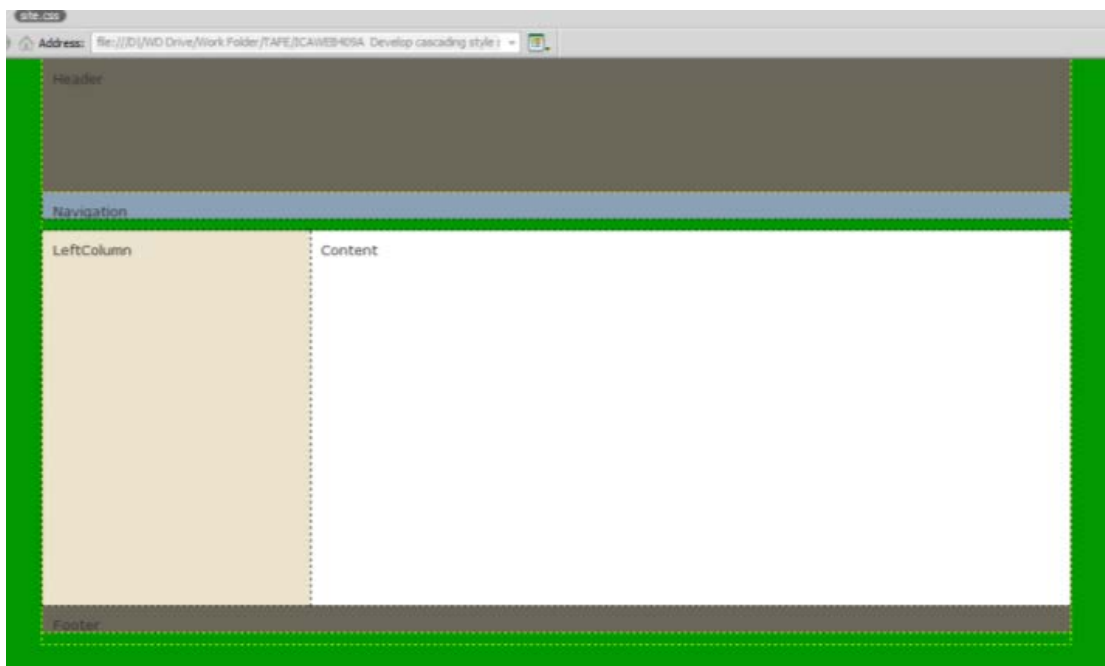
- 2 Click **OK**.
- 3 Create the other DIV settings using the following information in Table 4.

Table 4 – DIV settings

DIV	Set to
Header	124px high and as wide as the wrapper
Navigation	Same overall width, 25px high floated to the left
Left Column	350px high, 250px wide floated to the left
Content	710px wide, 350px high, floated to the left
Footer	Same width as navigation, 25px high, clear both

- 4 You might find it helpful to add background colours to see how things sit before adding any content or backgrounds.
- 5 Your design should look like Figure 25.

Figure 25 - The DIVs with settings attached



Your completed stylesheet should look something like this.

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

body {
    font-size: 13px;
    color: #000;
    background-color: #fff;
}

#wrapper {
    margin: 0 auto;
    width: 960px;
}

#content {
    float: left;
    background: #fff;
    height: 350px;
    width: 760px;
}

#header {
    width: 960px;
    float: left;
    height: 124px;
    background: #fff;
}

#footer {
    width: 960px;
    height: 25px;
    clear: both;
    background: #fff;
}

#navigation {
    float: left;
    width: 960px;
    height: 25px;
    background: #fff;
}

#leftcolumn {
    background: #FFF;
    height: 350px;
    width: 200px;
    float: left;
```


}

Step 6 - Horizontal navigation bar using CSS

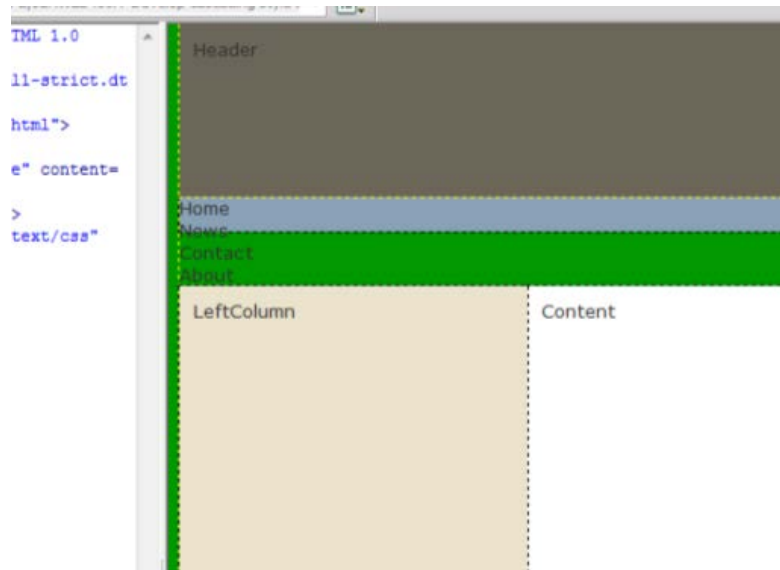
Once all the DIVs are in place and checked the next thing to do is start working on some of the more technical items to see if they make the page “break”. One item that could interfere with your page design is the navigation.

Most CSS menus are actually redefined lists that have been manipulated using CSS to look like a menu bar.

- 1 Click in the navigation DIV and make an **unordered list** with the values
 - Home
 - Gallery
 - Shopping
 - Workshops
 - Events
 - One Day Trips
 - In The Press
 - About Nature Snap
 - Contact Us
- 2 Give the link on each one the value of **JavaScript:void(0);** to create a link with no value. You could use # instead of JavaScript:void(0); but it has been found more and more that clicking on a # link will cause varying things to happen from not doing anything, to reloading the page, to going back a varying number of pages in the browser history.

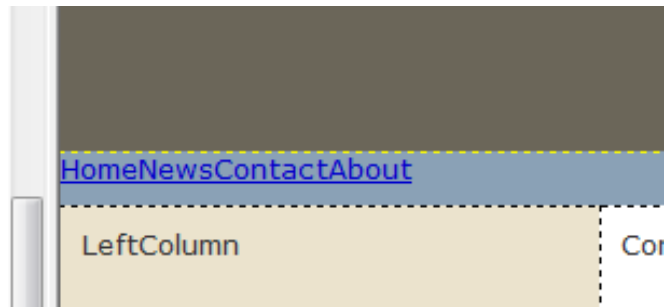
Your page should look like Figure 26.

Figure 26 - The unordered list without CSS



- 3 Now to create the CSS. Create a new compound rule for the UL in the navigation DIV. The compound tag should be named **#navigation ul** this means it refers **ONLY** to the UL in the ID called navigation. Note: You don't want to redefine the UL tag as you will need to use the UL later on, so you are just overwriting the UL in the navigation ID.
- 4 Turn off the bullets in the list so that they don't appear. Do this in **list type**.
- 5 Turn off any padding and margins by making them default values to **0**.
- 6 Set the width for the navigation to **960px**.
- 7 Press **OK**.
- 8 Make a new rule, this time for the LI in the navigation **#navigation li**
- 9 Float it to the left. If you click **apply** the menu will now run from left to right, but will look squashed (See Figure 27).

Figure 27 - the menu running left to right



- 10 Things seem to look a little more like a menu but it needs to be a bit more spaced out. Make another rule but this time it will be applied to anything that is a link in the navigation **#navigation a**. What is going to change is any/each link "block" is going to have their own set width, a set colour and it will be aligned in the centre.
- 11 Set the line height, the height of each of the navigation blocks, to **25px**.
- 12 Set the text decoration to **none**. This will take the underline away from the linked text.
- 13 Give it a text colour of **#000**.
- 14 Give it a background colour to match other pages blocks **#FFF**.
- 15 Set the display to **block** as you want to show the whole block.
- 16 Set the text alignment to **center**.
- 17 Set the vertical alignment of the text to **middle**. This will put the text in the middle of the set area.
- 18 Set the width to **auto** meaning each block will be as wide as the text in the container.
- 19 In padding, set the left and right padding to **10px**. This will give padding between each menu item.
- 20 Click OK.
- 21 To have it function more like a menu system, let's edit what happens when you rollover the text. Create a new rule. The name will be **#navigation a: hover**. This is part of the pseudo classes in CSS and defines what happens when you rollover a link in the navigation DIV.

- 22 Set the text decoration to **none**.
- 23 Change the text colour to a different colour, i.e. **#999**.
- 24 Click apply or OK to see the menu change as in Figure 28.

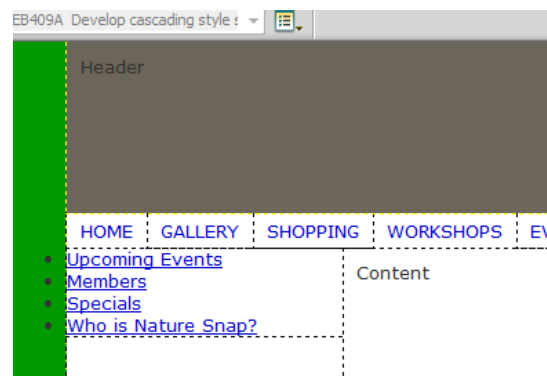
Figure 28 - The menu with CSS applied



Step 7 - Vertical navigation bar using CSS & images

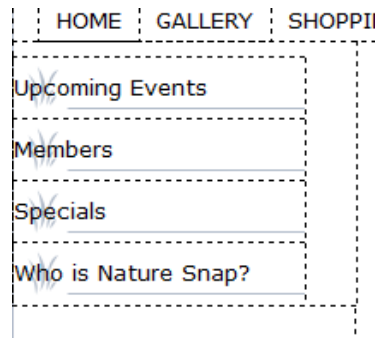
- 1 Create a new DIV inside the leftcolumn DIV called **navcontainer**.
- 2 We are now going to insert a vertical menu in the left column that will have four buttons/links. Click in the Left Column DIV and insert an unordered list called **navlist** which has four lines called
 - Upcoming Events
 - Members
 - Specials
 - Who is Nature Snap?
- 3 As in Step 6 with the horizontal menu, change each line to a link using the **JavaScript:void(0);** method. When complete the page should look like Figure 29.

Figure 29 - The un-styled menu



- 4 Inspect the image named **pg_button_bg.png**. You will see it is **193px** wide, which is important for this design to work.
- 5 Create a rule for the **#navcontainer** which is **193px** wide.
- 6 Create a rule for the UL in **#navcontainer** which sets the left margin and padding to **0** and the list type to **none**.
- 7 Make sure it stays on the left by floating it to the **left**.
- 8 The next step is to create the image button look for the menu. Because the link is the thing that is repeatable and what we want to make look like an image button this is what we want to work with. Make a new rule for the **#navcontainer a** rule.
- 9 Set display as **block**.
- 10 Set the width to **200px**.
- 11 Set the line height to **34px**. This is the height of the image, less a margin.
- 12 Set the background image to the image **pg_button_bg.png**. Select the image from the folder icon and navigate to the images folder and select image.
- 13 Set the background repeat to **no-repeat** as you only want the image to appear once.
- 14 Set the text alignment to **left**.
- 15 To separate the images just a little bit set the bottom padding to **2px** and the other paddings to **0**.
- 16 Press OK. The screen should look like Figure 30.

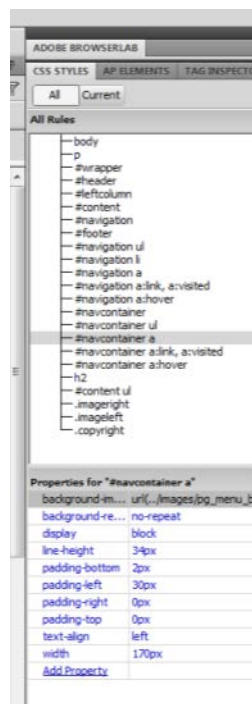
Figure 30 - The left menu with a problem



You should see a problem with the left navigation. The text floats over the top of the images for the background.

- 17 The padding has to be pushed **30px** to the right. To do this make sure you have the CSS panel open or press **shift +F11**.
- 18 Find the **#navcontainer a** and make sure the properties are showing and click on it (Figure 31).

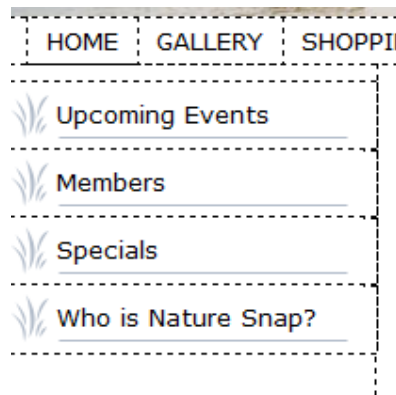
Figure 31 - The CSS panel with properties panel open



- 19 You can change the attributes by clicking in the box marked **0px** next to padding-left and changing it to **30** or click the pencil in the bottom right bar and go to the padding and change the left to **30** and click **OK**.

20 Your menu should now look like Figure 32.

Figure 32 - The styled vertical menu



21 Now make a new rule for the hover state that makes the text appear with an underline when you hover over it. Make a rule for the **a:link** and **a:visited** for both the **#navcontainer** and **#navigation**. It will make sure the text stays the same colour and appears with no text decoration.

Important!

- > **a:hover** MUST come after **a:link** and **a:visited** in the CSS definition in order to be effective!
- > **a:active** MUST come after **a:hover** in the CSS definition in order to be effective!
- > Pseudo-class names are not case-sensitive.

Example:

```
a:link {color:#FF0000;}           /* unvisited link */
a:visited {color:#00FF00;}        /* visited link */
a:hover {color:#FF00FF;}          /* mouse over link */
a:active {color:#0000FF;}         /* selected link */
```

Step 8 – Headings, text and lists

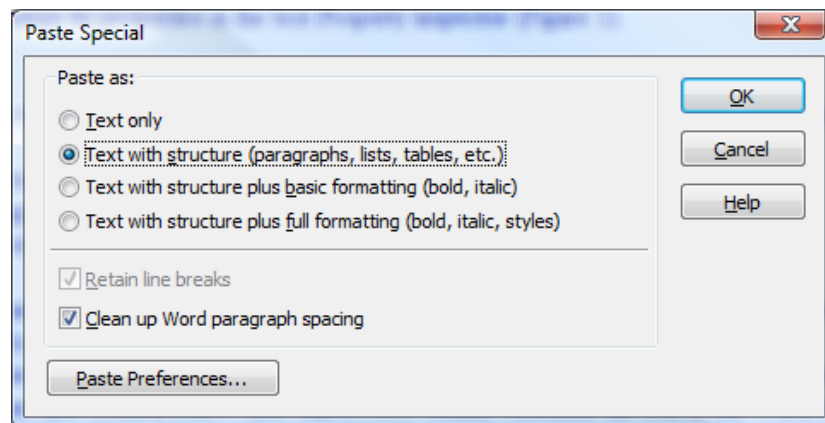
Pasting text into a web editor from Word

The text (content) for the website may be provided in various formats, such as Word Document files, PDF's, Open Office Documents, etc. Copying and pasting between these documents into web editors such as Adobe Dreamweaver can cause issues as the pasted content often brings formatting with it.

Copying and pasting the text brings the text into Dreamweaver just as it appears in Word, but behind the scenes what happens is that instead of pasting the text into the document, it has also created a new style called MsoNormal, reformats the text to the Word style and breaks the design that you have spent time setting up!

To avoid this problem, when you need to paste between different programs, go to Edit > Paste Special or Shift+Ctrl+V and the Paste special dialog box appears (Figure 33).

Figure 33 - The Paste Special Box



Most of the time you can use the second option which will allow to text to be put in and include paragraphs, tables and basic text structure. Now you can format the text the way you designed it, not the way another program has decided you should use it.

- 1 Paste the following text into the content area of the page.

One Day Trips

Our one day trips are one of the best ways to learn more about photography in a specialised setting.

Whether you are a new photographer or an experienced shooter wanting to hone your techniques or rediscover your inspiration, our Photographic Workshops provide challenging, fun-filled adventures, in the most spectacular locations.

Even with so much to learn, it's the sort of course that will seem more like a holiday than school! You'll enjoy luxurious accommodation and wonderful food while you drink from the font of knowledge. Choose an exclusive getaway totally in the field, or join a larger group and benefit from the combined experience and talent of fellow photographers.

Photography Explained - One Day Workshops

Our new Photography Explained workshop will cover everything from taking the photo to post production and equip you with skills and techniques to take better photos right away.

In this one-day workshop you will discover that you don't need loads of expensive equipment to take beautiful photos. Putting the WOW factor into your own images is quite simple when you understand more about:

- Composition and perspective
- The use of light
- The best equipment for your specific needs
- Which camera settings to use
- How to make use of the latest technologies
- Printing and displaying your photos
- Preserving your precious memories for the future
- Simple, effective work flows to organise your images

Whether you are a keen beginner or a more advanced photographer, after this informative day you will never look through the viewfinder the same way again.

Join us in this classroom-style seminar as he shares his knowledge, passion and experience to unravel the mysteries of photography with simple tips and techniques to help you create better photographs.

It's not about the size or the price of your camera gear, it's all about capturing the magic of the moment.

Numbers are strictly limited, so don't delay. Book your ticket now to join in what is guaranteed to be a great day spent with like-minded people, as you discover how to instantly improve your images.

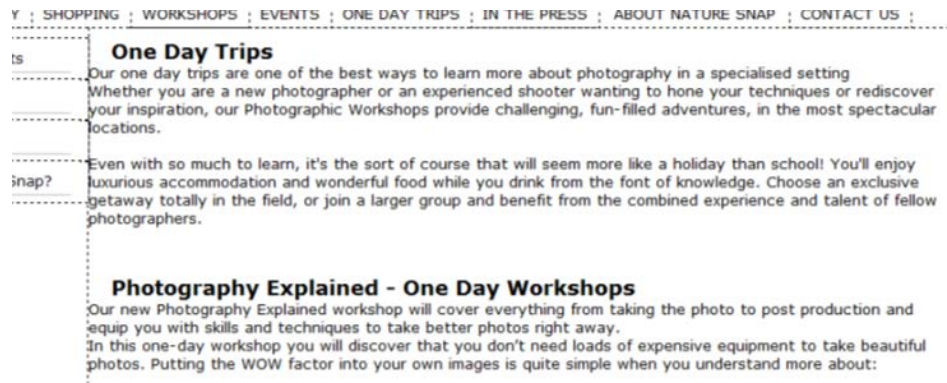
Coastal Exposure Workshop

Spaces are strictly limited in this hands-on workshop, which will take place on January 26 in one of our favourite spots in Australia - the Mid-North Coast of NSW, just a few hours north of Sydney.

You can take advantage of the wealth of our teams combined 50 years of photographic experience and learn how to capture better images. The beautiful Mid-North Coast affords an array of spectacular scenery, from beaches to lush green valleys, to rocky waterfalls.

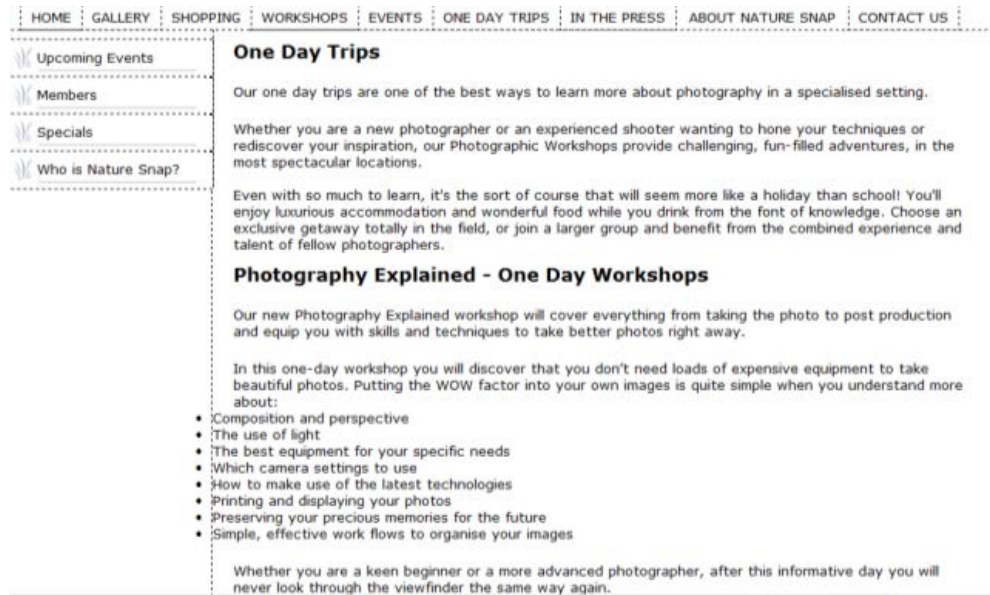
- 2 Format the text for One Day Trips, Photography Explained - One Day Workshops, Coastal Exposure Workshop to **Heading 2** or **H2** format.
- 3 Using CSS redefine the H2 tag to Verdana, Geneva, sans-serif. You will also need to bring it down from the top margin and because we reset all the margins & padding at the top you will need to bring the left margin out from the left navigation panel as it will be right up against it. Use your own discretion, the design uses **20px** on left and **10px** on the top.
- 4 Click **OK**. The page should look like Figure 34.

Figure 34 - the text with Heading formatted



- 5 Next you have to redesign the `<p>` tag to get it away from the edge and formatted into a more readable format. Open the CSS rule to redefine the `<p>` tag. The paragraph block needs to be shifted left and have some room made around it to stop the text blocks running together.
- 6 The padding needs to be put on the left to bring it in from the left navigation, the same as the Heading 2, away from the right edge so it doesn't run right into it. The design being used has set the left and right margins to **20px**.
- 7 To stop the paragraphs from running together you can do a few different things depending upon your design choice:
 - > Set the top and bottom margins to put the spacing at the start and finish of the paragraph
 - > Set the top margin to put the spacing at the start of the paragraph
 - > Set the bottom margin to put the spacing at the end of the paragraph
- 8 Click **apply** to see the changes and when you feel happy with it, press **OK**. Your design should look a little like Figure 35.

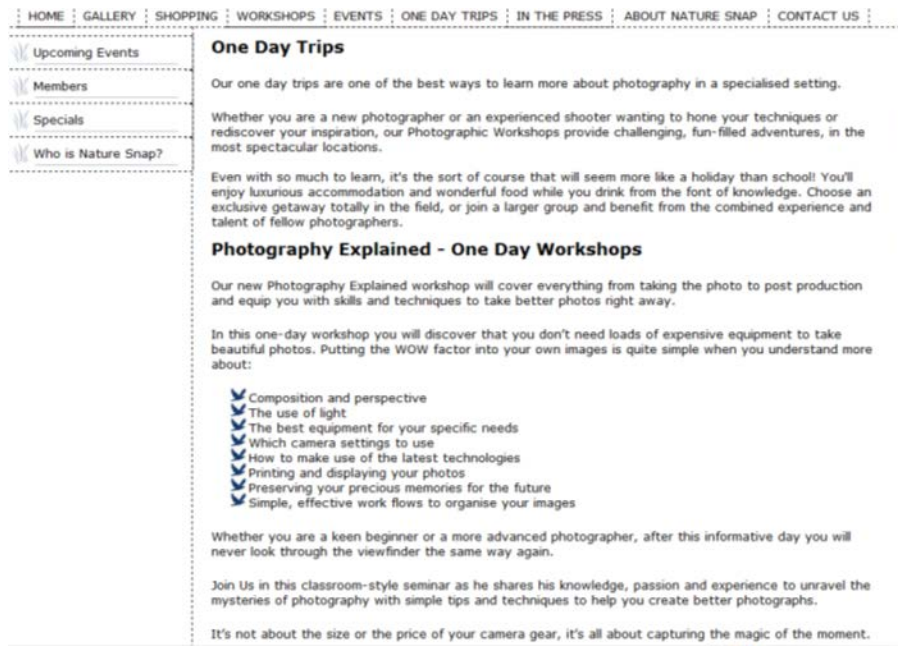
Figure 35 - The formatted paragraphs



Notice that the unordered list sticks out and it would be nice to change the bullet style from the usual dot.

- 9 Create a new compound rule for the **UL** in the content div **#content ul**.
- 10 First get the position right by shifting the text down and to the left. Again use your preference or use the setting for the design **60px left margin** and **20px top margin**. Press **apply** to see the application of the margins.
- 11 In the images folder there is an image called **bullet.jpg**. Go to the list section and select the **bullet.jpg** and the image style and set the position to **inside**.
- 12 Click **apply** and the design should look like Figure 36.

Figure 36 - The formatted paragraphs



- 13 The last piece of text that has to be formatted is the footer text. Click in the footer and change any existing text to “**Copyright 2016 – Nature Snap Photography**”.
- 14 Create a new class called **.copyright**.
- 15 Set the text alignment to **center**.
- 16 Set the font size to **10px**.
- 17 Click **OK**.
- 18 Highlight the copyright text. In the properties tab click on the **Class dropdown** and select **copyright**. The text should change to match the rule.

Step 9 – Images on the page and as backgrounds

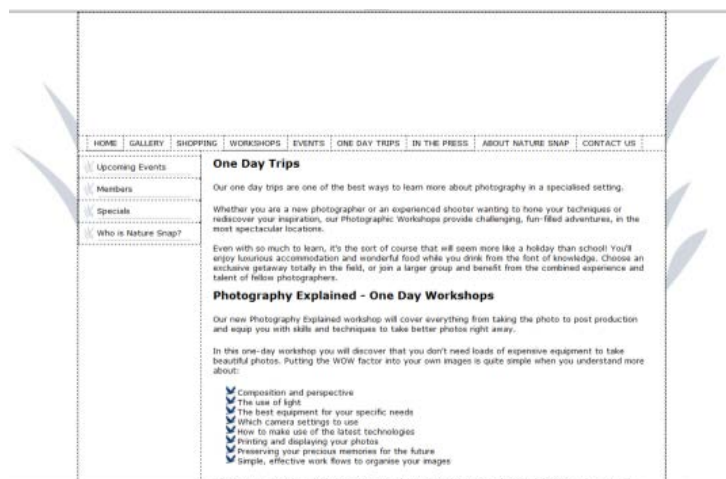
Let's now look at the use of images, both as background to DIVs and inserted in different positions on the page. Before the use of CSS it was quite complicated to get the page background to do what you wanted. With CSS it is now much easier.

Setting the background image

We are going to make the image **background.jpg** the page background.

- 1 Open the **body** CSS rule.
- 2 In the background section set the background to **white (#FFF)**.
- 3 Browse and select the **background.jpg** image to be used as the background.
- 4 In **background-repeat** select **no-repeat**.
- 5 In **background-attachment** select **scroll**.
- 6 The **background-position(X)** set it to **center**.
- 7 The **background-position(Y)** set it to **top**.
- 8 Press **apply** to see the changes. It should look like Figure 37.

Figure 37 - the background applied



What did you just do?

What happened is that the background, Figure 38, was placed as the background of the **body** in the top centre of the page. The wrapper and all of the content sit on top of this and hides parts of the background it covers.

Figure 38 - The background image



- 9 Open up the background settings again and change some of the background settings to see what changes they make. At least try the repeat, and the fixed and scroll settings so you know how they work. Make sure you set them back to how they were.
- 10 Change the background on the **#header** rule to put the **header.jpg** as a fixed image background. See the images dimensions to get the box height.

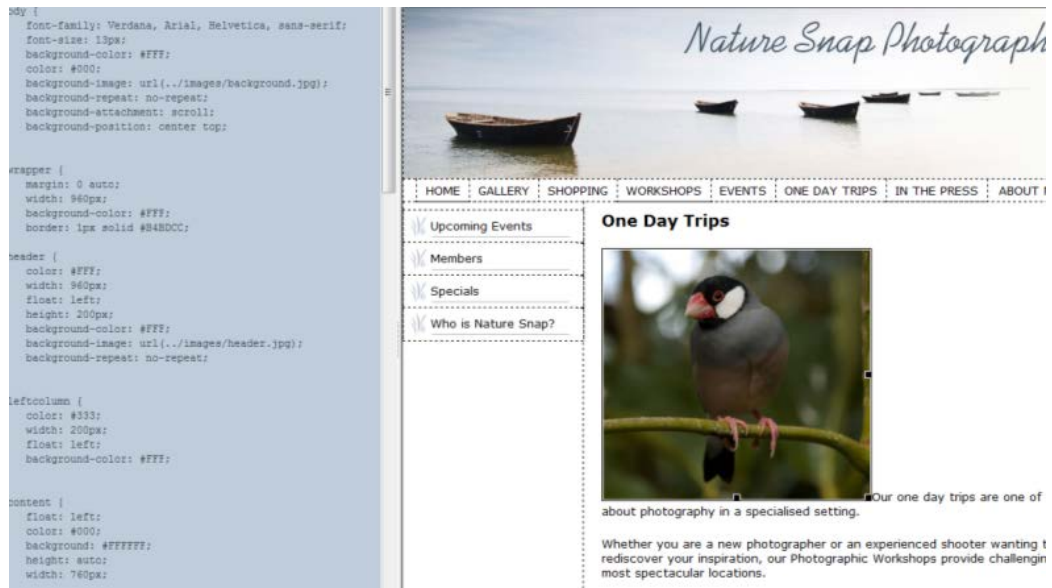
Page Images

Before CSS, web developers had to use tables to layout the text and images. We are now going to create two CSS rules to allow images to be put on the left and right of an area/page.

- 1 Create a class called **.imageleft**.
- 2 Set the float to **left**.
- 3 Set the margin around the image to be **15px**.
- 4 Create a **double** border **medium** width with a colour **#333**.
- 5 Click OK.
- 6 Create a class called **.imageright**. Use the same setting above and float the image to the right instead of left.

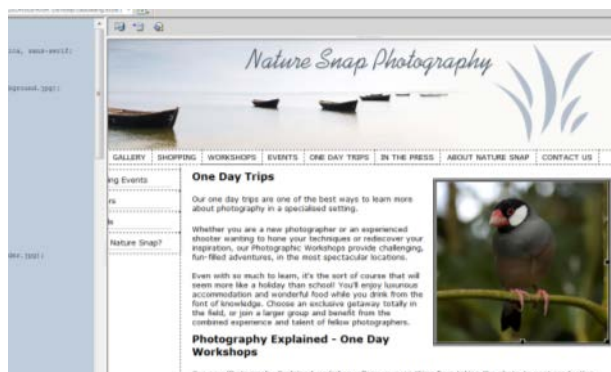
- 7 Now insert the **java_finch.jpg** image just under the text **One Day Trips** (Figure 39).

Figure 39 - Image on page with CSS rule applied



- 8 Click on the image and in the **property** tab, select the **class** drop down and select **imageright**. It will look out of place. It needs to be dragged up to the top in front of the text **One Day Trips** (Figure 40).

Figure 40 - Image with the CSS rule applied



- 9 Add in two more images (**fur_seal.jpg** and **striated_heron.jpg**), 1 to the left and one to the right so that it looks similar to Figure 41.

Figure 41 - Images floated left and right

- Upcoming Events
- Members
- Specials
- Who is Nature Snap?


One Day Trips

Our one day trips are one of the best ways to learn more about photography in a specialised setting.

Whether you are a new photographer or an experienced shooter wanting to hone your techniques or rediscover your inspiration, our Photographic Workshops provide challenging, fun-filled adventures, in the most spectacular locations.

Even with so much to learn, it's the sort of course that will seem more like a holiday than school! You'll enjoy luxurious accommodation and wonderful food while you drink from the font of knowledge. Choose an exclusive getaway totally in the field, or join a larger group and benefit from the combined experience and talent of fellow photographers.

Photography Explained - One Day Workshops



Our new Photography Explained workshop will cover everything from taking the photo to post production and equip you with skills and techniques to take better photos right away.

In this one-day workshop you will discover that you don't need loads of expensive equipment to take beautiful photos. Putting the WOW factor into your own images is quite simple when you understand more about:

- ✓ Composition and perspective
- ✓ The use of light
- ✓ The best equipment for your specific needs
- ✓ Which camera settings to use
- ✓ How to make use of the latest technologies
- ✓ Printing and displaying your photos
- ✓ Preserving your precious memories for the future
- ✓ Simple, effective work flows to organise your images

Whether you are a keen beginner or a more advanced photographer, after this informative day you will never look through the viewfinder the same way again.

Join us in this classroom-style seminar as he shares his knowledge, passion and experience to unravel the mysteries of photography with simple tips and techniques to help you create better photographs.


It's not about the size or the price of your camera gear, it's all about capturing the magic of the moment.

Numbers are strictly limited, so don't delay. Book your ticket now to join in what is guaranteed to be a great day spent with like-minded people, as you discover how to instantly improve your images.

Coastal Exposure Workshop

Spaces are strictly limited in this hands-on workshop, which will take place on January 26 in one of our favourite spots in Australia - the Mid-North Coast of NSW, just a few hours north of Sydney.

You can take advantage of the wealth of our teams combined 50 years of photographic experience and learn how to capture better images. The beautiful Mid-North Coast affords an array of spectacular scenery, from beaches to lush green valleys, to rocky waterfalls



Summary

In this topic you have learned how to identify and document appropriate styles that are to be controlled by the CSS including:

- > How floating works
- > How to set margins and padding
- > How to make lists go horizontal and vertical
- > How to make a menu out of a list
- > How to set background images and colours
- > How to redefine a HTML tag
- > How to make a CSS class
- > How to make a Compound class
- > What an ID is
- > The correct order for CSS Pseudo classes
- > How to use the Dreamweaver CSS panel to create & change CSS styles
- > What the auto term means when referring to margins
- > How to change bullets in lists to images.




Topic 3 - Validate CSS

HTML validators operate by comparing the markup on a web page to the W3C standards. The standards vary depending upon the declared version and so the validator will start by reading the DOCTYPE declaration to see which set of standards to apply.

Once the validator has read the page and determined the applicable standards it looks for such things as missing opening or closing tags, missing quotation marks and other hand-coding errors. The validator then provides a report indicating whether the coding is correct or if it contains errors, which will be listed.

Sometimes, one error, such as neglecting to close a tag, can cause a cascade of errors through the page, producing dozens or even hundreds of noted errors. However, when the page author addresses the first error listed it will also eliminate the "cascade errors".

The W3C Markup Validation Service checks the markup validity of web documents in HTML, XHTML, SMIL, MathML, etc. You can also validate specific content such as RSS/Atom feeds or CSS stylesheets, MobileOK content, or find broken links.

	ADDITIONAL RESOURCES	RECOMMENDED
<p>For more information on W3C validation visit:</p> <ul style="list-style-type: none">> http://en.wikipedia.org/wiki/W3C_Markup_Validation_Service <p>To use a W3C validation service, try:</p> <ul style="list-style-type: none">> http://validator.w3.org/> http://jigsaw.w3.org/css-validator/		

Here you can upload a file or copy and paste the contents of a file to check if it is valid or not. The validation service will check the file and report back any errors that it finds.

A link checker looks for issues in links, anchors and referenced objects in a web page, CSS style sheet, or on a whole website. For best results, it is recommended to first ensure that the documents checked use Valid XHTML, HTML and CSS.

Summary

In this topic you have learnt how to validate CSS and:

- > Test that the web site functions correctly and that the styles satisfy the purpose of the document
- > Validate the web pages and css documents at <http://validator.w3.org/> and <http://jigsaw.w3.org/css-validator/>
- > Test that the web site functions correctly using different browsers.