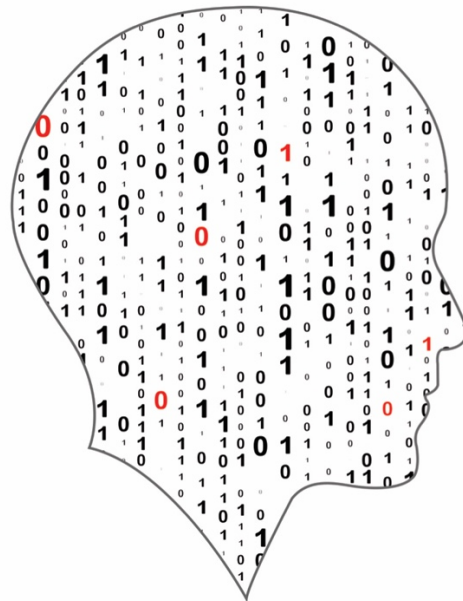




Professional Development  
Service for Teachers

An tSeirbhís um Fhorbairt  
Ghairmiúil do Mhúinteoirí



LEAVING CERTIFICATE  
**COMPUTER SCIENCE**

## **Fundamental Skills Development**

*HTML, CSS & UX DESIGN*



© PDST 2019

This work is made available under the terms of the Creative Commons Attribution Share Alike 3.0 Licence <http://creativecommons.org/licenses/by-sa/3.0/ie/>. You may use and re-use this material (not including images and logos) free of charge in any format or medium, under the terms of the Creative Commons Attribution Share Alike Licence.

Please cite as: PDST, Leaving Certificate Computer Science, Web Development Skills Workshop, Dublin, 2019

## Manual Overview

The purpose of this manual to provide Phase One Leaving Certificate Computer Science (LCCS) teachers with the knowledge, skills and confidence to design and develop websites and web applications independently.

Although the manual will serve as support material for teachers who attend the Web Application Development Workshop component of our two-year CPD programme, it is envisaged that its real value will only come into play in the months after the workshops have been delivered. Beyond these workshops, the manual may be used as a basic reference for web development, but more importantly, as a teaching resource that might be used to promote in teachers, a constructivist pedagogic orientation towards the planning and teaching of web development in the LCCS classroom.

The manual itself is divided into five separate sections and is divided into three separate hardcopy documents. This is Part A.

### **Part A**

#### **Section 1 – HTML**

#### **Section 2 – Cascading Style Sheets**

#### **Section 3 – UX Design**

### **Part B**

#### **Section 4 – JavaScript**

### **Part C**

#### **Section 5 – Databases**

# Section 1

## HTML

## Contents

<b>Introducing HTML</b>	<b>07</b>
<ul style="list-style-type: none"><li>• How the web works</li><li>• What is HTML?</li><li>• My first page</li><li>• Web browsers</li><li>• HTML tags</li><li>• HTML page structure</li><li>• HTML versions</li><li>• Exercise 1 - first page</li><li>• Reflection exercise</li></ul>	
<b>Basic Text Formatting</b>	<b>18</b>
<ul style="list-style-type: none"><li>• How white space is collapsed</li><li>• Creating headings using the &lt;h&gt; elements</li><li>• Structuring headings and paragraphs</li><li>• Creating paragraphs using the &lt;p&gt; element</li><li>• Creating line breaks using the &lt;br&gt; element</li><li>• Creating preformatted text using the &lt;pre&gt; element</li><li>• Exercise 2 - Café</li><li>• Reflection exercise</li></ul>	
<b>Working with Lists</b>	<b>24</b>
<ul style="list-style-type: none"><li>• Unordered lists using the &lt;ul&gt; and &lt;li&gt; elements</li><li>• Ordered lists using the &lt;ol&gt; and &lt;li&gt; elements</li><li>• Using the start attribute to change the starting number in ordered lists</li><li>• Specify a marker with the type attribute</li><li>• Definitions lists with the &lt;dl&gt;, &lt;dt&gt; and &lt;dd&gt; attributes</li><li>• Nesting lists</li></ul>	
<b>Fine-tuning Your Text</b>	<b>29</b>
<ul style="list-style-type: none"><li>• The &lt;strong&gt; element</li><li>• The &lt;cite&gt; element</li><li>• The &lt;q&gt; element</li><li>• The &lt;blockquote&gt; element</li><li>• The &lt;dfn&gt; element</li><li>• The &lt;code&gt; element</li><li>• The &lt;var&gt; element</li><li>• The &lt;samp&gt; element</li><li>• Exercise 3 - Café Recipes</li><li>• Reflection exercise</li></ul>	

<b>Links and Navigation</b>	<b>34</b>
<ul style="list-style-type: none"> <li>• Creating a basic link to a page in the same folder using the &lt;a&gt; element</li> <li>• Creating a link to an external web site</li> <li>• Link colours</li> <li>• A link to send an email</li> <li>• Create a bookmark</li> <li>• Exercise 4 - Adding links between pages</li> </ul>	
<b>Tables</b>	<b>38</b>
<ul style="list-style-type: none"> <li>• A basic table</li> <li>• Exercise 5 - Opening hours table</li> </ul>	
<b>Images &amp; Video</b>	<b>40</b>
<ul style="list-style-type: none"> <li>• Adding an image using the &lt;img&gt; element</li> <li>• The height and width attributes</li> <li>• Using images as links</li> <li>• Embedding a YouTube video</li> <li>• Exercise 6 - Cafe</li> <li>• Reflection exercise</li> </ul>	
<b>Forms</b>	<b>45</b>
<ul style="list-style-type: none"> <li>• Text controls</li> <li>• Option controls</li> <li>• Checkboxes</li> <li>• Radio buttons</li> <li>• Lists</li> <li>• Buttons</li> <li>• Labelling controls</li> <li>• Grouping controls</li> <li>• Exercise 7 - Forms</li> </ul>	
<b>Breakout Exercises</b>	<b>54</b>
<ul style="list-style-type: none"> <li>• Breakout A - Monster mark up</li> <li>• Breakout B - Mark up a formal letter</li> </ul>	
<b>HTML Glossary</b>	<b>63</b>
<b>HTML Resources</b>	<b>64</b>

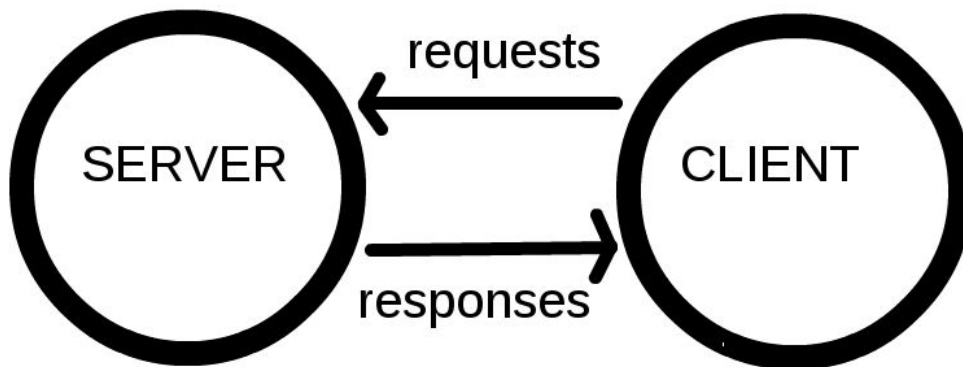
## Introducing HTML

### How the web works

What happens when you view a webpage in a web browser on your computer or phone? This detail is not essential to writing web code in the short term, but before long you'll really start to benefit from understanding what's happening in the background.

#### Clients and servers

Computers connected to the web are called clients and servers.



- Clients are the typical web user's internet-connected devices (for example, your computer connected to your Wi-Fi, or your phone connected to your mobile network) and web-accessing software available on those devices (usually a web browser like Firefox or Chrome).
- Servers are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.

#### The other parts of the toolbox

The client and server we've described above don't tell the whole story. There are many other parts involved, and we'll describe them below.

For now, let's imagine that the web is a road. On one end of the road is the client, which is like your house. On the other end of the road is the server, which is like a shop you want to buy something from.

In addition to the client and the server, we need to discuss:

- The internet connection: this allows you to send and receive data on the web. It's basically like the street between your house and the shop.
- TCP/IP: Transmission Control Protocol and Internet Protocol are communication protocols that define how data should travel across the web. This is like the transport mechanisms that let you place an order, go to the shop, and buy your goods.
- DNS: Domain Name Servers are similar to an address book for websites. When you type a web address in your browser, the browser looks at the DNS to find the website's real address before it can retrieve the website. The browser needs to find out which server the website lives on, so it can send HTTP messages to the right place (see below). This is like looking up the address of the shop so you can access it.
- HTTP: Hypertext Transfer Protocol is an application protocol that defines a language for clients and servers to speak to each other. This is the language you use to order your goods.
- Component files: A website is made up of many different files, which are different parts of the goods you buy from the shop. These files come in two main types:
  - Code files: Websites are built primarily from HTML, CSS, and JavaScript.
  - Assets: This is a collective name for all the other items that makes up a website, such as images, music, video, Word documents, and PDFs.



## So what happens?

When you type a web address into your browser (for our analogy that's like walking to the shop):

1. The browser goes to the DNS server, and finds the real address of the server that the website lives on.
2. The browser sends an HTTP request message to the server, asking it to send a copy of the website to the client. This message, and all other data sent between the client and the server, is sent across your internet connection using TCP/IP.
3. If the server approves the client's request, the server sends the client a "200 OK" message, which means "Of course you can look at that website! Here it is", and then starts sending the website's files to the browser as a series of small chunks called data packets.
4. The browser assembles the small chunks into a complete website and displays it (the goods arrive at your door!).

## DNS explained

Real web addresses aren't the memorable strings you type into your address bar to find your favourite websites. They are special numbers that look like this: 63.245.215.20.

This is called an IP address, and it represents a unique location on the web. However, it's not very easy to remember. That's why Domain Name Servers were invented. These are special servers that match up a web address you type into your browser (like "mozilla.org") to the website's real (IP) address. Websites can be reached directly via their IP addresses. You can find the IP address of a website by typing its domain into a tool like an IP Checker.

## Packets

Earlier we used the term "packets" to describe the format in which the data is sent from server to client. Basically, when data is sent across the web, it is sent as thousands of small chunks, so that many different web users can download the same website at the same time. If websites were sent as single big chunks, only one user could download one at a time, which obviously would make the web very inefficient and not much fun to use.

## What is HTML?

HTML is the standard markup language for creating Web pages. It is not a programming language; it is a *markup language* used to tell your browser how to structure the pages you visit. It can be as complicated or as simple as the web developer wishes it to be.



- HTML stands for Hyper Text Markup Language
- HTML describes the structure of web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page.

**TEACHER TIP**

Tags in HTML are case-insensitive, i.e. they can be written in uppercase or lowercase. For example, a <title> tag could be written as <title>, <TITLE>, <Title>, <TiTIE>, etc., and it will work fine. Best practice, however, is to write all tags in lowercase for consistency and readability .

## First Web Page

# My First Heading

My first paragraph.

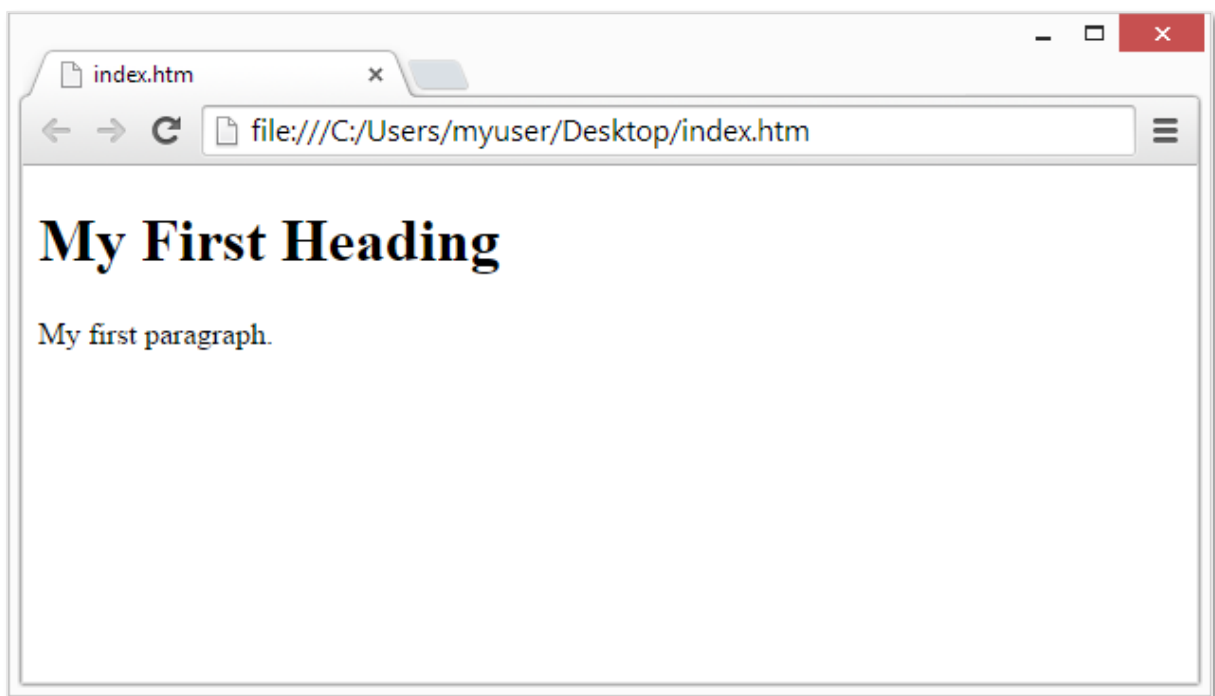
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Page Title</title>
5 </head>
6 <body>
7
8 <h1>My First Heading</h1>
9 <p>My first paragraph.</p>
10
11 </body>
12 </html>
13
14
15
```

- The <!DOCTYPE html> declaration defines this document to be HTML5.
- The <html> element is the root element of an HTML page.
- The <head> element contains meta information about the document.
- The <title> element specifies a title for the document.
- The <body> element contains the visible page content.

- The <h1> element defines a large heading.
- The <p> element defines a paragraph.

## Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari) is to read HTML documents and display them. The browser does not display the HTML tags, but uses them to determine how to display the document:

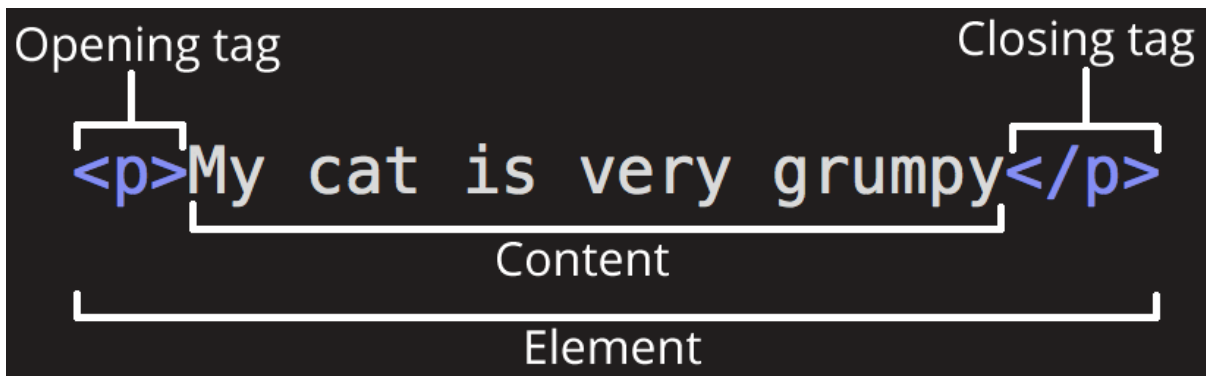


## HTML Tags

HTML tags are element names surrounded by angle brackets:

**<tagname> content goes here...</tagname>**

- HTML tags normally come in pairs like <p> and </p>
- The first tag in a pair is the start tag, the second tag is the end tag
- The end tag is written like the start tag, but with a forward slash inserted before the tag name



The main parts of our element are:

1. **The opening tag:** This consists of the name of the element (in this case, p), wrapped in opening and closing angle brackets. This states where the element begins or starts to take effect.
2. **The closing tag:** This is the same as the opening tag, except that it includes a forward slash before the element name. This states where the element ends. Failing to include a closing tag is a common beginner error and can lead to strange results.
3. **The content:** This is the content of the element.
4. **The element:** The opening tag plus the closing tag plus the content equals the element.

## HTML Page Structure

Most structured text consists of headings and paragraphs, whether you are reading a story, a newspaper, a college textbook, a magazine, etc.



- Users looking at a web page tend to scan quickly to find relevant content, often just reading the headings to begin with (we usually spend a very short time on a web page). If they can't see anything useful within a few seconds, they'll likely get frustrated and go somewhere else.
- Search engines indexing your page consider the contents of headings as important keywords for influencing the page's search rankings. Without headings, your page will perform poorly in terms of SEO (Search Engine Optimisation).
- Severely visually impaired people often don't read web pages; they listen to them instead. This is done with software called a screen reader. This software provides ways to get fast access to given text content. Among the various techniques used, they provide an outline of the document by reading out the headings, allowing their

users to find the information they need quickly. If headings are not available, they will have to listen to the whole document read out loud.

- To style content with CSS, or make it do interesting things with JavaScript, you need to have elements wrapping the relevant content, so CSS/JavaScript can effectively target it.

### Visualisation of an HTML page structure

The HTML head is the contents of the `<head>` element - unlike the contents of the `<body>` element (which are displayed on the page when loaded in a browser), the head's content is not displayed on the page. Instead, the head's job is to contain metadata about the document.

Metadata is data that describes data, and HTML has an "official" way of adding metadata to a document - the `<meta>` element. There are a lot of different types of `<meta>` elements that can be included in your page's `<head>`, but we won't try to explain them here.

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

## HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

### HTML5

HTML5 is the new web standard. It follows HTML 4 (which came out in 1997) and XHTML. Since the introduction of HTML4, a lot has happened with the web and something needed to be done to address all the new technologies and latest multimedia. HTML5 is the result of cooperation that began in 2006 between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The basic aim of HTML5 is to provide two things (1) to improve the language and (2) to support the latest multimedia. In order to accomplish this, some ground rules were established by the W3C and WHATWG. Among them were to reduce the need for external plug-ins (such as Flash plug-ins), better handling of errors, and more markup elements (tags) to replace scripting. HTML5 should also be device independent (that is, understood by computers and the many devices in existence today) while also keeping it easily readable.



## Exercise 1

Create the following simple web page. Please type out the text.

### About Google

Google is best known for its search engine, although Google now offers a number of other services.

Google's mission is to organize the world's information and make it universally accessible and useful.

Its founders Larry Page and Sergey Brin started Google at Stanford University.

## Reflection

Reflect on what you have learned about HTML so far.



Use the space below to write **five** things about HTML.

1.

---

2.

---

3.

---

4.

---

5.

---

## Basic Text Formatting

### How white space is collapsed

Most structured text consists of headings and paragraphs, whether you are reading a story, a newspaper, a college textbook, a magazine, etc.

This paragraph below shows how multiple spaces between words are treated as a single space. This is known as white space collapsing, and the big spaces between some of the words will not appear in the browser. It also demonstrates how the browser will treat multiple carriage returns (new lines) as a single space, too.

```
1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>White Space Collapsing and Line Wrapping</title>
5 </head>
6
7 <body>
8 <p>This paragraph shows how multiple spaces between words are treated as a single space. This is known as white space collapsing,
9   and the big spaces between some of the words will not appear in the browser.
10
11
12
13 It also demonstrates how the browser will treat multiple carriage returns (new lines) as a single space, too.</p>
14 </body>
15 </html>
16
17
18
```

This paragraph shows how multiple spaces between words are treated as a single space. This is known as white space collapsing, and the big spaces between

## Creating headings using the hn elements

The HTML <h1>–<h6> elements represent six levels of section headings. <h1> is the highest section level and <h6> is the lowest.

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Six Levels of Headings in XHTML</title>
6 </head>
7 <body>
8   <h1>Heading 1</h1>
9   <h2>Heading 2</h2>
10  <h3>Heading 3</h3>
11  <h4>Heading 4</h4>
12  <h5>Heading 5</h5>
13  <h6>Heading 6</h6>
14 </body>
15 </html>
16
17
```

# Heading 1

## Heading 2

### Heading 3

#### Heading 4

##### Heading 5

###### Heading 6

## Structuring headings and paragraphs

Almost every document you create will contain some form of text.

### White Space and Flow

Before you start to mark up your text, it is best to understand what HTML does when it comes across spaces and how browsers treat long sentences and paragraphs of text.

### Creating Headings

No matter what sort of document you are creating, most documents have headings in some form or other.

```
1
2 <!doctype html>
3 <html>
4 <head>
5   <meta charset="utf-8">
6   <title>Using Headings to Structure Text</title>
7 </head>
8
9 <body>
10 <h1>Basic Text Formatting</h1>
11 <p> This section is going to address the way in which you mark up text. Almost every document you create will contain some form of text, so this
12 will be a very important section. </p>
13
14 <h2>White Space and Flow</h2>
15 <p> Before you start to mark up your text, it is best to understand what HTML does when it comes across spaces and how browsers
16 treat long sentences and paragraphs of text.</p>
17
18 <h2>Creating Headings</h2>
19 <p> No matter what sort of document you are creating, most documents have headings in some form or other...</p>
20 </body>
21 </html>
22
23
```

## Creating paragraphs using the <p> element

The HTML <p> element represents a paragraph. Paragraphs are usually represented in visual media as blocks of text separated from adjacent blocks by blank lines and/or first-line indentation, but HTML paragraphs can be any structural grouping of related content, such as images or form fields.

```

1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Paragraphs</title>
6 </head>
7
8 <body>
9 <p>Here is a paragraph of text.</p>
10 <p>Here is a second paragraph of text.</p>
11 <p>Here is a third paragraph of text.</p>
12 </body>
13 </html>
14
15

```

## Creating line breaks using the <br> element

When you want to start a new line you can use the line break element. So, the next word will appear on a new line. Without the line break element, new lines are started only when the sentence reaches the end of the screen; this sentence should be long enough to wrap on your screen. Try resizing your browser window and see how the position where the line wraps onto a new line changes.

```

1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Creating Line Breaks Using the &lt;br /&gt; Element</title>
6 </head>
7
8 <body>
9 <p>When you want to start a new line you can use the line break element. So, the next<br>
10 word will appear on a new line.</p>
11 <p>Without the line break element, new lines are started only when the sentence reaches the end of the screen; this sentence should be long enough
12 to wrap on your screen. Nbw try resizing your browser window and see how the position where the line wraps onto a new line changes.</p>
13 <p>Some Web designers also use this element to control the layout of the document and add extra white space. You can use
14 multiple line break elements to create gaps of several lines, like I am about to do <br><br><br><br><br>here because this
15 text is still in the same paragraph element. Rather than using the line break element to add white space into your documents, i
16 t is better to use CSS.</p>
17 </body>
18 </html>
19
20

```

## Creating preformatted text using the `<pre>` element

The HTML `<pre>` element represents preformatted text which is to be presented exactly as written in the HTML file. The text is typically rendered using a non-proportional ("monospace") font. Whitespace inside this element is displayed as written. The following text is written inside a `<pre>` element.

```
1
2 <!doctype html>
3 <html>
4 <head>
5   <meta charset="utf-8">
6   <title>Creating Prefromatted Text Using the &lt;pre&gt; Element</title>
7 </head>
8
9 <body>
10 <p>The following text is written inside a &lt;pre&gt; element. Multiple spaces should be preserved and the line breaks should appear where
11 they do in the source document.</p>
12
13 <pre>
14 function testFunction(strText){
15   console.log(strText)
16 }
17 </pre>
18
19 <p>The content of the &lt;pre&gt; element is most likely displayed in a monospaced font.</p>
20
21 </body>
22 </html>
23
24
```

## Exercise 2 - Example Café

Create the following simple web page for a Café. Use the text file provided to avoid typing it out.

### **EXAMPLE CAFE**

Welcome to example cafe. We will be developing this site throughout the book.

#### **A community cafe serving home cooked, locally sourced, organic food**

With stunning views of the ocean, Example Cafe offers the perfect environment to unwind and recharge the batteries.

Our menu offers a wide range of breakfasts, brunches and lunches, including a range of vegetarian options.

Whether you sip on a fresh, hot coffee or a cooling smoothie, you never need to feel rushed - relax with friends or just watch the world go by.

#### **This weekend's special brunch**

This weekend, our season of special brunches continues with scrambled egg on an English muffin. Not for the faint-hearted, the secret to these eggs is that the

## Reflection

Reflect on what you have learned about formatting text in HTML.



Use the space below to write three different tags for formatting text.

1.

---

2.

---

3.

---

## Working with Lists

Now let's turn our attention to lists. Lists are everywhere in life - from your shopping list to the list of directions you subconsciously follow to get to your house every day. Lists are everywhere on the Web too, and we've got three different types to worry about.

### Unordered lists using the `<ul>` and `<li>` elements

An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with bullets (small black circles) by default.

Here is an unordered list, which is just a bulleted list:

- Bullet point number one
- Bullet point number two
- Bullet point number three

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Unordered Lists</title>
6 </head>
7
8 <body>
9
10 <p>Here is an unordered list, which is just a bulleted list:</p>
11
12 <ul>
13   <li>Bullet point number one</li>
14   <li>Bullet point number two</li>
15   <li>Bullet point number three</li>
16 </ul>
17
18 </body>
19 </html>
20
21
```



## Ordered lists using the `<ol>` and `<li>` elements

The HTML `<ol>` element represents an ordered list of items, typically rendered as a numbered list.

Here is an ordered list:

1. Point number one
2. Point number two
3. Point number three

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Ordered Lists</title>
6 </head>
7
8 <body>
9
10 <p>Here is an ordered list:</p>
11
12 <ol>
13   <li>Point number one</li>
14   <li>Point number two</li>
15   <li>Point number three</li>
16 </ol>
17
18 </body>
19 </html>
20
21
```

## Using the start attribute to change the starting number in ordered lists

The start attribute specifies the start value of the first list item in an ordered list.

Here is an ordered list:

4. Point number one
5. Point number two
6. Point number three

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Ordered Lists</title>
6 </head>
7
8 <body>
9
10 <p>Here is an ordered list:</p>
11
12 <ol start="4">
13   <li>Point number one</li>
14   <li>Point number two</li>
15   <li>Point number three</li>
16 </ol>
17
18 </body>
19 </html>
20
21
```

## Definitions lists with the <dl>, <dt> and <dd> attributes

The <dd> tag is used to describe a term/name in a description list. The <dd> tag is used in conjunction with <dl> (defines a description list) and <dt> (defines terms/names). Inside a <dd> tag you can put paragraphs, line breaks, images, links, lists, etc.

Here is a definition list:

### Unordered List

A list of bullet points.

### Ordered List

An ordered list of points, such as a numbered set of steps.

### Definition List

A list of terms and definitions.

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Definition Lists</title>
6 </head>
7
8 <body>
9
10 <p>Here is a definition list:</p>
11
12 <dl>
13   <dt>Unordered List</dt>
14   <dd>A list of bullet points.</dd>
15   <dt>Ordered List</dt>
16   <dd>An ordered list of points, such as a numbered set of steps.</dd>
17   <dt>Definition List</dt>
18   <dd>A list of terms and definitions.</dd>
19 </dl>
20
21 </body>
22 </html>
23
24
25
```

## Nesting lists

A nested list is a list within a list. If you've ever created a bulleted outline in a word processing document you probably used a variety of indentations and bullet point types to denote items that were subpoints of another item in the outline. This is the effect we're going for when we create nested lists.

Here is a nested ordered list:

- I. Item one
- II. Item two
- III. Item three
- IV. Item four
  - i. Item 4.1
  - ii. Item 4.2
  - iii. Item 4.3
- V. Item Five

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Nested ordered lists</title>
6 </head>
7
8 <body>
9
10 <p>Here is a nested ordered list:</p>
11
12 <ol type="I">
13   <li>Item one</li>
14   <li>Item two</li>
15   <li>Item three</li>
16   <li>Item four
17     <ol type="i">
18       <li>Item 4.1</li>
19       <li>Item 4.2</li>
20       <li>Item 4.3</li>
21     </ol>
22   </li>
23   <li>Item Five</li>
24 </ol>
25
26 </body>
27 </html>
28
```

## Fine-tuning Your Text

### The `<strong>` element

The HTML Strong Importance Element (`<strong>`) indicates that its contents have strong importance, seriousness, or urgency.

In the following sentence, the words can cause blindness are contained inside the `<strong>` element.

```
1 <!doctype html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Creating Emphasis Using the <em> and <strong> Elements</title>
6 </head>
7 <body>
8 <p>In the following sentence the words <strong>can cause blindness</strong> are contained inside the <strong> element.</p>
9
10 </body>
11 </html>
12
13
```

### The `<cite>` element

The `<cite>` tag defines the title of a work (e.g. a book, a song, a movie, a TV show, a painting, a sculpture, etc.). It is used for quoting text from another source.

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>Citations</title>
5 </head>
6
7 <body>
8
9 <h2>The <cite> Element for Quoting Text From Another Source</h2>
10 <p>This chapter is taken from <cite>Beginning Web Programming</cite>.</p>
11
12 </body>
13 </html>
14
15
```

## The <q> element

If your quotation is going to appear only within a sentence, you should use the <q> element.

The following sentence uses the <q> element to form a quote:

```

1
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Quotes</title>
6 </head>
7
8 <body>
9 <h1>The &lt;q&gt; Element for Small Quotations</h1>
10 <p>If your quotation is going to appear only within a sentence, you should use the &lt;q&gt; element. The following sentence u
11 ses the &lt;q&gt; element to form a quote:</p>
12 <p>As Dylan Thomas said, <q>Somebody's boring me. I think it's me</q>.</p>
13
14 </body>
15 </html>
16
17
  
```

## The <blockquote> element

The blockquote element defines "a section [within a document] that is quoted from another source". The blockquote element is used to indicate the quotation of a large section of text from another source.

Using the default HTML styling of most web browsers, it will indent the right and left margins both on the display and in printed form, but this may be overridden by Cascading Style Sheets (CSS).

```

1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Quotes</title>
6 </head>
7
8 <body>
9 <h1>The &lt;blockquote&gt; Element</h1>
10 <p>The following description of the blockquote element is taken from the WHATWG site:</p>
11 <blockquote> The blockquote element represents a section that is quoted from another source. Content inside a blockquote
12 must be quoted from another source, whose address, if it has one, may be cited in the cite attribute.</blockquote>
13
14 </body>
15 </html>
16
17
  
```

## The <dfn> element

The HTML Definition element (<dfn>) is used to indicate the term being defined within the context of a definition phrase or sentence. The following sentence uses a <dfn> element for the important term HTML.

```

1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Definitions</title>
6 </head>
7
8 <body>
9 <p>The following sentence uses a &lt;dfn&gt; element for the important term <b>HTML</b>.</p>
10 <p>This book teaches you how mark up your documents for the web using <dfn>HTML</dfn>.</p>
11 <br>
12
13 </body>
14 </html>
15
16
  
```

## The <code> element

The <code> Element For Adding Code to Your Web Pages

```

1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Representing Code</title>
6 </head>
7
8 <body>
9 <h2>The &lt;code&gt; Element For Adding Code to Your Web Pages</h2>
10 <p>The following line appears inside a &lt;code&gt; element.</p>
11 <p><code>&lt;h1&gt;This is a primary heading&lt;/h1&gt;</code></p>
12 </body>
13 </html>
14
15
16
  
```

## The <var> element

The HTML Variable element (<var>) represents the name of a variable in a mathematical expression or a programming context. It's typically presented using an italicized version of the current typeface.

```
1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Representing Programming Variables</title>
5 </head>
6
7 <body>
8 <h1>The &lt;var>; Element for Programming Variables</h1>
9 <p>The following line is written inside a &lt;code>; element, while <b>user-name</b> is written inside a &lt;var>; element.</p>
10 <p><code>console.log( "<var>user-name</var>" )</code></p>
11 <p>As you can see, the content of the &lt;var>; element is italicized.</p>
12 </body>
13 </html>
14
15
```

## The <samp> element

The <samp> element is used to display output from a process, such as an error message from a computer script. It was originally designed for technical documentation, and renders the content of the element in a monospace font.

```
1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Representing Programming Variables</title>
5 </head>
6
7 <body>
8 <h1>The &lt;samp>; Element for Sample Program Output</h1>
9 <p>The following line uses the &lt;samp>; element to indicate the output from a script or program.</p>
10 <p><samp>This is the output from our test script.</samp></p>
11
12 </body>
13 </html>
14
15
```

### TEACHER TIP!

Develop pages in small steps.

Save and test your new page every time you add or change tags.



## Exercise 3 - Example Café Recipes

Create the following simple web page for the Example Café. The text will be provided to avoid typing.

### Example Cafe Recipes - World's Best Scrambled Eggs

I adapted this recipe from a book called [Sydney Food](#) by Bill Grainger. Ever since tasting these eggs on my 1<sup>st</sup> visit to Bill's restaurant in Kings Cross, eggs I have ever tasted.

This recipe is what I call a "very special breakfast"; just look at the ingredients to see why. It has to be tasted to be believed.

#### Ingredients

The following ingredients make one serving:

- 2 eggs
- 1 tablespoon of butter (10g)
- 1/3 cup of cream (2 3/4 fl ounces)
- A pinch of salt
- Freshly milled black pepper
- 3 fresh chives (chopped)

#### Instructions

1. Whisk eggs, cream, and salt in a bowl.
2. Melt the butter in a non-stick pan over a high heat (*taking care not to burn the butter*).
3. Pour egg mixture into pan and wait until it starts setting around the edge of the pan (around 20 seconds).
4. Using a wooden spatula, bring the mixture into the center as if it were an omelet, and let it cook for another 20 seconds.
5. Fold contents in again, leave for 20 seconds, and repeat until the eggs are only just done.
6. Grind a light sprinkling of freshly milled pepper over the eggs and blend in some chopped fresh chives.

You should only make a **maximum** of two servings per frying pan.

## Reflection

Reflect on what you have learned about fine-tuning your text in HTML.



Use the space below to write three different tags for fine-tuning your text.

## Links & Navigation

### Creating a basic link to a page in the same folder using the `<a>` element

- HTML links are hyperlinks.
- You can click on a link and jump to another document.
- When you move the mouse over a link, the mouse arrow will turn into a little hand.

In HTML, links are defined with the `<a>` tag:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>A basic link to another page</title>
6 </head>
7
8 <body>
9   <p>Return to the <a href="index.html">home page</a>.</p>
10 </body>
11
12 </html>
13
14
```

### Creating a link to an external Web site

The href attribute specifies the destination address (<https://www.rte.ie>) of the link. The link text is the visible part (RTE Website).

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>A basic link to an external web site</title>
6 </head>
7
8 <body>
9   <p>Why not visit the <a href="http://www.rte.ie/">RTE Web site</a>?</p>
10 </body>
11 </html>
12
13
```

Why not visit the [RTE Web site](http://www.rte.ie/)?

## Link Colours

By default, a link will appear like this (in all browsers):

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

You can change the default colours, by using CSS.

## A link to send an email

Using the *mailto* link it will allow you to send you an e-mail

```
<A HREF="mailto:name@mydomain.com"> Click Here To Email Me </A>
```

The *mailto* link is written in the same format as a hyperlink except you use *mailto:* in place of the *http://* and your e-mail address in place of the page address or URL. You must include the *</A>* code at the end of the line in order for the *mailto* link to work. There is NO space between the *mailto:* and the e-mail address.

```
1
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <meta charset="utf-8">
6   <title>A link to send an email</title>
7 </head>
8
9 <body>
10 <p><a href="mailto:name@example.com">name@example.com</a></p>
11 </body>
12 </html>
13
14
```

[name@example.com](mailto:name@example.com)

## Create a Bookmark

- HTML bookmarks are used to allow readers to jump to specific parts of a Web page.
- Bookmarks can be useful if your webpage is very long.
- To make a bookmark, you must first create the bookmark, and then add a link to it.
- When the link is clicked, the page will scroll to the location with the bookmark.

First, create a bookmark with the id attribute:

```
<h2 id="C4">Chapter 4</h2>
```

Then, add a link to the bookmark ("Jump to Chapter 4"), from within the same page:

```
<a href="#C4">Jump to Chapter 4</a>
```

## Example 4 Cafe - Adding links between pages

Open the following web page for a Café and make edits or changes to it. Use the file provided.

### EXAMPLE CAFE

HOME [MENU](#) [RECIPES](#) [CONTACT](#)

#### A community cafe serving home cooked, locally sourced, organic food

With stunning views of the ocean, Example Cafe offers the perfect environment to unwind and recharge the batteries.

Our menu offers a wide range of breakfasts, brunches and lunches, including a range of vegetarian options.

Whether you sip on a fresh, hot coffee or a cooling smoothie, you never need to feel rushed - relax with friends or just watch the world go by.

#### This weekend's special brunch

This weekend, our season of special brunches continues with scrambled egg on an English muffin. Not for the faint hearted, the secret to these eggs is that they are made v



Use the space below for rough work.

## Tables

An HTML table is defined with the `<table>` tag. Each table row is defined with the `<tr>` tag. A table header is defined with the `<th>` tag. By default, table headings are bold and centred. A table data/cell is defined with the `<td>` tag.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>A basic table</title>
6 </head>
7
8 <body>
9
10 <table border="1">
11   <tr>
12     <td>Row 1, Column 1</td>
13     <td>Row 1, Column 2</td>
14   </tr>
15   <tr>
16     <td>Row 2, Column 1</td>
17     <td>Row 2, Column 2</td>
18   </tr>
19 </table>
20
21 </body>
22 </html>
23
24
25
```

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

- Use the HTML `<table>` element to define a table
- Use the HTML `<tr>` element to define a table row
- Use the HTML `<td>` element to define a table data
- Use the HTML `<th>` element to define a table heading
- Use the HTML `<caption>` element to define a table caption

## Example 5 - Adding a table

Create the following web page for the Café. A text file will be provided.

example  
cafe

[HOME](#) [MENU](#) [RECIPES](#) [CONTACT](#)

### Contact

12 Sea View, Newquay, Cornwall, UK

[Find us on Google Maps](#)

[Email Example Cafe](#)

Opening hours for the Example Cafe in Newquay

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
<b>Breakfast</b>	7:00am - 10:00am	7:00am - 10:00am	7:00am - 10:00am	7:00am - 10:00am	7:00am - 11:00am	8:00am - 11:30pm	8:00am - 11:30pm
<b>Lunch</b>	11:30am - 2:30pm	11:30am - 2:30pm	11:30am - 2:30pm	11:30am - 2:30pm	11:30am - 2:30pm	11:30am - 3:30pm	11:30am - 3:30pm



Use the space below for rough work.

## Image & Video

### Adding an image using the <img> element

- In HTML, images are defined with the <img> tag.
- The <img> tag is empty, it contains attributes only, and does not have a closing tag.
- The src attribute specifies the URL (web address) of the image.

```
1
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <meta charset="utf-8">
6   <title>Image</title>
7 </head>
8
9 <body>
10
11 
12
13 </body>
14
15 </html>
16
17
```

### The height and width attributes

- You can use the width and height attributes.
- The width and height attributes are in pixels.



```
1
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <meta charset="utf-8">
6   <title>Fruit Pictures</title>
7 </head>
8 <body>
9   <p>Fixed size: width 130 height 130</p>
10  
11 <p>Enlarged: width 160 (no height specified)</p>
12  
13 <p>Stretched: width 80 height 150</p>
14  
15 </body>
16 </html>
17
18
```

Fixed size: width 130 height 130



Enlarged: width 160 (no height specified)



Stretched: width 80 height 150



## Using images as links

To use an image as a link, put the `<img>` tag inside the `<a>` tag.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Image as a link</title>
6 </head>
7
8 <body>
9
10 <a href="http://www.wrox.com"></a>
11
12 </body>
13
14 </html>
15
16
```

## Images in Another Folder

If not specified, the browser expects to find the image in the same folder as the web page.

However, it is common to store images in a sub-folder. You must then include the folder name in the `src` attribute:

```
` element in your web page
- Let the `src` attribute point to the video URL
- Use the `width` and `height` attributes to specify the dimension of the player
- Add any other parameters to the URL (see below)

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Adding a YouTube Video</title>
6 </head>
7 <body>
8   <iframe width="420" height="315"
9     src="http://www.youtube.com/embed/J---aiyznGQ"
10    frameborder="0" allowfullscreen></iframe>
11 </body>
12 </html>
13
14
```

## Reflection

Reflect on what you have learned about using HTML so far.



Use the space below to write your thoughts.

## Exercise 6 - Example Cafe

Create the following simple web page for a Café. All files will be provided.

The screenshot shows a website for 'example cafe'. At the top left, the logo 'example cafe' is displayed in blue, with navigation links 'HOME MENU RECIPES CONTACT' below it. The main heading is 'Home'. On the left, there is a photograph of a plate of food consisting of a fried tomato slice, scrambled eggs, and a muffin. To the right of the photo, the text reads: 'A community cafe serving home cooked, locally sourced, organic food'. Below this, three paragraphs describe the cafe's environment, menu options, and a special brunch offer for the weekend.

## Roughwork



Use the space below for roughwork:

## Forms

(Tutorial from <http://www.htmlquick.com>)

HTML forms serve the purpose of collecting information, which is later sent back to the server. For proper operation, it's important that the form provided in HTML is paired with some server-side code, usually called "processing agent", that will be responsible for receiving and processing the information as the author sees fit. This processing may consist of, for example, saving the information received or sending it by e-mail.

A form is basically a container for controls. Each control in a form is thought to collect information input by users, in ways that go from text lines to file uploads, through options, dates, passwords and many more. Once users have filled the form with data, they can submit it back to the server in order to let the processing agent administrate the gathered information.

The following code shows the basic structure of a form, with its opening and closing tags wrapping a set of controls.

```
<form>  
  [Set of controls]  
</form>
```

But this model says nothing about how the form is going to be processed and where. Such information can be provided with attributes like: `action`, which indicates the location of the processing agent; `method` which determines the method used to pack form data before it's sent to the processing agent; and `target`, which indicates where the processing results will be displayed. Here we'll talk about the `action` attribute alone, leaving the other two to be analysed in the reference of form.

The following form has been declared with the URI of a processing agent in the `action` attribute. You can see the processing agent taking over in a new window when you submit the form.

```
<form action="../../../form-result.php" target="_blank">  
  <input type="submit" value="Submit the form">  
</form>
```

## Text Controls

Text input controls provide the means to collect textual data, like names, directions, phrases, messages, passwords, etc. In the following sections we'll analyse the two most used and basic text controls.

### Single-line Text Input

A single-line text input, which allows only one line of text to be entered, is one of the many controls that's declared with the input element. In this case, the input element should present the value "text" in the type attribute.

With this alone, the control is already visible, but a name is needed if there's an intention to gather the data entered by the user in this control. The value of the name attribute will identify, server-side, the user's input in the control. The following example shows a basic implementation of a text input. Additionally, we're enclosing the control and its label in a paragraph, as these two can be considered to conform a unit with an idea that separates it from the rest.

```
<form action="../../../form-result.php" target="_blank">
  <p>
    Enter your full name: <input type="text" name="fullname">
    <input type="submit" value="Send data">
  </p>
</form>
```

After sending the form's data you can see clearly in the information displayed by the processing agent, how the data is received server-side. There you can also see that the name declared for the control is associated to the data entered by the user.

### Multi-line Text Input

This type of control is very similar to the single-line text input but it has the particularity of allowing multiple lines of text to be entered. A multi-line text control is usually rendered as a box, tall enough to contain more than one line of text at the same time. This representation, usually provides a scrollbar mechanism to allow users to see all entered text, particularly when it's long enough to exceed the box boundaries.

A multi-line text input is inserted with the text area element. Like before, the name attribute provides a name for the control that will help processing agents to identify the data submitted by the user.

In the following example we're improving the previous form to allow both types of text strings (single-line and multi-line) to be entered. In the example proposed, each control adapts better to the type of data it's supposed to collect.

```
<form action="../../../form-result.php" target="_blank">
  <p>Enter your full name: <input type="text" name="fullname"></p>
  <p>
    Leave a message:<br>
    <textarea name="message"></textarea>
  </p>
  <input type="submit" value="Send data">
</form>
```

## Option Controls

These controls allow users to select one or more options from a list of predefined values. Option controls can be presented in different styles and with different mechanisms of interaction with the user, depending mainly on the element utilized. A list of options can be declared in three ways: with checkboxes, with radio buttons and with lists.

### Checkboxes

A checkbox is a particular type of option that can be checked or unchecked upon user interaction. This allows authors to collect data like preferences, acceptance of terms y conditions, categories, or any other subject that can be answered with "yes" or "no". One thing this control has in particular is that, even when it can be declared as part of a thematic group, each checkbox is independent from all other checkboxes in the form.

Checkboxes are represented by the input element, when it has the value "checkbox" in the type attribute. Here the value of the name attribute also plays a role, by identifying the option server-side. In the following example, a few checkboxes have been declared as part of a thematic group of options. Remember that this aggrupation is only made by theme and position; the selection of checkboxes continues to be independent.

```
<form action="../../../form-result.php" target="_blank">
  <p>
```

```
Select your interests:<br>
<input type="checkbox" name="movies"> Movies<br>
<input type="checkbox" name="sports"> Sports<br>
<input type="checkbox" name="videogames"> Videogames
</p>
<p><input type="submit" value="Send data"></p>
</form>
```

A couple of things can be noted in the previous example. The first one is the lack of association between the checkbox and the text that describes it or, in other words, the impossibility of activating the checkbox by clicking the associated text. This is something that can be easily remedied converting the text into a label for the control, subject we'll treat later in this tutorial.

The second one is about how checkbox data is received server-side. If you submit the form, you'll see that only the selected checkboxes are sent to the processing agent. Their value, that depends very much on the language used server-side, is irrelevant considering that the mere presence of the checkbox's data is indicating, alone, its activation state.

## Radio Buttons

While checkboxes are independent and can be declared on their own, radio buttons are options that need to be grouped in order to have a meaning. Only one option can be selected at a time. This means, among other things, that when you select one option, the previous selected option gets deselected.

A radio button is also declared with the input element, but with the value "radio" in the type attribute. Here things get a little different from what happened with checkboxes, because the value of the name attribute needs to be shared by all the options in the same group. In other words, this is the mechanism that needs to be used in order to create a group of radio buttons.

But then, where's the value that tells the processing agent what option in the group has been selected? The answer to this question is in the value attribute. As the purpose of this attribute is to identify options in a group, its value should be different for each option. In the following example a group of radio buttons has been declared to conform a group where only one option can be selected, something that absolutely makes sense in this context. For this purpose, all the buttons share the same name and have each a different value.



```
<form action="../../../form-result.php" target="_blank">
  <p>
    Income:<br>
    <input type="radio" name="income" value="lowerthan1000"> Lower than
    $1,000.00<br>
    <input type="radio" name="income" value="from1000to5000"> From
    $1,000.00 to $5,000.00<br>
    <input type="radio" name="income" value="higherthan5000"> Higher than
    $5,000.00
  </p>
  <p><input type="submit" value="Send data"></p>
</form>
```

## Lists

A list of options is a control that can resemble, concerning its mechanics, each of the two controls previously analysed, depending on the presence of the Boolean attribute `multiple`. This attribute changes radically the behaviour of a list, by making possible the selection of only one single option at a time or many.

The structure of a list is composed, mainly, by two elements: `select`, that acts as the container for the options; and `option`, that represents one of the many options the control may present.

When the `multiple` attribute is absent, a list control behaves like a radio button group, where only one option can be selected at a time. The next example reflects this behaviour, which fits perfectly with the purpose of the field.

```
<p>
  Gender:
  <select name="gender">
    <option>Male</option>
    <option>Female</option>
  </select>
</p>
<p><input type="submit" value="Send data"></p>
</form>
```

In the previous example, you can see that what's sent to the server is the content of the selected option. But authors can change this behaviour if they think it's necessary, by declaring the attribute `value` for the option. When this attribute is present, its value is sent to the processing agent instead of the content of the element.

Now, when the multiple attribute is present, the list behaves like a set of checkboxes, where not only one but many options can be selected at the same time. This configuration requires that a couple of square braces follow any value the author chooses for the name attribute. If this requirement isn't fulfilled, processing agents will receive only the first selected option. The next example shows a list of options that can be selected without restrictions. It also makes use of the value attribute in the options, to avoid using unnecessary long values during data processing, server-side.

```

<form action="../../../form-result.php" target="_blank">
  <p>
    Select categories:<br>
    <select multiple name="categories[]">
      <option value="art">Art and entertainment</option>
      <option value="tv">Television and movies</option>
      <option value="kids">Kids and teenagers</option>
      <option value="diy">Do it yourself</option>
    </select>
  </p>
  <p><input type="submit" value="Send data"></p>
</form>

```

## Buttons

A button is a special type of control that's been designed to interact with the user in a singular way: an action is executed every time the user presses it. There's a wide range of buttons, each having some peculiarities in relation to its capabilities or behaviour, but here we'll only analyse the two most widely used in basic forms.

### Submit buttons

A submit button has the predefined action of submitting the form when activated. Unless other mechanisms for form submission are provided, the presence of this button is necessary if there's an intention to allow users to submit the form.

Submit buttons are inserted with the input element, having the value "submit" in its type attribute. The value attribute is important in this control, as its value is displayed as a label inside the button. The following example shows a form with a text input and a submit button.

```

<form action="../../../form-result.php" target="_blank">
  <p>
    Edit your description:

```

```
<input type="text" name="desc">
<input type="submit" value="Save edits">
</p>
</form>
```

## Reset buttons

Like submit buttons, reset buttons also have a predefined action. But in this case, the predefined action consists in the reset of the form fields to their initial values. In other words, the state of the fields in a form that's been reset is the same as when the page has loaded. This action removes all changes the user has applied to the values of the controls. It would be good to note here that all controls may have a predefined value, this is, a value that's present in the form when the page loads. The way authors have to specify this default value depends on the control. To know how to specify a default value in a particular control type, check the reference for the control in this list.

In the following example you'll be able to test the functionality of the reset button.

This form has been declared with a single-line text input, a couple of radio buttons and a checkbox. All these controls have a default value specified with the attributes value and checked.

```
<form action="../../../form-result.php" target="_blank">
  <p>Send message: <input type="text" name="message" value="I'm
ready!"></p>
  <p>
    <input type="radio" name="when" value="today" checked> Today<br>
    <input type="radio" name="when" value="tomorrow"> Tomorrow
  </p>
  <p><input type="checkbox" name="copy" checked> Send me a copy</p>
  <p>
    <input type="reset" value="Reset the form">
    <input type="submit" value="Send message">
  </p>
</form>
```

## Labelling Controls

Almost any control in a form can be labelled. Labelling controls is a worthwhile operation that enhances accessibility on many fronts. This association between a piece of text and a control will solve the problem noted in previous examples of this tutorial, particularly with radio buttons and checkboxes.

A label can be assigned with the label element. The easiest of the two existing methods for assigning a label to a control, consists of declaring both, the text and the control, as content of the label element. The next example has a couple of controls associated to labels with this method. There you can see how a control receives the focus when its label is clicked.

```
<form action="../../../form-result.php" target="_blank">
  <p><label>Name: <input type="text" name="fullname"></label></p>
  <p>
    Gender:
    <label><input type="radio" name="gender" value="male"> Male</label>
    <label><input type="radio" name="gender" value="female"> Female</label>
  </p>
  <p><label><input type="checkbox" name="newsletter"> I'd like to receive
the newsletter</label></p>
  <p><input type="submit" value="Send data"></p>
</form>
```

## Grouping controls

Sometimes, when a form is large, segmentation might play a role in the improvement of organisation and ease of use. This is why HTML provides the fieldset element, which is a container for controls. With this element, authors can make divisions to the form and organize controls thematically.

A fieldset can also have a title to identify the composition or purpose of the set of controls it contains. This title can be provided with the legend element, which must be declared as the first child of the fieldset. The following example shows a small form divided into two thematic groups.

```
<form action="../../../form-result.php" target="_blank">
  <fieldset>
    <legend>Personal information</legend>
    <p><label>Name: <input type="text" name="fullname"></label></p>
    <p><label>Address: <input type="text" name="address"></label></p>
  </fieldset>
  <fieldset>
    <legend>Preferences</legend>
    <p>
      <label><input type="checkbox" name="arts"> Arts</label><br>
      <label><input type="checkbox" name="television">
Television</label><br>
      <label><input type="checkbox" name="videogames">
Videogames</label><br>
      <label><input type="checkbox" name="sports"> Sports</label><br>
    </p>
  </fieldset>
  <input type="submit" value="Send data">
```

## Exercise 7 – Forms

1. Have a look at the sample files supplied.
2. Create a simple form for one of the pages from the previous Café website.
3. Add in a new link (which links to the form) in the menu at the top of the page.

## Roughwork



Use the space below for roughwork:



## Breakout Exercises

Sourced from



### Breakout A - Monster Mark-up

#### Getting Started

You will redesign a jumbled webpage into something more clear and informative for your readers! Download all the files from the folder supplied.

##### 1. Add a title with h1

Let's wrap the first "line" of the webpage with h1 tags. h1 stands for heading, level 1, which is like a top level title.

Try to add the tags like this:

```
<h1>Wanted: Creature Collectors</h1>
```

If it works, you should have a title on its own line on your webpage because h1 is a block element.

##### 2. Make a paragraph

Next, let's use the <p> tag to create a paragraph. Paragraphs are block elements, so if we do this right, the paragraph should be on its own line, so to speak, with space above and below it.

Try something like this:

`<p>Do you have what it takes to track and capture harmless monsters all around town? Are you ready to collect and train an army of adorable minions? When the time comes, will you be victorious in the fight to raise the cutest monster imaginable?</p>`

### 3. **Put the image on its own line**

Now we'll try to put the `<img>` - or image - on its own line. Images can be treated like block elements or inline elements depending on whether or not you wrap inline text around them. Let's try to give the image its own space with one paragraph above it and one below it. Hit return to put the `<img>` tag on its own line like this:

```

```

If you want to use a different picture, find one online with a search engine like Creative Commons that will find images you are free to use. Once you have the URL, or web address, of your new image, replace the `src` - or `source` - of the original image with your new address.

The `alt` attribute here stands for "alternative text." That text will show up if the image or link to it is ever broken and it's what the computer reads out loud to people who use screen-readers to help them access the web if they have difficulty seeing a page. Always include `alt` text for your images to improve the accessibility of your webpage!

### 4. **Make a paragraph followed by a list**

For our next step, we'll make a list. We'll build the list using the stats shared on the webpage. To make a list in HTML you can use an ordered list tag - `<ol>` - with list items - or `<li>` - that show up as numbers. You can also use an unordered list tag - `<ul>` - to make a list with bullet points instead of numbers. Each list and list item is its own block element.

Try to start a new paragraph followed by a list like this:

```
<p>Play Creature Collector: Totes Adorbs Edition today! Features</p>
```

- `<ul>`
- `<li>345 equally adorable monsters</li>`

- `<li>722 ability & equipment slots</li>`
- `<li>921 trainable skills</li>`
- `<li>12 difficult levels</li>`
- `<li>Countless virtual collection spots and battleground around the world</li>`
- `</ul>`

##### 5. **Create a closing paragraph & check out the link**

Finally, wrap the rest of the text in its own closing paragraph. You can also break up the text into several paragraphs if you want to chunk the information further.

Aim for something like this:

```
<p>Everything is accessible through your smartphone, watch, tablet, or 3D glasses! Available wherever Creature Collector: The Endbeginningning is sold. Not compatible with Creature Collector: Travel Edition. Requires Creature Collector: Starter Pack 3 to play. Visit <a href="https://creaturecollector.netorgcom">creaturecollector.netorgcom</a> for exclusive downloadable content.</p>
```

Check out the `<a>` which is the tag for a link. You can see that links are inline - they stay within the line of text that holds the link. You can try to change the href - or "hypertext reference" - of the link to one of your favourite websites to share with your fellow learners.

##### 6. **Add some style with inline tags**

Go back through your paragraphs and add some style using the tags for bold, italics, and underline. Try to use each tag at least once to highlight parts of the page that you want to emphasize.

- You put `<strong></strong>` around words you want to make bold.
- You put `<em></em>` around words you want to put in italics.
- You put `<u></u>` around words you want to underline.

Congratulations on revising this jumbled webpage into something more clear and informative for your readers! Look at the way you used block elements to chunk



information and how you used inline elements to add links and highlight key information. Keep these tags and their uses in mind as you begin working on your own webpages!

## Breakout B - Mark up a formal letter

For this project, your task is to mark up a letter that needs to be hosted on a university intranet. The letter is a response from a research fellow to a prospective PhD student concerning their application to work at the university

- To get started, you will need two files - the CSS you need to include in your HTML and the raw text for the letter.
- Create a new .html file using your text editor to do your work in
- You don't need to know any CSS to do this assessment; you just need to put the provided CSS inside an HTML element.
- Use the W3C HTML validator to validate your HTML - <https://validator.w3.org/>

### Block/structural semantics:

- You should structure the overall document with an appropriate structure including doctype, and <html>, <head> and <body> elements.
- The letter in general should be marked up with a structure of paragraphs and headings, with the exception of the below points. There is one heading (the "Re:" line) and three second level headings.
- The semester start dates, study subjects and exotic dances should be marked up using an appropriate list type.
- The two addresses can just be put inside paragraphs. The <address> element is not appropriate for them — think about why. In addition, each line of the addresses should sit on a new line, but not be in a new paragraph.

### Inline semantics:

- The names of the sender and receiver (and "Tel" and "Email") should be marked up with strong importance.

- The four dates in the document should be given appropriate elements containing machine-readable dates.
- The first address and first date in the letter should be given a class attribute value of "sender-column"; the CSS you'll add later will then cause these to be right aligned, as should be the case in a classic letter layout.
- The five acronyms/abbreviations in the main text of the letter should be marked up to provide expansions of each acronym/abbreviation.
- The six sub/superscripts should be marked up appropriately — in the chemical formulae, and the numbers 10<sup>3</sup> and 10<sup>4</sup> (they should be 10 to the power of 3 and 4, respectively).
- Try to mark up at least two appropriate words in the text with strong importance/emphasis.
- There are two places where a hyperlink should be added; add appropriate links with titles. For the location that the links point to, just use <http://example.com>.
- The university motto quote and citation should be marked up with appropriate elements.

## The head of the document:

- The character set of the document should be specified as utf-8 using an appropriate meta tag.
- The author of the letter should be specified in an appropriate meta tag.
- The provided CSS should be included inside an appropriate tag.

### TEACHER TIP!

Use the W3C HTML validator to validate your HTML - <https://validator.w3.org>

**Dr. Eleanor Gaye**  
Awesome Science faculty  
University of Awesome  
Bobtown, CA 99999,  
USA  
Tel: 123-456-7890  
Email: no\_reply@example.com

20 January 2016

**Miss Eileen Dover**  
4321 Cliff Top Edge  
Dover, CT9 XXXX  
UK

### Re: Eileen Dover university application

Dear Eileen,

Thank you for your recent application to join us at the University of Awesome's science faculty to study as part of your PhD next year. I will answer your questions one by one, in the following sections.

#### Starting dates

We are happy to accommodate you starting your study with us at any time, however it would suit us better if you could start at the beginning of a semester; the start dates for each one are as follows:

- First semester: 9 September 2016
- Second semester: 15 January 2017
- Third semester: 2 May 2017

Please let me know if this is ok, and if so which start date you would prefer.

You can find more information about [important university dates](#) on our website.

#### Subjects of study

At the Awesome Science Faculty, we have a pretty open-minded research facility — as long as the subjects fall somewhere in the realm of science and technology. You seem like an intelligent, dedicated researcher, and just the kind of person we'd like to have on our team. Saying that, of the ideas you submitted we were most intrigued by are as follows, in order of priority:

1. Turning H<sub>2</sub>O into wine, and the health benefits of Resveratrol (C<sub>14</sub>H<sub>12</sub>O<sub>3</sub>.)
2. Measuring the effect on performance of funk bassplayers at temperatures exceeding 30°C (86°F), when the audience size exponentially increases (effect of  $3 \times 10^3 > 3 \times 10^4$ .)
3. [HTML](#) and [CSS](#) constructs for representing musical scores.

So please can you provide more information on each of these subjects, including how long you'd expect the research to take, required staff and other resources, and anything else you think we'd need to know? Thanks.

#### Exotic dance moves

Yes, you are right! As part of my post-doctorate work, I *did* study exotic tribal dances. To answer your question, my favourite dances are as follows, with definitions:

##### Polynesian chicken dance

A little known but *very* influential dance dating back as far as 300BC, a whole village would dance around in a circle like chickens, to encourage their livestock or be "fruitful".

##### Icelandic brownian shuffle

Before the Icelanders developed fire as a means of getting warm, they used to practice this dance, which involved huddling close together in a circle on the floor, and shuffling their bodies around in imperceptibly tiny, very rapid movements. One of my fellow students used to say that he thought this dance inspired modern styles such as Twerking.

##### Arctic robot dance

An interesting example of historic misinformation, English explorers in the 1960s believed to have discovered a new dance style characterised by "robotic", stilted movements, being practiced by inhabitants of Northern Alaska and Canada. Later on however it was discovered that they were just moving like this because they were really cold.

For more of my research, see my [exotic dance research page](#).

Yours sincerely,

Dr Eleanor Gaye

University of Awesome motto: "Be excellent to each other." -- *Bill S Preston, Esq*

### Breakout Exercise II - Mark up a formal letter

## Breakout C

For this breakout, your task is to take the content for the homepage of a bird watching website and add structural elements to it so it can have a page layout applied to it. It needs to have:

- A header spanning the full width of the site containing the main title for the page, the site logo, and the navigation menu. The title and logo appear side by side once styling is applied, and the navigation appears below those two items.
- A main content area containing two columns - a main block to contain the welcome text, and a sidebar to contain image thumbnails.
- A footer containing copyright information and credits.

You need to add a suitable wrapper for:

- The header
- The navigation menu
- The main content
- The welcome text
- The image sidebar
- The footer

You should also:

Apply the provided CSS to the page by adding another `<link>` element just below the existing one provided at the start.

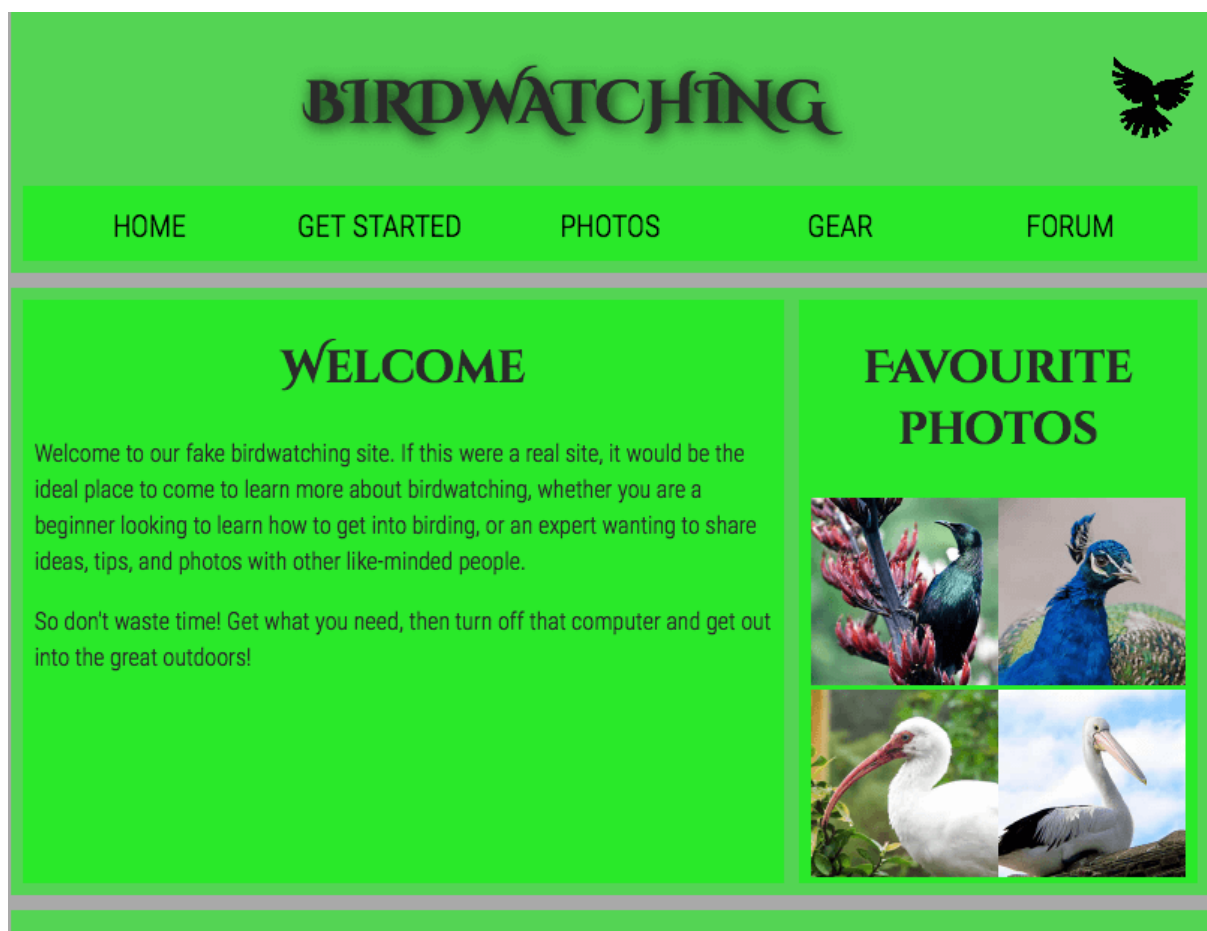
## Hints and tips

- You don't need to know any CSS to do this assessment; you just need to put the provided CSS inside an HTML element.
- The provided CSS is designed so that when the correct structural elements are added to the mark-up, they will appear green in the rendered page.

- If you are getting stuck and can't envisage what elements to put where, it often helps to draw out a simple block diagram of the page layout, and write on the elements you think should wrap each block.

## Example

The following screenshot shows an example of what the homepage might look like after being marked up.

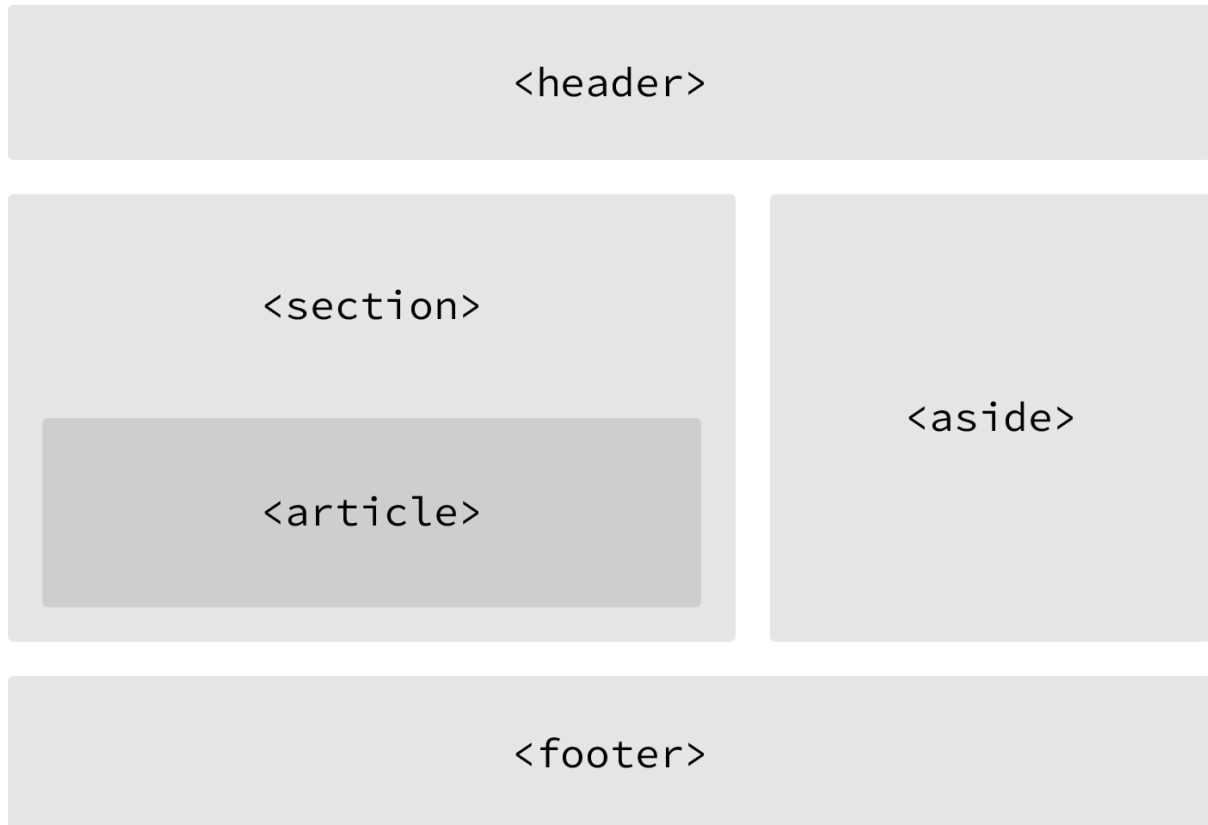


## Note

HTML5 introduced new structurally based elements, including the `<header>`, `<nav>`, `<article>`, `<section>`, `<aside>`, and `<footer>` elements.

All of these new elements are intended to give meaning to the organisation of our pages and improve our structural semantics. They are all block-level elements and do not have any

implied position or style. Additionally, all of these elements may be used multiple times per page, so long as each use reflects the proper semantic meaning.



## Block vs. Inline Elements

Most elements are either block- or inline-level elements. What's the difference?

Block-level elements begin on a new line, stacking one on top of the other, and occupy any available width.

Block-level elements may be nested inside one another and may wrap inline-level elements. We'll most commonly see block-level elements used for larger pieces of content, such as paragraphs.

Inline-level elements do not begin on a new line. They fall into the normal flow of a document, lining up one after the other, and only maintain the width of their content. Inline-level elements may be nested inside one another; however, they cannot wrap block-level elements. We'll usually see inline-level elements with smaller pieces of content, such as a few words.

## HTML Glossary

**HTML:** Hypertext Markup Language, the language of the web, the skeleton of a webpage.

**Element:** part of a webpage, like a paragraph or image.

**Tag:** an HTML label that identifies an element on a webpage, like `<p>` identifies a paragraph.

**Opening Tag:** the tag that begins part of a webpage, like `<p>` before a paragraph.

**Closing Tag:** the tag that ends part of a webpage with a slash, like `</p>` to end a paragraph.

**Attribute:** a characteristic of an element on a webpage, like `background-color` or `color`.

**Value:** the value for an attribute, like `“red”` for `“color”` or `“12px”` for `“font-size”`.

## HTML Resources

### **Glitch**

[www.Glitch.com](http://www.Glitch.com)

### **CodePen**

<https://codepen.io>

### **W3 Schools - HTML**

<https://www.w3schools.com/html>

### **Mozilla Development - HTML**

[https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction\\_to\\_HTML](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML)

### **HTML Cheat Sheet - A Simple Guide to HTML**

<http://www.simplehtmlguide.com/cheatsheet.php>

### **Beginner's Guide To HTML**

<https://www.beginnersguidetohtml.com>

### **HTML Crash Course For Absolute Beginners - YouTube**

<https://www.youtube.com/watch?v=UB1O30fR-EE>

### **Get HTML Color Codes**

<https://htmlcolorcodes.com>

### **HTML Tags Ordered Alphabetically**

<https://www.w3schools.com/tags>

### **HTML CheatSheet**

<https://htmlcheatsheet.com/>

### **HTML Templates**

<https://html5-templates.com/>

### **HTML Quick Tutorials**

<http://www.htmlquick.com>

*Exercises & Breakouts sourced from W3 Schools, Mozilla Development & HTML Quick.*



# Section 2

## Cascading Style Sheets

## Contents

<b>Introduction to CSS</b>	<b>68</b>
<ul style="list-style-type: none"><li>• What is CSS</li><li>• History</li><li>• Background</li><li>• CSS example I</li><li>• Reflection exercise</li></ul>	
<b>CSS Structure</b>	<b>72</b>
<ul style="list-style-type: none"><li>• Structure</li><li>• How does CSS affect HTML?</li><li>• How does CSS work?</li><li>• How to apply your CSS to your HTML</li><li>• CSS Example II</li><li>• CSS syntax</li><li>• CSS declarations</li><li>• Beyond syntax: make CSS readable</li><li>• Reflection exercise</li></ul>	
<b>Selectors</b>	<b>82</b>
<ul style="list-style-type: none"><li>• Introduction to selectors</li><li>• Different types of selectors</li><li>• Simple selectors</li><li>• Pseudo-classes and pseudo-elements</li></ul>	
<b>CSS Units</b>	<b>86</b>
<ul style="list-style-type: none"><li>• CSS units</li><li>• Reflection exercise</li></ul>	
<b>Cascade</b>	<b>89</b>
<b>The Box Model</b>	<b>90</b>
<ul style="list-style-type: none"><li>• The Box model</li></ul>	
<b>Styling Text</b>	<b>92</b>
<ul style="list-style-type: none"><li>• Font properties - style, font weight</li><li>• Text properties - colour, text-align, vertical-align</li><li>• Text layout - alignment and line height</li><li>• Reflection exercise</li></ul>	

## Styling boxes

98

- CSS outline
- Styling boxes overview
- Outline colour
- Outline width
- Outline - shorthand property
- Outline offset

## CSS layout

103

- Overview of CSS layout
- Normal flow
- Display property
- Flexbox
- Grid layout

## Responsive Web Design

108

- Introduction
- The best experience for all users

## Breakout Exercises

110

- Box model
- Business card
- Typesetting a school homepage
- Creating fancy letter headed paper
- A fancy box
- Webpage layout

## CSS Glossary

124

## CSS Resources

125

## Introduction to CSS

### What is CSS?

CSS (Cascading Style Sheets) is a language that describes the style of an HTML document. CSS describes how HTML elements should be displayed.

- CSS stands for Cascading Style Sheets.
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media.
- CSS saves a lot of work. It can control the layout of multiple web pages all at once.
- External stylesheets are stored in CSS files.

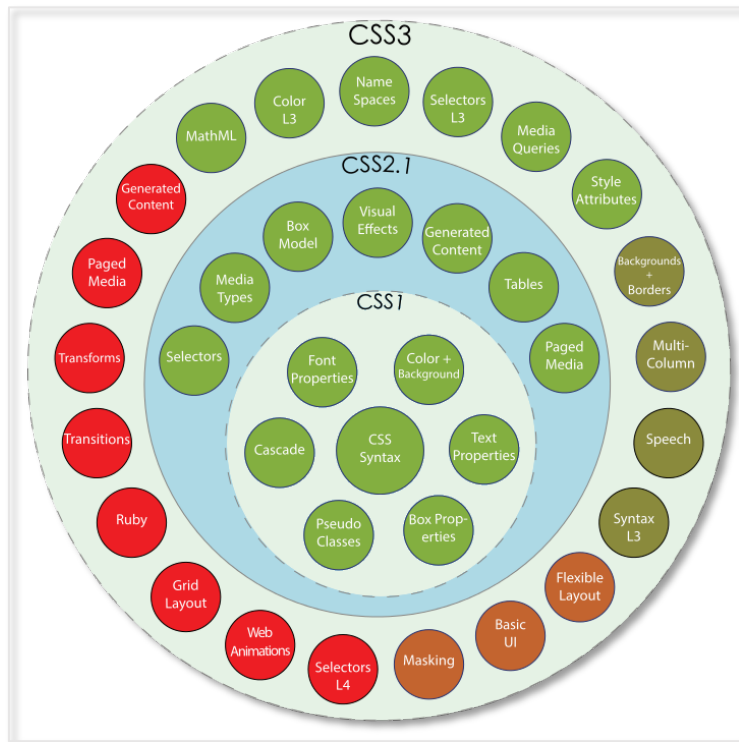


### History

CSS was first developed in 1997 as a way for web developers to define the visual appearance of the web pages that they were creating. It was intended to allow web professionals to separate the content and structure of a website's code from the visual design, something that had not been possible prior to this time.

The separation of structure and style allows HTML to perform more of its original function, the markup of content, without having to worry about the design and layout of the page itself, something commonly known as the "look and feel" of the page.

CSS didn't gain in popularity until around 2000 when web browsers began using more than the basic font and colour aspects of this markup language. As CSS continues to evolve and new styles are introduced, web browsers have begun to implement modules that bring new CSS support into those browsers and give web designers powerful new styling tools to work with.



The Schematic of the evolution of CSS from 1 to 3

<https://goo.gl/2cwT1I>

## Background

HTML was NEVER intended to contain tags for formatting a web page. HTML was created to describe the content of a web page, like:

```
<h1>This is a heading</h1>
```

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

When tags like <font>, and colour attributes were added to the HTML 3.2 specification, it created huge problems for web developers. Development of large websites, where fonts and colour information were added to every single page, became a long and expensive process. To solve this problem, the World Wide Web Consortium (W3C) created CSS. CSS removed the style formatting from the HTML page.

The style definitions are normally saved in external .css files. With an external stylesheet file, you can change the look of an entire website by changing just one file!

## CSS Example

The CSS Zen Garden is a World Wide Web development resource "built to demonstrate what can be accomplished visually through CSS-based design." Style sheets contributed by designers from around the world are used to change the visual presentation of a single HTML file, producing hundreds of different designs.

Explore the CSS Zen Garden example files which are supplied - HTML and CSS file. Visit the CSS zen website to view different designs. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. A demonstration of what can be accomplished through CSS-based design.



<http://www.csszengarden.com/>

## Reflection

Reflect on what you have learned about CSS so far.



Use the space below to write **five** things about CSS.

1

---

2

---

3

---

4

---

5

---

## CSS Structure

### Structure

CSS is a language for specifying how documents are presented to users - how they are styled, laid out, etc.

A document is usually a text file structured using a markup language - HTML is the most common markup language, but you will also come across other markup languages such as SVG or XML.

Presenting a document to a user means converting it into a usable form for your audience. Browsers, like Firefox, Chrome or Internet Explorer, are designed to present documents visually, for example, on a computer screen, projector or printer.

### How does CSS affect HTML?

Web browsers apply CSS rules to a document to affect how they are displayed. A CSS rule is formed from:

- A set of properties, which have values set to update how the HTML content is displayed, for example, the element's width is 50% of its parent element, and its background to be red.
- A selector, which selects the element(s) you want to apply the updated property values to. For example, the application a CSS rule to all the paragraphs in a HTML document.

A set of CSS rules contained within a stylesheet determines how a webpage should look.



## CSS Example II

Open the two files for this exercise in the folder called Structure. The first rule starts with an h1 selector, which means that it will apply its property values to the <h1> element. It contains three properties and their values.

1. The first one sets the text colour to blue.
2. The second sets the background colour to yellow.
3. The third one puts a border around the header that is 1 pixel wide, solid (not dotted, or dashed, etc.), and coloured black.

The second rule starts with a p selector, which means that it will apply its property values to the <p> element. It contains one declaration, which sets the text colour to red.

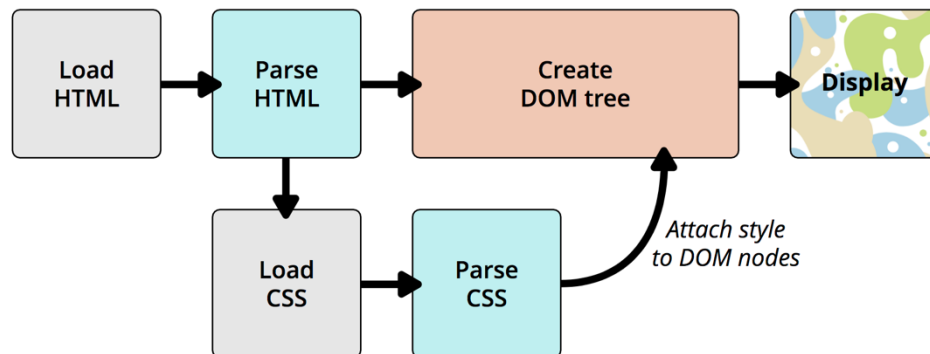
In a web browser, the code above would produce the following output:



## How does CSS work?

When a browser displays a document, it must combine the document's content with its style information. It processes the document in two stages:

1. The browser converts HTML and CSS into the DOM (*Document Object Model*). The DOM represents the document in the computer's memory. It combines the document's content with its style.
2. The browser displays the contents of the DOM.



## How to apply your CSS to your HTML

There are three different ways to apply CSS to an HTML document that you'll commonly come across, some more useful than others. There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

### External stylesheet

You've already seen external stylesheets previously, but not by that name. An external stylesheet is when you have your CSS written in a separate file with a .css extension, and you reference it from an HTML <link> element. The HTML file looks something like this:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="utf-8">
5    <title>My CSS experiment</title>
6    <link rel="stylesheet" href="style.css">
7  </head>
8  <body>
9    <h1>Hello World!</h1>
10   <p>This is my first CSS example</p>
11 </body>
12 </html>
13
  
```

And the CSS file:

```
1 h1 {
2   color: blue;
3   background-color: yellow;
4   border: 1px solid black;
5 }
6
7 p {
8   color: red;
9 }
10
```

This method is arguably the best, as you can use one stylesheet to style multiple documents.

### Internal stylesheet

An internal stylesheet is where you don't have an external CSS file, but instead place your CSS inside a `<style>` element, contained inside the HTML head. So the HTML would look like this:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>My CSS experiment</title>
6     <style>
7       h1 {
8         color: blue;
9         background-color: yellow;
10        border: 1px solid black;
11      }
12
13      p {
14        color: red;
15      }
16    </style>
17  </head>
18  <body>
19    <h1>Hello World!</h1>
20    <p>This is my first CSS example</p>
21  </body>
22 </html>
23
24
```

This can be useful in some circumstances (maybe you're working with a content management system where you can't modify the CSS files directly), but it isn't quite as

efficient as external stylesheets - in a website, the CSS would need to be repeated across every page.

## Inline styles

Inline styles are CSS declarations that affect one element only, contained within a style attribute:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>My CSS experiment</title>
6 </head>
7 <body>
8   <h1 style="color: blue;background-color: yellow;border:
9     1px solid black;">Hello World!</h1>
10  <p style="color:red;">This is my first CSS example</p>
11 </body>
12 </html>
13
14
```

The only time you might have to resort to using inline styles is when your working environment is really restrictive (perhaps your CMS only allows you to edit the HTML body).

## CSS syntax

At its most basic level, CSS consists of two building blocks:

**Properties:** Human-readable identifiers that indicate which stylistic features (e.g. font, width, background colour) you want to change.

**Values:** Each specified property is given a value, which indicates how you want to change those stylistic features (e.g. the font, width or background colour).

A property paired with a value is called a *CSS declaration*. CSS declarations are put within *CSS Declaration Blocks*. CSS declaration blocks are paired with *selectors* to produce *CSS Rulesets* (or *CSS Rules*).

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>My CSS experiment</title>
6     <link rel="stylesheet" href="style.css">
7   </head>
8   <body>
9     <h1>Hello World!</h1>
10    <p>This is my first CSS example</p>
11
12    <ul>
13      <li>This is</li>
14      <li>a list</li>
15    </ul>
16  </body>
17 </html>
18
19
```

And the CSS file:

```
1  h1 {
2    colour: blue;
3    background-color: yellow;
4    border: 1px solid black;
5  }
6
7  p {
8    color: red;
9  }
10
11  p, li {
12    text-decoration: underline;
13  }
14
15
```

**Hello World!**

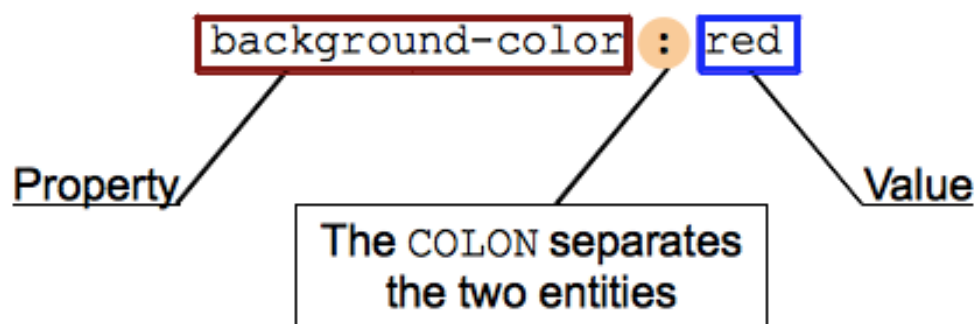
This is my first CSS example

- This is
- a list

## CSS declarations

Setting CSS properties to specific values is the core function of the CSS language. The CSS engine calculates which declarations apply to every single element of a page in order to appropriately lay it out and style it. The property and value in each pair is separated by a colon (:). There are more than 300 different properties in CSS and nearly an infinite number of different values.

A CSS declaration :



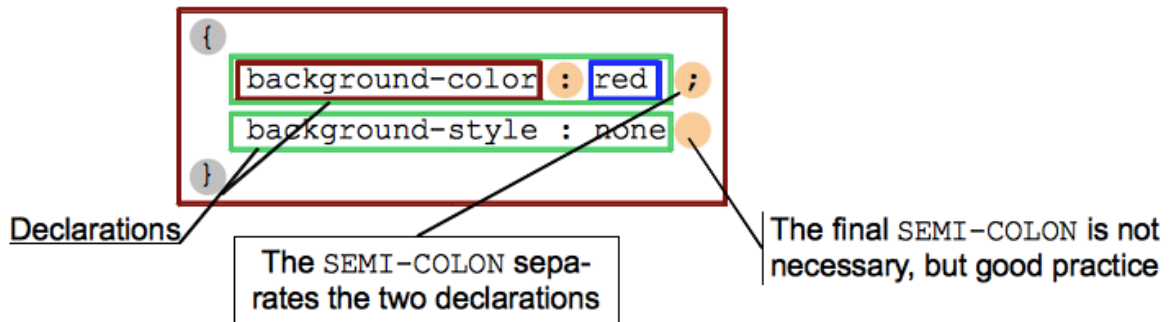
### CSS declaration blocks

Declarations are grouped in blocks, with each set of declarations being wrapped by opening curly brackets { and a closing one }.

Each declaration contained inside a declaration block has to be separated by a semi-colon ; otherwise the code won't work (or will at least give unexpected results). The last

declaration of a block doesn't need to be terminated by a semi-colon, though it is often considered *good style* to do so.

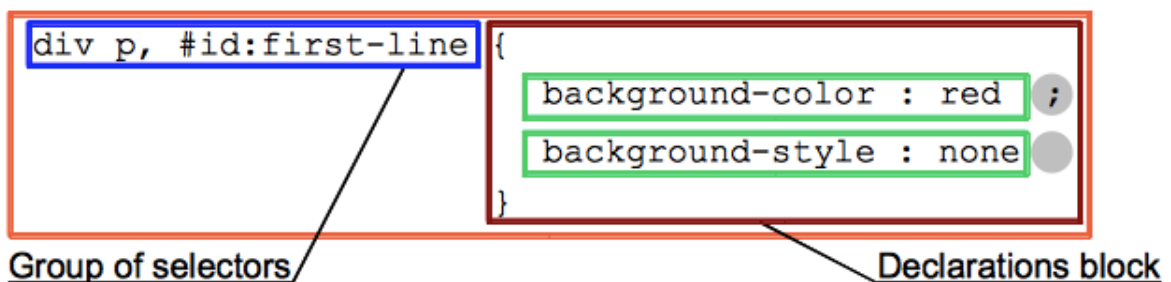
A CSS declarations block:



### CSS selectors and rules

We are missing one part of the puzzle - we need to discuss how to tell our declaration blocks which elements they should be applied to. This is done by prefixing each declaration block with a selector — a pattern that matches some elements on the page. The associated declarations will be applied to those elements only. The selector plus the declaration block is called a ruleset or a rule.

A CSS ruleset (or rule):



## Beyond syntax: make CSS readable

There are some good tips worth knowing to make your CSS code easier to use and maintain.

### White space

White space means actual spaces, tabs and new lines. You can add white space to make your stylesheets more readable. In the same manner as HTML, the browser tends to ignore much of the whitespace inside your CSS; a lot of the whitespace is just there to aid readability.

### Comments

As with HTML, you are encouraged to make comments in your CSS, to help you understand how your code works when coming back to it after several months, and to help others understand it. Comments are also useful for temporarily *commenting out* certain parts of the code for testing purposes, for example if you are trying to find which part of your code is causing an error. Comments in CSS begin with `/*` and end with `*/`.



## Reflection

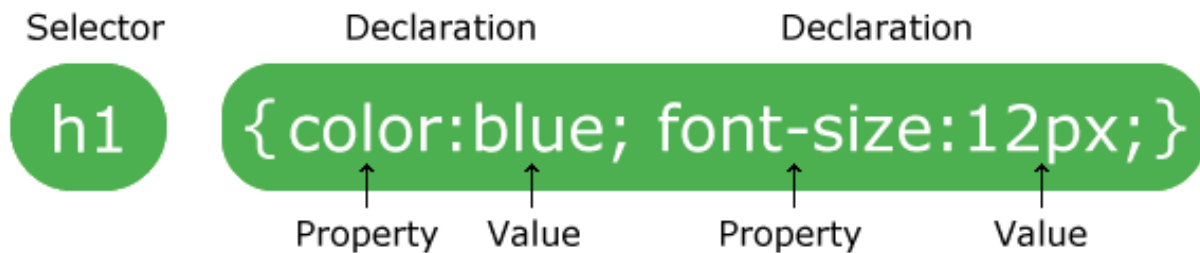
Reflect on what you have learned about how CSS works.



Use the space below

## Introduction to Selectors

To recap, selectors are one part of a CSS rule and come just before CSS declaration blocks.



## Different types of selectors

Selectors can be divided into the following categories:

- Simple selectors: Match one or more elements based on element type, class, or id.
- Attribute selectors: Match one or more elements based on their attributes/attribute values.
- Pseudo-classes: Match one or more elements that exist in a certain state, such as an element that is being hovered over by the mouse pointer, or a checkbox that is currently disabled or checked, or an element that is the first child of its parent in the DOM tree.
- Pseudo-elements: Match one or more parts of content that are in a certain position in relation to an element, for example the first word of each paragraph, or generated content appearing just before an element.
- Combinators: These are not exactly selectors themselves, but ways of combining two or more selectors in useful ways for very specific selections. So for example, you could select only paragraphs that are direct descendants of divs, or paragraphs that come directly after headings.

- Multiple selectors: Again, these are not separate selectors; the idea is that you can put multiple selectors on the same CSS rule, separated by commas, to apply a single set of declarations to all the elements selected by those selectors.

## Simple selectors

### Type selectors /element selectors

This selector is just a case-insensitive match between the selector name and a given HTML element name. This is the simplest way to target all elements of a given type. Let's take a look at an example:



*View the example above (HTML & CSS) in folder supplied.*

### Class selectors

The class selector consists of a dot, '.', followed by a class name. A class name is any value, without spaces, placed within an HTML class attribute. It is up to you to choose a name for the class. It is also noteworthy that multiple elements in a document can have the same class value, and a single element can have multiple class names separated by white space.



*View the example above (HTML & CSS) in folder supplied.*

### ID selectors

The ID selector consists of a hash/pound symbol (#), followed by the ID name of a given element. Any element can have a unique ID name set with the id attribute. It is up to you to choose an ID name. It's the most efficient way to select a single element.



*View the example above (HTML & CSS) in folder supplied.*



CSS Selectors Video - <https://youtu.be/viJJoo8uJuY>

## Pseudo-classes and pseudo-elements

### Pseudo-classes

A CSS pseudo-class is a keyword added to the end of a selector, preceded by a colon (:), which is used to specify that you want to style the selected element but only when it is in a certain state. For example, you might want to style a link element only when it is being hovered over by the mouse pointer, or a checkbox when it is disabled or checked. For example:

- :active
- :checked
- :default
- :dir
- :disabled
- :empty
- :enabled

- :first
- :first-child

We will look into every pseudo-class right now but here is a simple example.

## [Mozilla Developer Network](#)





*View the example files (HTML & CSS) in folder supplied.*

### **Pseudo-elements**

Pseudo-elements are very much like pseudo-classes, but they have differences. They are keywords, this time preceded by two colons :: , that can be added to the end of selectors to select a certain part of an element.

- ::after
- ::before
- ::first-letter
- ::first-line
- ::selection
- ::backdrop

They all have some very specific behaviours and interesting features.

- [CSS](#)  defined in the MDN glossary.
- [HTML](#)  defined in the MDN glossary.



*View the example above (HTML & CSS) in folder supplied.*

## CSS Units

- CSS has several different units for expressing a length.
- Many CSS properties take "length" values, such as width, margin, padding, font-size, etc.
- Length is a number followed by a length unit, such as 10px, 2em, etc.
- A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted.
- For some CSS properties, negative lengths are allowed.
- There are two types of length units: absolute and relative.

### Absolute Lengths

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

Unit	Description
cm	centimetres
mm	millimetres
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)

pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

## Relative Lengths

Relative length units specify a length relative to another length property. Relative length units scales better between different rendering mediums.

Unit	Description
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
ex	Relative to the x-height of the current font (rarely used)
ch	Relative to width of the "0" (zero)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport*
vh	Relative to 1% of the height of the viewport*
vmin	Relative to 1% of viewport's* smaller dimension
vmax	Relative to 1% of viewport's* larger dimension
%	Relative to the parent element

## Reflection

Reflect on what you have learned about CSS units.



Use the space below

A large, empty rounded rectangular box intended for writing reflections.



## Cascade

At some point in your work, you'll find yourself in the situation where multiple CSS rules will have selectors matching the same element. In such cases, which CSS rule "wins", and ends up being the one that is finally applied to the element? This is controlled by a mechanism called the *Cascade*; this is also related to inheritance (elements will take some property values from their parents, but not others).

### The Cascade

CSS is an abbreviation for *Cascading Style Sheets*, which indicates that the notion of the cascade is important. At its most basic level, it indicates that the order of CSS rules matter, but it's more complex than that. What selectors win out in the cascade depends on three factors (these are listed in order of weight - earlier ones will overrule later ones):

- Importance
- Specificity
- Source order

### Importance

In CSS, there is a special piece of syntax you can use to make sure that a certain declaration will always win over all others: *!important*.

#### TEACHER TIP

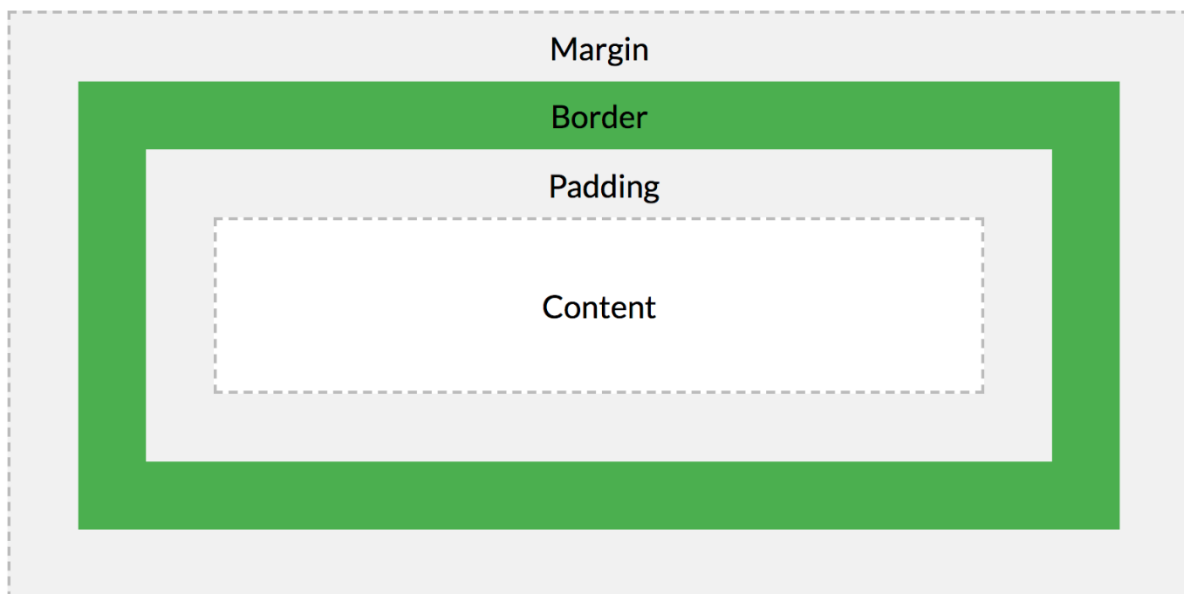
To learn more on Cascade, please see visit <https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade>

## The Box Model

The CSS box model is the foundation of layout on the Web - each element is represented as a rectangular box, with the box's content, padding, border, and margin built up around one another like the layers of an onion. As a browser renders the web page layout, it works out what styles are applied to the content of each box, how big the surrounding onion layers are, and where the boxes sit in relation to one another. Before understanding how to create CSS layouts, you need to understand the box model.

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:



Explanation of the different parts:

- **Content** - The content of the box, where text and images appear.
- **Padding** - Clears an area around the content. The padding is transparent.
- **Border** - A border that goes around the padding and content.
- **Margin** - Clears an area outside the border. The margin is transparent.

The box model allows us to add a border around elements, and to define space between elements.

**TEACHER TIP**

The default behaviour of browsers when calculating the width of an element is to apply the calculated width and height to the content area, without taking any of the padding, border and margin in consideration. This approach has proven to be quite complicated to work with. You can change this behaviour by setting the box-sizing property.

**TEACHER TIP**

Use the W3C CSS validator to validate your CSS - <https://jigsaw.w3.org/css-validator>

## Styling Text

As you'll have already experienced in your work with HTML and CSS, text inside an element is laid out inside the element's content box. It starts at the top left of the content area, and flows towards the end of the line. Once it reaches the end, it goes down to the next line and continues, then the next line, until all the content has been placed in the box. Text content effectively behaves like a series of inline elements, being laid out on lines adjacent to one another, and not creating line breaks until the end of the line is reached, or unless you force a line break manually using the `<br>` element.

The CSS properties used to style text generally fall into two categories, which we'll look at separately:

**Font styles:** Properties that affect the font that is applied to the text, affecting what font is applied, how big it is, whether it is bold, italic, etc.

**Text layout styles:** Properties that affect the spacing and other layout features of the text, allowing manipulation of, for example, the space between lines and letters, and how the text is aligned within the content box.

## Font Properties

The CSS font properties define the font family, weight, size, and the style of a text.

### The font-family Property

The font-family property specifies the font for an element. The font-family property can hold several font names as a second choice. If the browser does not support the first font, it tries the next font.

There are two types of font family names:

- **family-name** - The name of a font-family, like "times", "courier", "arial", etc.
- **generic-family** - The name of a generic-family, like "serif", "sans-serif", "cursive", "fantasy", "monospace".

Start with the font you want, and always end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

#### TEACHER TIP

Separate each value with a comma.

If a font name contains white-space, it must be quoted. Single quotes must be used when using the "style" attribute in HTML.

### Font style, font weight, text transform, and text decoration

CSS provides four common properties to alter the visual weight/emphasis of text:

- **font-style:** Used to turn italic text on and off. Possible values are as follows (you'll rarely use this, unless you want to turn some italic styling off for some reason):
  - normal: Sets the text to the normal font (turns existing italics off.)
  - italic: Sets the text to use the *italic version of the font* if available; if not available, it will simulate italics with oblique instead.
  - oblique: Sets the text to use a simulated version of an italic font, created by *slanting the normal version*.
- **font-weight:** Sets how bold the text is. This has many values available in case you have many font variants available (such as *-light*, *-normal*, *-bold*, *-extrabold*, *-black*, etc.), but realistically you'll rarely use any of them except for normal and bold:
  - normal, bold: Normal and **bold** font weight
  - lighter, bolder: Sets the current element's boldness to be one step lighter or heavier than its parent element's boldness.
  - 100–900: Numeric boldness values that provide finer grained control than the above keywords, if needed.
- **text-transform:** Allows you to set your font to be transformed. Values include:
  - none: Prevents any transformation.
  - uppercase: Transforms ALL TEXT TO CAPITALS.
  - lowercase: Transforms all text to lower case.
  - capitalise: Transforms all words to Have The First Letter Capitalised.

- full-width: Transforms all glyphs to be written inside a fixed-width square, similar to a monospace font, allowing aligning of e.g. Latin characters along with Asian language glyphs (like Chinese, Japanese, Korean.)
  
- **text-decoration:** Sets/unsets text decorations on fonts (you'll mainly use this to unset the default underline on links when styling them.) Available values are:
  - none: unsets any text decorations already present.
  - underline: Underlines the text.
  - overline: Gives the text an overline.
  - line-through: Puts a strikethrough over the text.
  
- **The font-size Property:** The font-size property sets the size of a font.

## Text Properties

### The color Property

The colour property specifies the colour of text. Use a background colour combined with a text colour that makes the text easy to read.

### The text-align Property

The text-align property specifies the horizontal alignment of text in an element.

### The vertical-align Property

The vertical-align property sets the vertical alignment of an element.

Value	Description
<b>baseline</b>	The element is aligned with the baseline of the parent. This is default
<b>length</b>	Raises or lowers an element by the specified length. Negative values are allowed.
<b>%</b>	Raises or lowers an element in a percent of the "line-height" property. Negative values are allowed
<b>sub</b>	The element is aligned with the subscript baseline of the parent
<b>super</b>	The element is aligned with the superscript baseline of the parent
<b>top</b>	The element is aligned with the top of the tallest element on the line
<b>text-top</b>	The element is aligned with the top of the parent element's font
<b>middle</b>	The element is placed in the middle of the parent element
<b>bottom</b>	The element is aligned with the lowest element on the line
<b>text-bottom</b>	The element is aligned with the bottom of the parent element's font

<b>initial</b>	<b>Sets this property to its default value</b>
<b>inherit</b>	<b>Inherits this property from its parent element</b>

## Text layout

With basic font properties out the way, let's now have a look at properties we can use to affect text layout.

### Text alignment

The text-align property is used to control how text is aligned within its containing content box. The available values are as follows, and work in pretty much the same way as they do in a regular word processor application:

- left: Left justifies the text.
- right: Right justifies the text.
- center: Centers the text.
- justify: Makes the text spread out, varying the gaps in between the words so that all lines of text are the same width. You need to use this carefully — it can look terrible, especially when applied to a paragraph with lots of long words in it. If you are going to use this, you should also think about using something else along with it, such as hyphens, to break some of the longer words across lines.

### Line height

The line-height property sets the height of each line of text — this can take most length and size units, but can also take a unitless value, which acts as a multiplier and is generally considered the best option — the font-size is multiplied to get the line-height. Body text generally looks nicer and is easier to read when the lines are spaced apart; the recommended line height is around 1.5–2 (double spaced.) So to set our lines of text to 1.5 times the height of the font, you'd use this:



## Reflection

Reflect on what you have learned about CSS so far.



Use the space below to write **five** things about styling text:

1.

---

2.

---

3.

---

4.

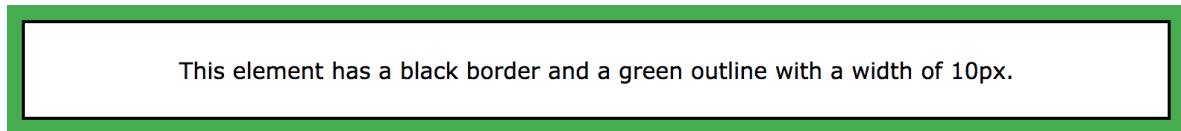
---

5.

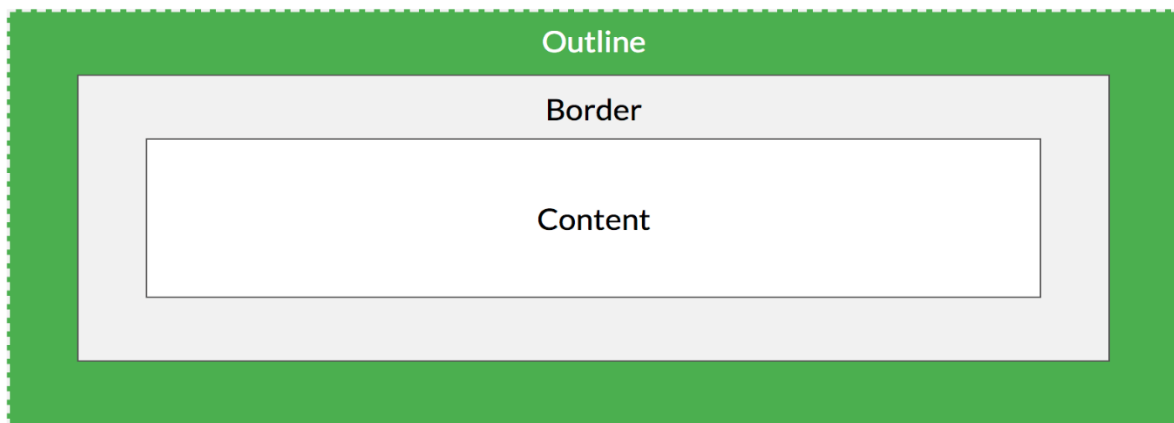
---

## Styling boxes

### CSS Outline



An outline is a line that is drawn around elements, **OUTSIDE** the borders, to make the element "stand out".



### Styling boxes overview

CSS has the following outline properties:

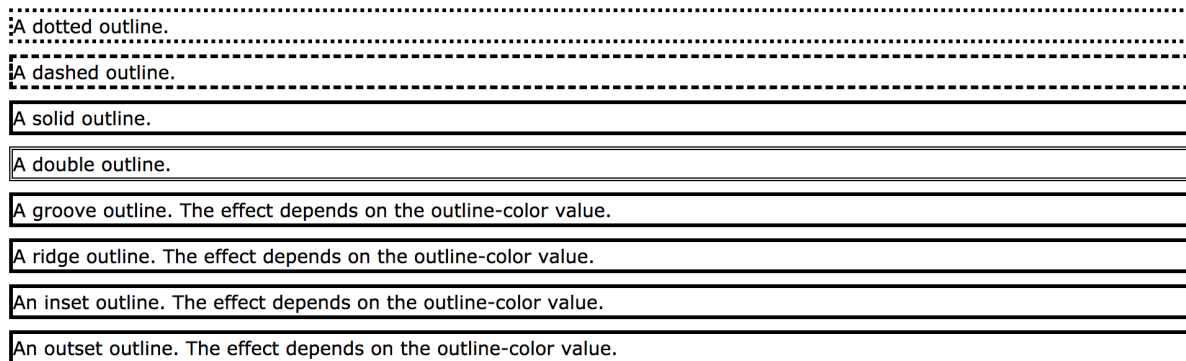
- outline-style
- outline-color
- outline-width
- outline-offset
- outline

## Outline Style

The outline-style property specifies the style of the outline, and can have one of the following values:

- dotted - Defines a dotted outline
- dashed - Defines a dashed outline
- solid - Defines a solid outline
- double - Defines a double outline
- groove - Defines a 3D grooved outline
- ridge - Defines a 3D ridged outline
- inset - Defines a 3D inset outline
- outset - Defines a 3D outset outline
- none - Defines no outline
- hidden - Defines a hidden outline

The following example shows the different outline-style values:



*Open the files above (HTML & CSS) in folder supplied.*

## Outline Colour

The outline-colour property is used to set the colour of the outline.

The colour can be set by:

- name - specify a colour name, like "red"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"
- invert - performs a colour inversion (which ensures that the outline is visible, regardless of colour background)

The following example shows some different outlines with different colours. Also notice that these elements also have a thin black border inside the outline:



*Open the files above (HTML & CSS) in folder supplied.*

## Outline Width

The outline-width property specifies the width of the outline, and can have one of the following values:

- thin (typically 1px)
- medium (typically 3px)
- thick (typically 5px)
- A specific size (in px, pt, cm, em, etc)

The following example shows some outlines with different widths:



*Open the files above (HTML & CSS) in folder supplied.*

## Outline - Shorthand property

The outline property is a shorthand property for setting the following individual outline properties:

- outline-width
- outline-style (required)
- outline-colour

The outline property is specified as one, two, or three values from the list above. The order of the values does not matter.

The following example shows some outlines specified with the shorthand outline property:

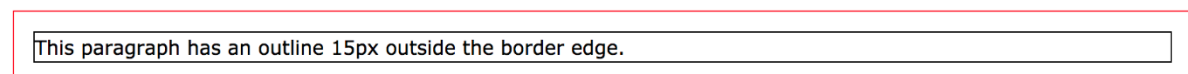


*Open the files above (HTML & CSS) in folder supplied.*

## Outline Offset

The outline-offset property adds space between an outline and the edge/border of an element. The space between an element and its outline is transparent.

The following example specifies an outline 15px outside the border edge:



*Open the files above (HTML & CSS) in folder supplied.*

## CSS layout

### CSS layout overview

CSS page layout techniques allow us to take elements contained in a web page and control where they are positioned relative to their default position in normal layout flow, the other elements around them, their parent container, or the main viewport/window.

Each technique has its uses, advantages, and disadvantages, and no technique is designed to be used in isolation. By understanding what each method is designed for you will be in a good place to understand which is the best layout tool for each task.

### Normal Flow

Normal flow is how the browser lays out HTML pages by default when you do nothing to control page layout.

```
1~ <html>
2~ <head>
3
4~ <title>I love my cat</title>
5  </head>
6
7~ <body>
8~   <p>I love my cat.</p>
9
10~ <ul>
11~   <li>Buy cat food</li>
12~   <li>Exercise</li>
13~   <li>Cheer up friend</li>
14  </ul>
15
16~ <p>The end!</p>
17  </body>
18  </html>
19
20
```

I love my cat.

- Buy cat food
- Exercise
- Cheer up friend

The end!

Note here how the HTML is displayed in the same order in which it appears in the source code, with elements stacked up on top of one another — the first paragraph, followed by the unordered list, followed by the second paragraph.

The elements that appear one below the other are described as block elements, in contrast to inline elements, which appear one beside the other, like the individual words in a paragraph.

When you use CSS to create a layout, you are moving the elements away from the normal flow, but for many of the elements on your page the normal flow will create exactly the layout you need. This is why starting with a well-structured HTML document is so important, as you can then work with the way things are laid out by default rather than fighting against it.

The methods that can change how elements are laid out in CSS are as follows:

- **The *display property*** - Standard values such as block, inline or inline-block can change how elements behave in normal flow. We then have entire layout methods that are switched on via a value of display, for example CSS Grid and Flexbox.
- **Floats** - Applying a float value such as left can cause block level elements to wrap alongside one side of an element, like the way images sometimes have text floating around them in magazine layouts.
- **The *position property*** - Allows you to precisely control the placement of boxes inside other boxes. static positioning is the default in normal flow, but you can cause elements to be laid out differently using other values.



- **Table layout** - features designed for styling the parts of an HTML table can be used on non-table elements using `display: table` and associated properties.
- **Multi-column layout** - The multi-column layout properties can cause the content of a block to layout in columns, as you might see in a newspaper.

## The display property

This property allows us to change the default way something displays. Everything in normal flow has a value of `display`, used as the default way that elements they are set on behave. For example, the fact that paragraphs in English display one below the other is due to the fact that they are styled with `display: block`. If you create a link around some text inside a paragraph, that link remains inline with the rest of the text, and doesn't break onto a new line. This is because the `<a>` element is `display: inline` by default.

You can change this default display behaviour. For example, the `<li>` element is `display: block` by default, meaning that list items display one below the other in our English document. If we change the `display` value to `inline` they now display next to each other, as words would do in a sentence. The fact that you can change the value of `display` for any element means that you can pick HTML elements for their semantic meaning, without being concerned about how they will look. The way they look is something that you can change. In addition to being able to change the default presentation by turning an item from `block` to `inline` and vice versa, there are some bigger layout methods that start out as a value of `display`. However when using these you will generally need to invoke additional properties. The two values most important for our purposes when discussing layout are `display: flex` and `display: grid`.

## Flexbox

Flexbox is the short name for the Flexible Box Layout Module, designed to make it easy for us to lay things out in one dimension - either as a row or as a column. To use flexbox, you apply `display: flex` to the parent element of the elements you want to lay out; all its direct children then become flex items.

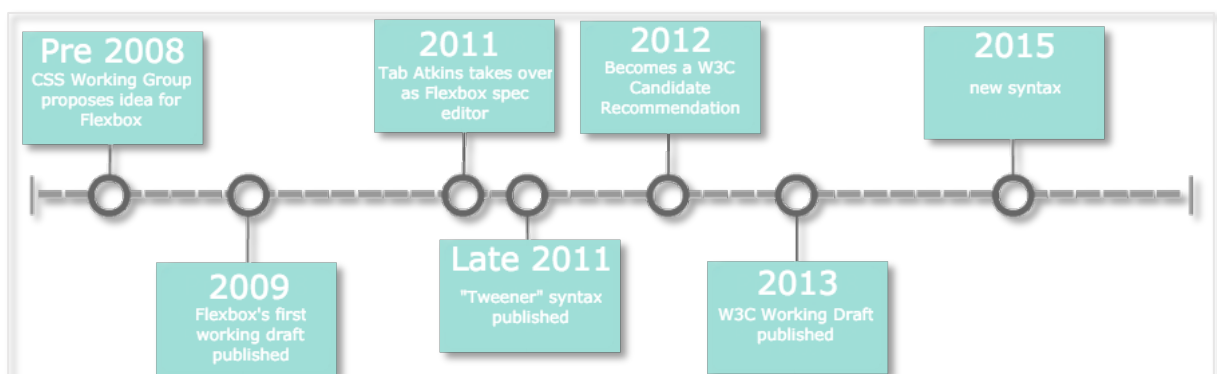
The HTML markup in the example below gives us a containing element, with a class of `wrapper`, inside which are three `<div>` elements. By default these would display as block elements, below one another, in our English language document.

However, if we add `display: flex` to the parent, the three items now arrange themselves into columns. This is due to them becoming *flex items* and using some initial values that flexbox gives them. They are displayed as a row, because the initial value of `flex-direction` is `row`. They all appear to stretch to the height of the tallest item, because the initial value of the `align-items` property is `stretch`. This means that the items stretch to the height of the flex container, which in this case is defined by the tallest item. The items all line up at the start of the container, leaving any extra space at the end of the row.



*Open the Flexbox sample files above (HTML & CSS) in folder supplied.*

In addition to the above properties that can be applied to the flex container, there are properties that can be applied to the flex items. These properties, among other things, can change the way that the items flex, enabling them to expand and contract to fit into the available space.



## Grid Layout

While flexbox is designed for one-dimensional layout, Grid Layout is designed for two dimensions — lining things up in rows and columns.

Once again, you can switch on Grid Layout with a specific value of display — display: grid. The below example uses similar markup to the flex example, with a container and some child elements. In addition to using display: grid, we are also defining some row and column tracks on the parent using the grid-template-rows and grid-template-columns properties respectively. We've defined three columns each of 1fr and two rows of 100px. I don't need to put any rules on the child elements; they are automatically placed into the cells our grid has created.



*Open the Grid 1 example above (HTML & CSS) in folder supplied.*

Once you have a grid, you can explicitly place your items on it, rather than relying on the auto-placement behaviour seen above. In the second example below we have defined the same grid, but this time with three child items. We've set the start and end line of each item using the grid-column and grid-row properties. This causes the items to span multiple tracks.



*Open the Grid 2 example above (HTML & CSS) in folder supplied.*

There are other layout methods, which are less important for the main layout structures of your page but can still help you achieve specific tasks. These include floats and positioning.

### TEACHER TIP

Use the GridbyExample website to learn more about using grids with CSS - <https://gridbyexample.com/examples/page-layout/#layout1>

## Responsive Web Design

### Introduction

- Responsive web design makes your web page look good on all devices.
- Responsive web design uses only HTML and CSS.
- Responsive web design is not a program or a JavaScript.

### The Best Experience For All Users

Websites can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:



**Desktop**



**Tablet**



**Phone**

It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

**TEACHER TIP**

To learn more on Responsive Web Design including details on Viewpoint, Grid View, Media Queries, Image, Videos, Frameworks and to try some templates out please see [https://www.w3schools.com/css/css\\_rwd\\_viewport.asp](https://www.w3schools.com/css/css_rwd_viewport.asp)



## Breakout Exercises

Sourced from



### Breakout A - Box Model

In the editable sample below, we have a set of three boxes, all of which contain text content and have been styled to span the whole of the body width. They are represented by `<header>`, `<main>`, and `<footer>` elements in the markup. We'd like you to concentrate on the bottom three CSS rules — the ones that target each box individually — and try the following:

1. Have a look at the box model of each individual element on the page by opening up the browser developer tools and clicking on the elements in the DOM inspector.

- **Menu bar:**
  - **Firefox:** Menu ≡ ► *Web Developer* ► *Toggle Tools*, or *Tools* ► *Web Developer* ► *Toggle Tools*
  - **Chrome:** *More tools* ► *Developer tools*
  - **Safari:** *Develop* ► *Show Web Inspector*. If you can't see the *Develop* menu, go to *Safari* ► *Preferences* ► *Advanced*, and check the *Show Develop menu in menu bar* checkbox.
  - **Opera:** *Developer* ► *Developer tools*

Each browser has a box model viewer that shows exactly what margin, border and padding is applied to each box, how big the content box is, and the total space the element takes up.

2. Set some `margin-bottom` on the `<main>` element, say 20px. Now set some `margin-top` on the `<footer>` element, say 15px. Note how the 2nd one of these actions

makes no difference to the layout — this shows margin collapsing in action; the smaller margin's effective width is reduced to 0, leaving only the larger margin.

3. Set a margin of 30px and a padding of 30px on every side of the `<main>` element — note how the space around the element (the margin) and the space between the border and the content (the padding) both increase, causing the actual content to take up a smaller amount of space. Again, check this with the browser developer tools.
4. Set a larger border on all sides of the `<main>` element, say 40px, and notice how this takes space away from the content rather than the margin or padding. You could do this by setting a complete new set of values for the width, style and color with the border property, e.g. `60px dashed red`, but since the properties are already set in a previous rule, you could just set a new border-width.
5. By default, the content width is set to 100% of the available space (after the margin, border, and padding have taken their share) — if you change the browser window width, the boxes will grow and shrink to stay contained inside the example output window. The height of the content will default to the height of the content inside it.
6. Try setting a new width and height on the `<main>` element - start with say 400px width and 200px height - and observe the effect. You'll notice that the width no longer changes as the browser window is resized.
7. Try setting a percentage width on the `<main>` element instead - say 60% width - and observe the effect. You should see that the width now changes again as the browser window is resized. Remove the `<main>` element's height setting for now.
8. Try setting your `<main>` element's padding and margin to be 5% on all sides, and observe the result. If you use your browser developer tools to look at the width of the example output window and compare that to the width of the margin/padding, you'll see that this 5% means "5% of the containing element's width." So as the size of the example output window increases, so does the padding/margins.

9. Margins can accept negative values, which can be used to cause element boxes to overlap. Try setting `margin-top: -50px;` on the `<main>` element to see the effect.

## Breakout B - Business Card Exercise

You have been provided with some raw HTML and an image, and need to write the necessary CSS to style this into a nifty little online business card, which can perhaps double as a gamer card or social media profile. The following sections describe what you need to do.

### Basic setup:

1. First of all, create a new file in the same directory as your HTML and image files. Call it something really imaginative like `style.css`.
2. Link your CSS to your HTML file via a `<link>` element.
3. The first two rulesets in the CSS resource file are yours, for FREE! After you've finished rejoicing at your good fortune, copy and paste them into the top of your new CSS file. Use them as a test to make sure your CSS is properly applied to your HTML.
4. Above the two rules, add a CSS comment with some text inside it to indicate that this is a set of general styles for the overall page. "General page styles" would do. Also add three more comments at the bottom of the CSS file to indicate styles specific to the setup of the card container, styles specific to the header and footer, and styles specific to the main business card contents. From now on, subsequent styles added to the stylesheet should be organized in an appropriate place.
5. Taking care of the selectors and rulesets provided in the CSS resource file.
6. Next up, we'd like you to look at the four selectors, and calculate the specificity for each one. Write these down somewhere where they can be found later on, such as in a comment at the top of your CSS.



7. Now it's time to put the right selector on the right rule set! You've got four pairs of selector and ruleset to match in your CSS resources. Do this now, and add them to your CSS file.

**You need to:**

8. Give the main card container a fixed width/height, solid background color, border, and border-radius (rounded corners!), amongst other things.
9. Give the header a background gradient that goes from darker to lighter, plus rounded corners that fit in with the rounded corners set on the main card container.
10. Give the footer a background gradient that goes from lighter to darker, plus rounded corners that fit in with the rounded corners set on the main card container.
11. Float the image to the right so that it sticks to the right hand side of the main business card contents, and give it a max-height of 100% (a clever trick that ensures that it will grow/shrink to stay the same height as its parent container, regardless of what height it becomes).
12. Beware! There are two errors in the provided rulesets. Using any technique that you know, track these down and fix them before moving on.
13. New rulesets you need to write:
  - a. Write a ruleset that targets both the card header, and card footer, giving them both a computed total height of 50px (including a content height of 30px and padding of 10px on all sides.) But express it in `ems`.
  - b. The default margin applied to the `<h2>` and `<p>` elements by the browser will interfere with our design, so write a rule that targets all these elements and sets their margin to 0.
  - c. To stop the image from spilling out of the main business card content (the `<article>` element), we need to give it a specific height. Set the `<article>`'s height to 120px, but expressed in `ems`. Also give it a

background colour of semi-transparent black, resulting in a slightly darker shade that lets the background red colour shine through a bit too.

- d. Write a ruleset that gives the `<h2>` an effective font size of 20px (but expressed in `ems`) and an appropriate line height to place it in the centre of the header's content box. Recall from earlier that the content box height should be 30px - this gives you all the numbers you need to calculate the line height.
- e. Write a ruleset that gives the `<p>` inside the footer an effective font size of 15px (but expressed in `ems`) and an appropriate line height to place it in the centre of the footer's content box. Recall from earlier that the content box height should be 30px — this gives you all the numbers you need to calculate the line height.
- f. As a last little touch, give the paragraph inside the `<article>` an appropriate padding value so that its left edge lines up with the `<h2>` and footer paragraph, and set its colour to be fairly light so it is easy to read.

## Roughwork.



Use the space below

## Breakout C - Typesetting a Community School Homepage

You have been provided with some raw HTML for the homepage of an imaginary School, plus some CSS that styles the page into a two column layout and provides some other rudimentary styling. You are to write your CSS additions below the comment at the bottom of the CSS file, to make sure it is easy to mark the bits you have done.

### Fonts:

- First of all, download a couple of free-to-use fonts. Because this is a School, the fonts should be chosen to give the page a fairly serious, formal, trustworthy feel — a serif site-wide font for the general text body, coupled with sans-serif or slab serif for the headings might be nice.
- Use a suitable service to generate bulletproof `@font-face` code for these two fonts.
- Apply your body font to the whole page, and your heading font to your headings.

### General text styling:

- Give the page a site-wide font-size of 10px.
- Give your headings and other element types appropriate font-sizes defined using a suitable relative unit.
- Give your body text a suitable line-height.
- Center your top level heading on the page.
- Give your headings a little bit of letter-spacing to make them not too too squashed, and allow the letters to breathe a bit.
- Give your body text some letter-spacing and word-spacing, as appropriate.
- Give the first paragraph after each heading in the `<section>` a little bit of text-indentation, say 20px.

### Links:

- Give the link, visited, focus, and hover states some colors that go with the color of the horizontal bars at the top and bottom of the page.
- Make it so that links are underlined by default, but when you hover or focus them, the underline disappears.
- Remove the default focus outline from ALL the links on the page.

- Give the active state a noticeably different styling so it stands out nicely, but make it still fit in with the overall page design.
- Make it so that external links have the external link icon inserted next to them.

### **Lists:**

- Make sure the spacing of your lists and list items works well with the styling of the overall page. Each list item should have the same `line-height` as a paragraph line, and each list should have the same spacing at its top and bottom as you have between paragraphs.
- Give your list items a nice bullet, appropriate for the design of the page. It is up to you whether you choose a custom bullet image or something else.

### **Navigation menu:**

- Style your navigation menu so that it has an appropriate look for the look and feel for the page.

### **Hints and tips**

- You don't need to edit the HTML in any way for this exercise.
- You don't necessarily have to make the nav menu look like buttons, but it needs to be a bit taller so that it doesn't look silly on the side of the page; also remember that you need to make this one a vertical nav menu.

The following screenshot shows an example of what the finished design could look like:

## St Huxley's Community College

---

### Brave new world

It's a brave new world out there. Our children are being put in increasing more competitive situations, both during recreation, and as they start to move into the adult world of [examinations](#), [jobs](#), [careers](#) and other life choices. Having the wrong mindset, becoming [too emotional](#), or making the wrong choices can contribute to them experiencing difficulty in taking their rightful place in today's ideal society.

As concerned parents, guardians or carers, you will no doubt want to give your children the best possible start in life – and you've come to the right place.

### The best start in life

At St. Huxley's, we pride ourselves in not only giving our students a top quality education, but also giving them the societal and emotional intelligence they need to win big in the coming utopia. We not only excel at subjects such as genetics, data mining, and chemistry, but we also include compulsory lessons in:

- Emotional control
- Judgement
- Assertion
- Focus and resolve

If you are interested, then your next steps will likely be to:

a. [Call us](#) for more information  
b. [Ask for a brochure](#), which includes sign-up form

### Top course choices

- [Genetic engineering](#)
- [Genetic mutation](#)
- [Organic Chemistry](#)
- [Pharmaceuticals](#)
- [Biochemistry with behaviour](#)
- [Pure biochemistry](#)
- [Data mining](#)
- [Computer security](#)
- [Bioinformatics](#)
- [Cybernetics](#)

[See more](#)

Home

Finding us

Courses

Staff

Media

Prospectus

## Roughwork.



Use the space below

## Breakout D - Creating fancy letter headed paper

You have been given the files needed to create a letter headed paper template. You just need to put the files together. To get there, you need to:

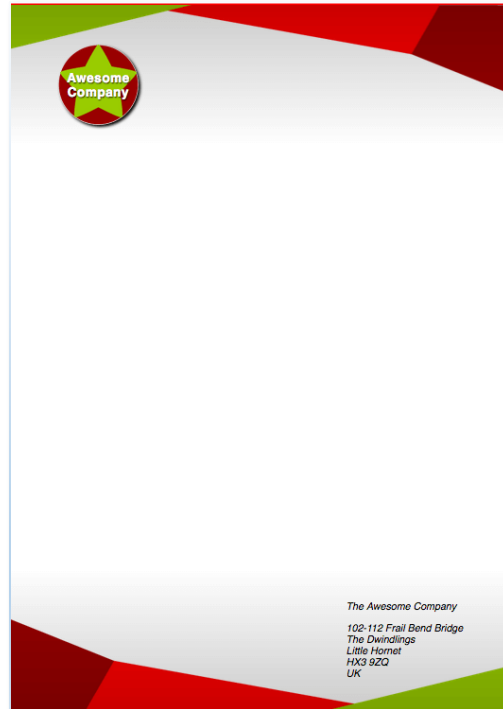
### The main letter

- Apply the CSS to the HTML.
- Add a background declaration to the letter that:
  - Fixes the top image to the top of the letter
  - Fixes the bottom image to the bottom of the letter
  - Adds a semi-transparent gradient over the top of both of the previous backgrounds that gives the letter a bit of texture. Make it slightly dark right near the top and bottom, but completely transparent for a large part of the center.
- Add another background declaration that just adds the top image to the top of the letter, as a fall-back for browsers that don't support the previous declaration.
- Add a white background colour to the letter.
- Add a 1mm top and bottom solid border to the letter, in a colour that is inkeeping with the rest of the colour scheme.

### The logo

- To the `<h1>`, add the logo as a background image.
- Add a filter to the logo to give it a subtle drop shadow.
- Now comment out the filter and implement the drop shadow in a different (slightly more cross-browser compatible) way, which still follows the shape of the round image.

The following screenshot shows an example of what the finished design could look like:



## Roughwork.



Use the space below

## Breakout E - A Fancy Box

Your task is to create fancy box and explore the fun we can have with CSS.

### General task

Apply the CSS to the HTML.

### Styling the box

We'd like you to style the provided `<p>`, giving it the following:

- A reasonable width for a large button, say around 200 pixels.
- A reasonable height for a large button, centering the text vertically in the process.
- A slight increase in font size, to around 17-18 pixel computed style. Use rems.  
Write a comment about how you worked out the value.
- A base colour for the design. Give the box this colour as its background colour.
- The same colour for the text; make it readable using a black text shadow.
- A fairly subtle border radius.
- A 1-pixel solid border with a colour similar to the base colour, but a slightly darker shade.
- A linear semi-transparent black gradient that goes toward the bottom right corner. Make it completely transparent at the start, gradating to around 0.2 opacity by 30% along, and remaining at the same colour until the end.
- Multiple box shadows. Give it one standard box shadow to make the box look slightly raised off the page. The other two should be inset box shadows — a semi-transparent white shadow near the top left and a semi-transparent black shadow near the bottom right — to add to the nice raised 3D look of the box.

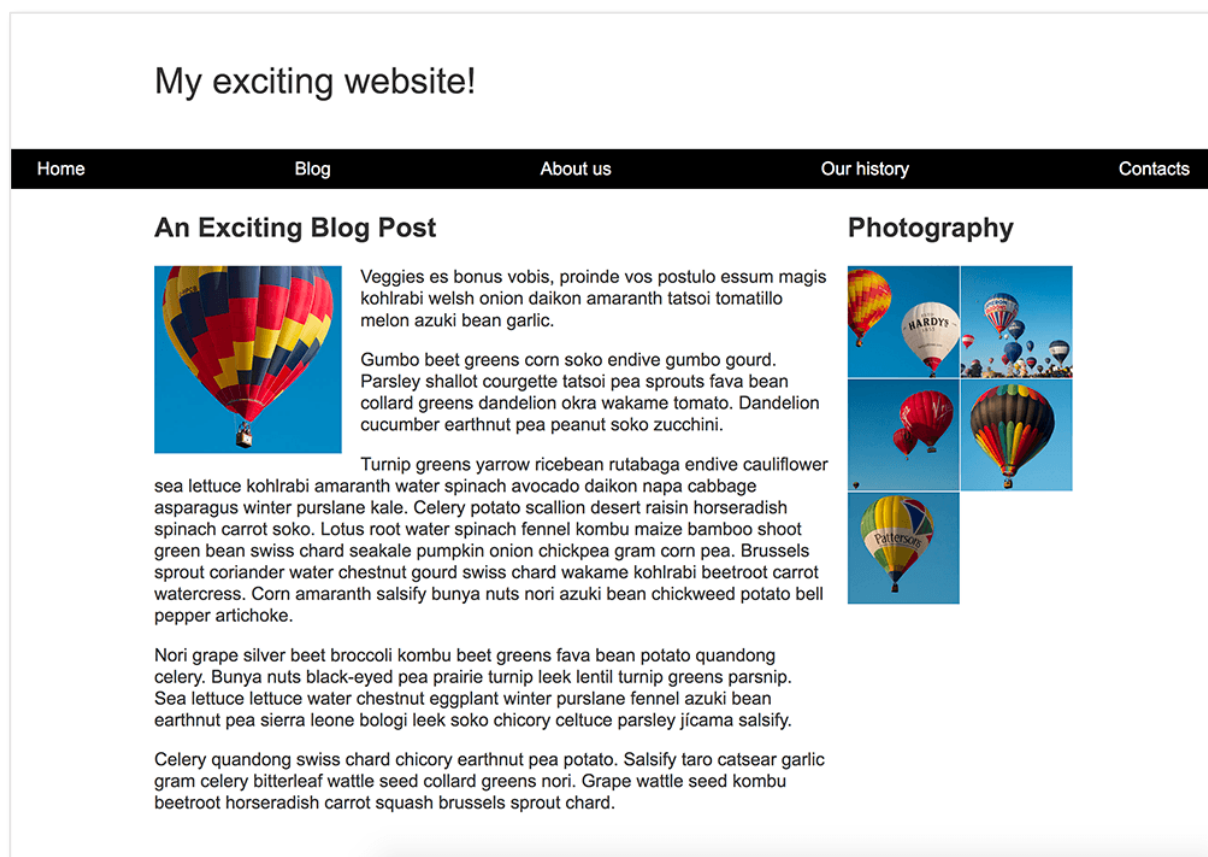
The following screenshot shows an example of what the finished design could look like:





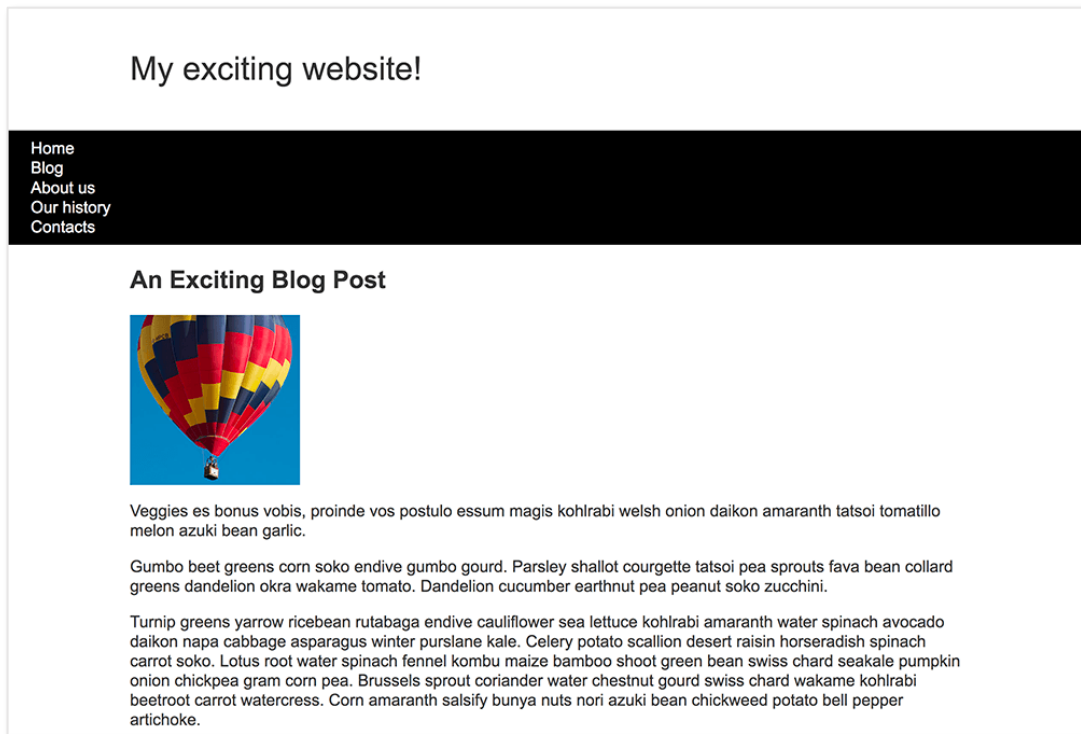
## Breakout F - Webpage Layout

You have been provided with some raw HTML, basic CSS, and images — now you need to create a layout for the design, which should look just like the image below.



## Basic Setup

- You can download all the required files in the folder supplied.
- Save the HTML document and stylesheet into a directory on your computer, and add the images into a folder named `images`. Opening the `index.html` file in a browser should give you a page with basic styling but no layout, which should look something like the image seen below.
- This starting point has all of the content of your layout as displayed by the browser in normal flow.



## Your tasks

You now need to implement your layout. The tasks you need to achieve are:

1. To display the navigation items in a row, with an equal amount of space.
2. The navigation bar should scroll with the content and then become stuck at the top of the viewport when it reaches it.
3. The image that is inside the article should have text wrapped around it.
4. The <article> and <aside> elements should display as a two column layout. The columns should be a flexible size so that if the browser window shrinks smaller the columns become narrower.
5. The photographs should display as a two column grid with a 1 pixel gap between the images.

You do not need to edit the HTML in order to achieve this layout and the techniques you should use are:

- Positioning
- Float
- Flexbox

- CSS Grid Layout

## Roughwork.



Use the space below.

## CSS Glossary

Please refer to W3schools A-Z guide with illustrated examples:



<https://www.w3schools.com/cssref/>

## CSS Resources

### Box Alignment Cheat sheet

<https://rachelandrew.co.uk/css/cheatsheets/box-alignment>

### Everything you need to learn CSS Grid Layout

<https://gridbyexample.com/>

### The Experimental Layout Lab

<https://labs.jensimmons.com/>

### Grid by Example – Rachel Andrew

[https://www.youtube.com/playlist?list=PLQkVA6z3dFvbnBJetfYDAF3-cG\\_ubgdZR](https://www.youtube.com/playlist?list=PLQkVA6z3dFvbnBJetfYDAF3-cG_ubgdZR)

### CodePen Tool

<https://codepen.io>

### CSS Exercises

[https://www.w3schools.com/css/exercise.asp?filename=exercise\\_margin3](https://www.w3schools.com/css/exercise.asp?filename=exercise_margin3)

### W3 Schools

<https://www.w3schools.com/>

### Mozilla Development

[https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/CSS\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics)

### Free Code Camp

<https://medium.freecodecamp.org/the-css-handbook-a-handly-guide-to-css-for-developers-b56695917d11>

### Listing the most useful free online tools and resources for web developers

<https://html-css-js.com/html/links/>

***Exercises & Breakouts sourced from W3 Schools, Mozilla Development, Free Code Camp.***

# Section 5

## UX Design

## Contents

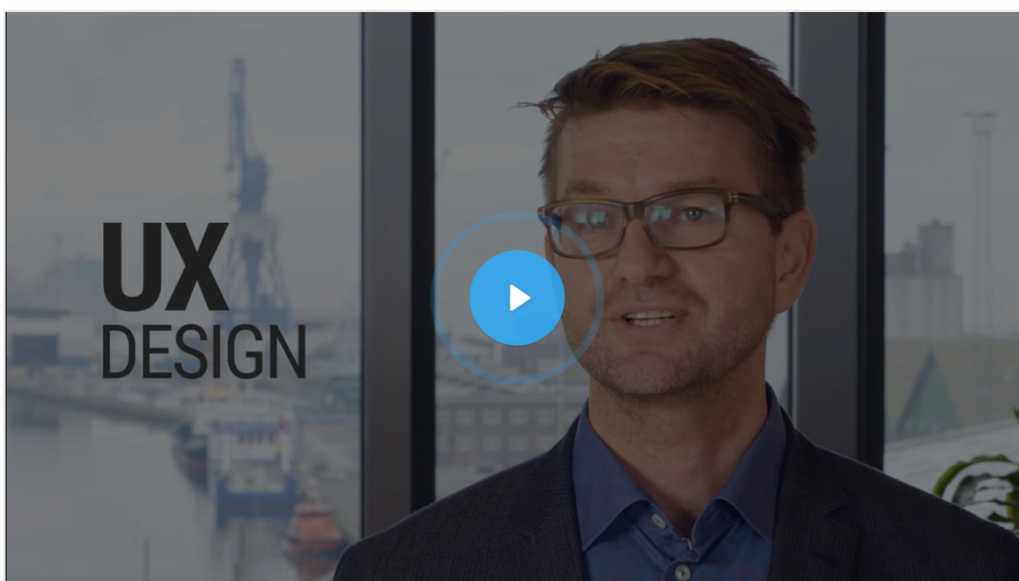
• <b>UX Design</b>	<b>128</b>
○ What is User Experience Design?	
○ UX versus UI design	
○ What UX designers consider	
○ UX design is user-centred	
○ The user experience process	
○ Don Norman	
○ Don Norman's Design Principles	
○ Don Norman on UX	
○ Jakob Nielsen	
○ Nielsen's 10 Usability Heuristics	
○ Shneiderman's Golden Rules	
○ Reflection Exercise	
• <b>Breakout Exercise</b>	<b>143</b>
• <b>UX Resources</b>	<b>144</b>

## UX Design



### What is User Experience (UX) Design?

According to the Interaction-Design.org - User experience (UX) design is the process of creating products that provide meaningful and relevant experiences to users. This involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability, and function.



<https://www.interaction-design.org/literature/topics/ux-design>



## UX Versus UI

“User Experience Design” is often used interchangeably with terms such as “User Interface Design” and “Usability”. However, while Usability and User Interface Design are important aspects of UX Design, they are subsets of it – UX design covers a vast array of other areas, too. A UX designer is concerned with the entire process of acquiring and integrating a product, including aspects of branding, design, usability and function. It is a story that begins before the device is even in the user’s hands.

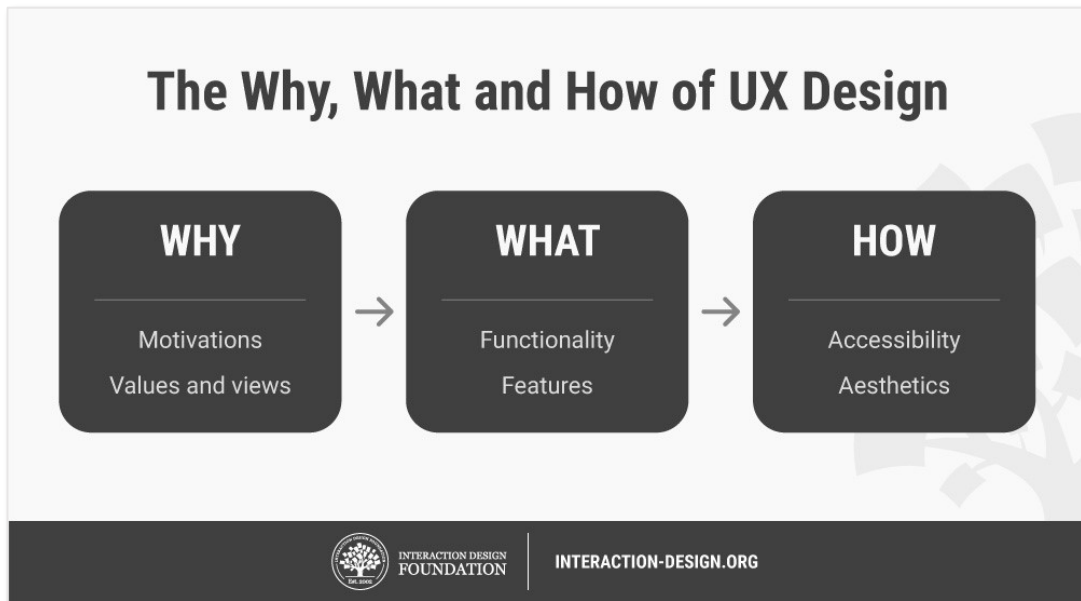
“No product is an island. A product is more than the product. It is a cohesive, integrated set of experiences. Think through all of the stages of a product or service – from initial intentions through final reflections, from first usage to help, service, and maintenance. Make them all work together seamlessly.”

- Don Norman, inventor of the term “User Experience”

Products that provide great user experience (e.g., the iPhone) are thus designed with not only the product’s consumption or use in mind but also the entire process of acquiring, owning, and even troubleshooting it. Similarly, UX designers don’t just focus on creating products that are usable; they concentrate on other aspects of the user experience, such as pleasure, efficiency and fun. Consequently, there is no single definition of a good user experience. Instead, a good user experience is one that meets a particular user’s needs in the specific context.

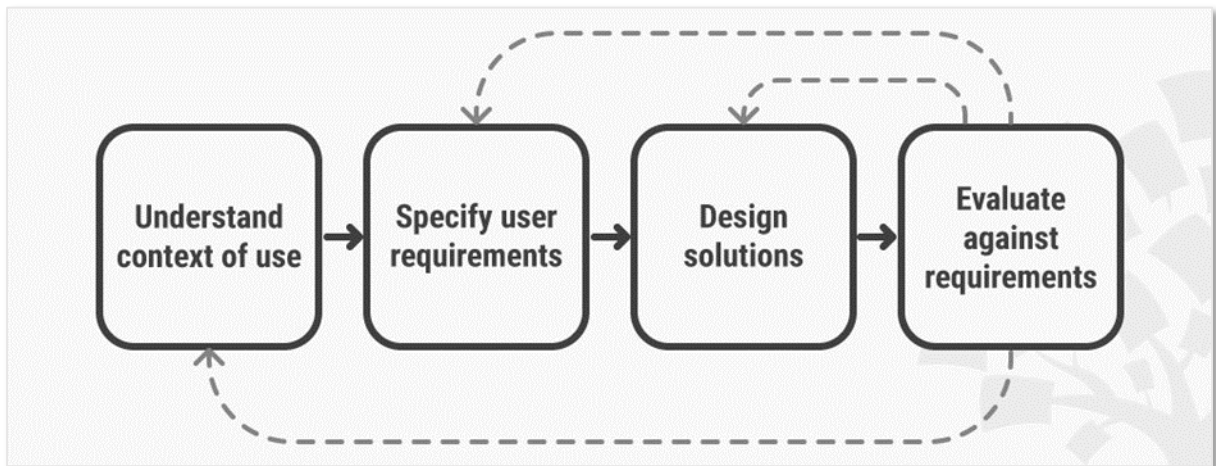
## What UX Designers consider - the Why, What and How

A UX designer will consider the Why, What and How of product use. The Why involves the motivations for adopting a product, whether they relate to a task they wish to perform with it, or to values and views associated with the ownership and use of the product. The What addresses the things people can do with a product - its functionality. Finally, the How relates to the design of functionality in an accessible and aesthetically pleasant way. UX designers start with the Why before determining the What and then, finally, the How in order to create products that users can form meaningful experiences with. In software designs, designers must ensure the product’s “substance” comes through an existing device and offers a seamless, fluid experience.

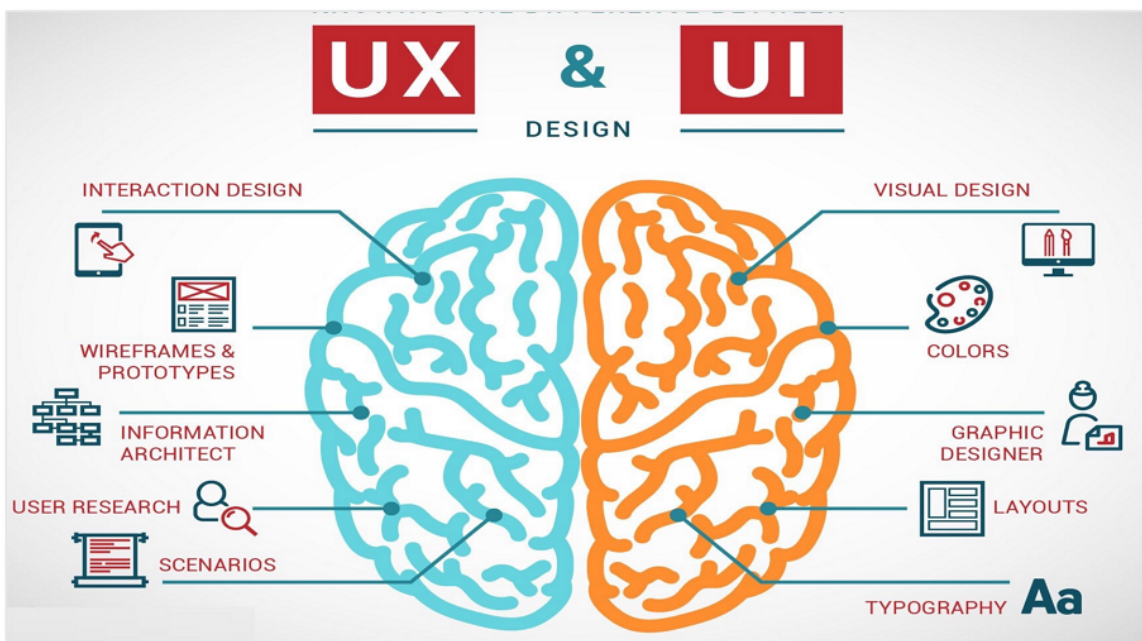


## UX Design is User-Centred

Since UX design encompasses the entire user journey, it's a multidisciplinary field – UX designers come from a variety of backgrounds such as visual design, programming, psychology and interaction design. Designing for human users also demands heightened scope regarding accessibility and accommodating many potential users' physical limitations, such as reading small text. A UX designer's typical tasks vary, but often include user research, creating personas, designing wireframes and interactive prototypes as well as testing designs. These tasks can vary greatly from one company to the next, but they always demand designers to be the users' advocate and keep the users' needs at the centre of all design and development efforts. That's also why most UX designers work in some form of user-centred work process, and keep channelling their best-informed efforts until they address all of the relevant issues and user needs optimally.



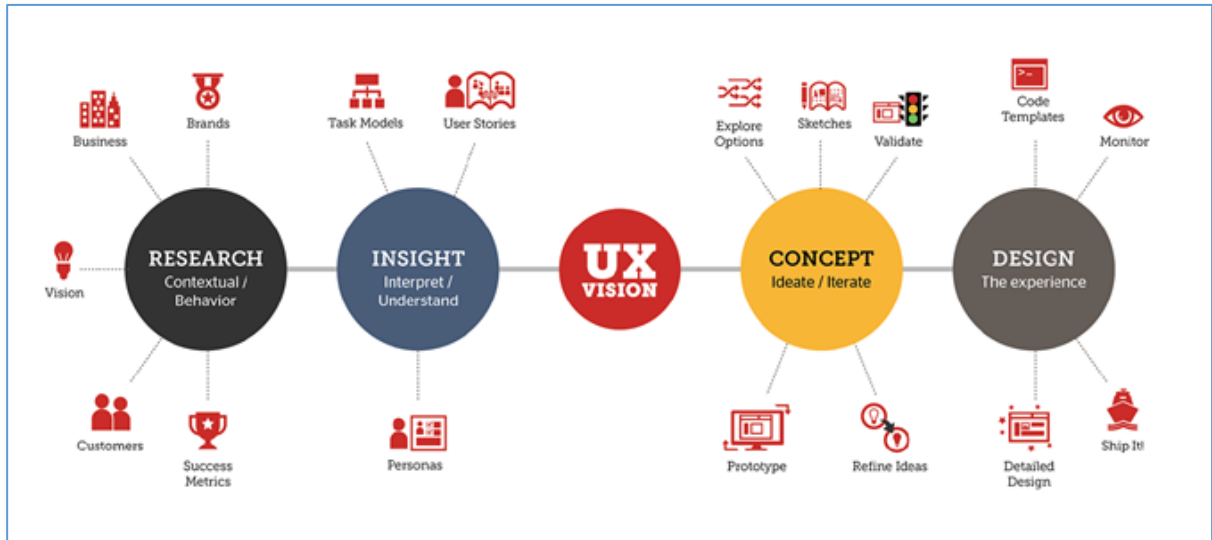
User-Centred Design is an iterative process that takes an understanding of the users and their context as a starting point for all design and development.



## The User Experience Process

The general user experience process is pretty similar in most cases, but still we cannot say that there is only one path that leads to the key to success. That's because we usually encounter a lot of external factors and a wide variety of projects. For example a “Website Project” can be totally different than a “Mobile App”. There are situations when product

owners ask for a proof of concept or a BETA version first, and after that want to work on estimations and how to tackle the project.



### Robert Dumitru's Steps ([www.centric.eu](http://www.centric.eu))

#### 1 - Project details and product description

This first step represents the documentation part. Depending on the situation, it can be done by the client or it can be a high-level investigation. Here we define the strong points, client target, and platforms or other relevant information relevant to project development. We can define specific dates and timelines regarding the project (kick-off, version beta 1.0, proof of concept, prototypes).

#### 2 - Product investigation

Here, we can discuss the info we got from the marketing department and how we can place the product on the market. We discuss aspects like potential age of the user, his financial status, IT knowledge, and field of work (banks, municipality etc.).

#### 3 - Project analysis and use cases

Based on the information above, we can create different scenarios, and we can build so-called "personas" - fictional characters who might use the product (for ex: manager, employee, new user, admin etc.).

#### 4 - Concept

Having the package ready, the next step is to create the architecture of the app, or better said the site map. Based on this, we can start sketching the wireframes to create a better

view of the functionality and the components of the app (what we can have on a specific page)

### 5 - Prototype

Now we can start creating a prototype based on the wireframes, so we can provide a nice flow to the app. This should be a clickable model and emulate the general functionality. This is a very important step for UI/UX designers who choose to skip UX and jump to the UI part. They are sometimes successful at making a good impression on the client, but this move will have a negative impact and slow down the overall process.

### 6 - The Visual

Once we have agreed on the prototype and the functionality, we can finally move to the user interface part. This can be done easily since we already have a prototype. The artistic part can be done with transitions and animations but also by the design approach decided upon (Material, iOS, Flat or a combination).

### 7 - Testing and validating the concept

Even if it doesn't seem to be a designer's job, the scenarios we discussed in the beginning force the designer to also be a tester (functionality wise and also implementation wise). In conclusion, this small plan could change depending on the moment and the state of the project. There are a lot of cases when the project has already started and we only need a redesign or a face-lift, cases when the project failed once, and cases where we have a lot restrictions (time, finances etc.). Also, concepts like Scrum/Kanban, Agile/Waterfall might have a big impact.

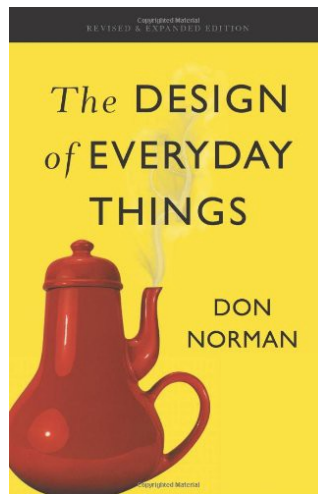
## Don Norman



Author of the watershed book *The Design of Everyday Things* and coined the term “User Experience” in the early days at Apple. Don Norman was recognized as the "Guru of Workable Technology" by Newsweek.

## Don Normans Principles of Design

These principles are from Don Normans seminal book, *The Design of Everyday Things*.



### Visibility

The more visible functions are, the more likely users will be able to know what to do next. In contrast, when functions are out of sight, it makes them more difficult to find and know how to use.

### Feedback

Feedback is concerned with sending back information about what action has been done and what has been accomplished, allowing the person to continue with the activity. Various kinds of feedback are available for interaction design-audio, tactile, verbal, and combinations of these.

### Constraints

The design concept of constraining refers to determining ways of restricting the kind of user interaction that can take place at a given moment. There are various ways this can be achieved.

## Mapping

This refers to the relationship between controls and their effects in the world. Nearly all artefacts need some kind of mapping between controls and effects, whether it is a flashlight, car, power plant, or cockpit. An example of a good mapping between control and effect is the up and down arrows used to represent the up and down movement of the cursor, respectively, on a computer keyboard.

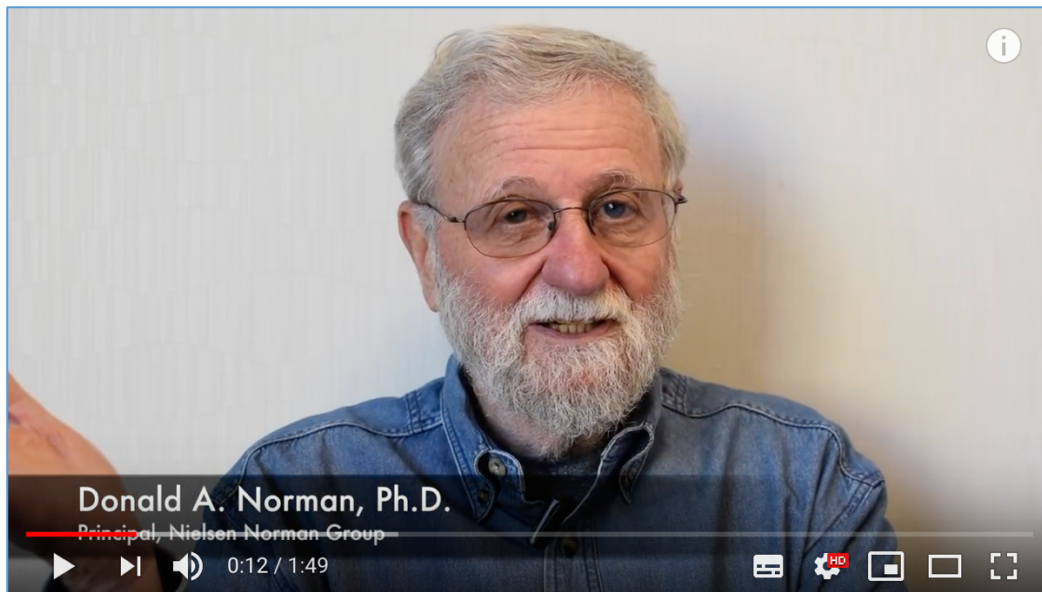
## Consistency

This refers to designing interfaces to have similar operations and use similar elements for achieving similar tasks. In particular, a consistent interface is one that follows rules, such as using the same operation to select all objects. For example, a consistent operation is using the same input action to highlight any graphical object at the interface, such as always clicking the left mouse button. Inconsistent interfaces, on the other hand, allow exceptions to a rule.

## Affordance

A term used to refer to an attribute of an object that allows people to know how to use it. For example, a mouse button invites pushing (in so doing acting clicking) by the way it is physically constrained in its plastic shell. At a very simple level, to afford means to give a clue (Norman, 1988). When the affordances of a physical object are perceptually obvious it is easy to know how to interact with it.

## Don Norman on UX



<https://www.youtube.com/watch?v=9BdtGjoIN4E>

## Jakob Nielsen



Author of the quintessential usability checklist *Ten Usability Heuristics* and an early champion of usability testing, *Jakob Nielsen* was recognized as the "Guru of Usable Web Pages " by the New York Times.

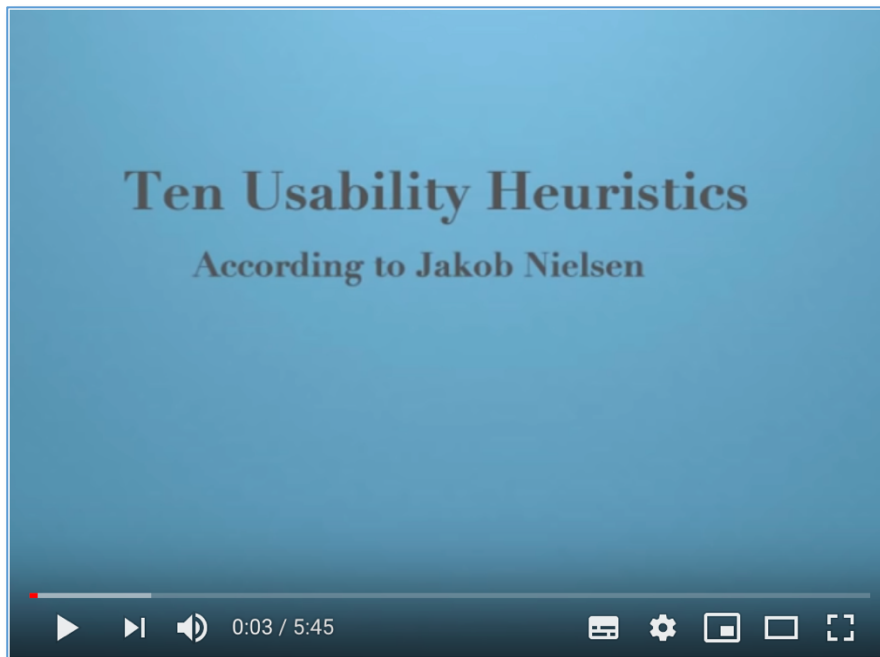
## Ten Usability Heuristics for User Interface Design

"The 10 most general principles for interaction design. They are called 'heuristics' because they are more in the nature of rules of thumb than specific usability guidelines."



These are one of the most used heuristics for User Interface Design. They were developed by Jakob Nielsen together with Rolf Molich in the early 90's. The final set, which you see here, was released by Nielsen in 1994.

The heuristics are explained in greater depth in this video.



<https://youtu.be/hWc0Fd2AS3s>

### **Visibility of system status**

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Read the full article at <https://www.nngroup.com/articles/visibility-system-status/>

### **Match between system and the real world**

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

Read the full article at <https://www.nngroup.com/articles/match-system-real-world/>

### **User control and freedom**

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue.

Support undo and redo.

### **Consistency and standards**

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

### **Error prevention**

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Read the full article at <https://www.nngroup.com/articles/slips/>

### **Recognition rather than recall**

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.

Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Read the full article at <https://www.nngroup.com/articles/recognition-and-recall/>

### **Flexibility and efficiency of use**

Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.

Allow users to tailor frequent actions.

### **Aesthetic and minimalist design**

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

### **Help users recognize, diagnose, and recover from errors**

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

More error message guidelines can be viewed at <https://www.nngroup.com/articles/error-message-guidelines>.

### **Help and documentation**

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

## **Shneiderman's "Eight Golden Rules of Interface Design"**

These Golden Rules of Interface Design are taken from the book, *Designing the User Interface*, which Ben Shneiderman co-authored. They were originally created in 1987 from the research Shneiderman done in Human Computer Interaction. They are applicable for most interactive systems.

These principles can help you create a well-designed User Interface and thereby improve the usability of the system.

**Source:** [Designing the User Interface: Strategies for Effective Human-Computer Interaction](#)

### **Strive for consistency**

Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.

### **Enable frequent users to use shortcuts**

As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

### **Offer informative feedback.**

For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

### **Design dialog to yield closure.**

Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.

### **Offer simple error handling.**

Where possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.

### **Permit easy reversal of actions**

This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

### **Support internal locus of control**

Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.

### **Reduce short-term memory load**

The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

## Reflection Exercise

Reflect on what you have learned about UX..



Use the space below



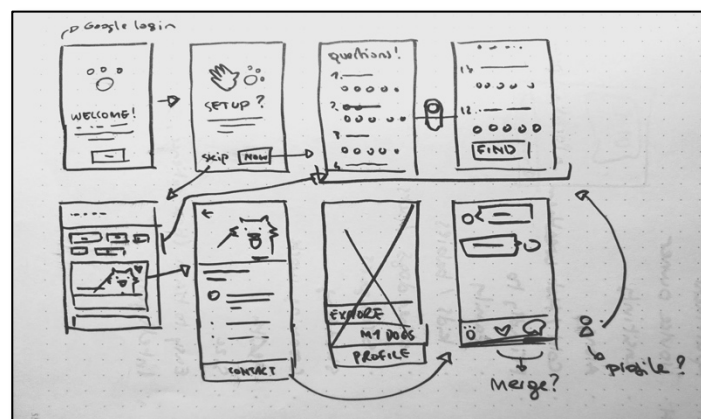
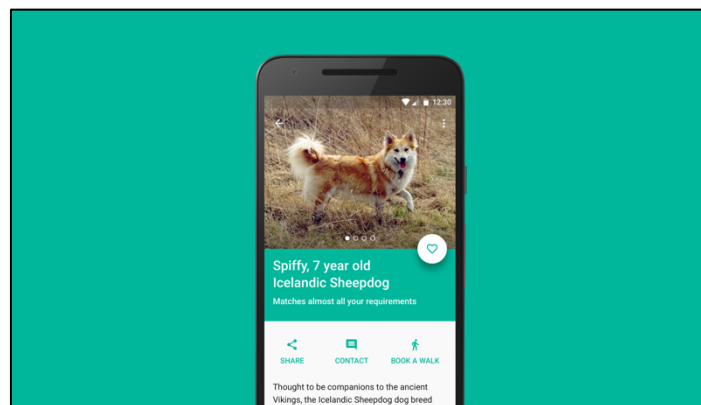
## Breakout Exercise

Review Pol Kuijken's case study on his 'Google Design Exercise: Solving the shelter' problem when he was applying for a position at Google.

<https://medium.com/@polkuijken/pet-adoption-8798b14af117>

The design brief was:

*Millions of animals are currently in shelters and foster homes awaiting adoption. Design an experience that will help connect people looking for a new pet with the right companion for them. Help an adopter find a pet which matches their lifestyle, considering factors including breed, gender, age, temperament, and health status. Provide a high-level flow and supporting wire frames.*



## UX Resources

- <https://www.interaction-design.org>
- <https://uxmastery.com>
- <https://wireframestogo.com/>
- [https://www.ted.com/talks/don\\_norman\\_on\\_design\\_and\\_emotion?language=en](https://www.ted.com/talks/don_norman_on_design_and_emotion?language=en)
- <https://www.youtube.com/watch?v=v6n1i0qojws>
- <https://www.youtube.com/watch?v=a5KYIHNKQB8>
- <https://www.youtube.com/watch?v=MELKuexR3sQ>
- <https://www.springboard.com/blog/ux-design-principles/>
- <https://asktog.com/atc/principles-of-interaction-design/>
- <https://www.nngroup.com/>
- [www.centric.eu](http://www.centric.eu)
- [https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/UXPIN\\_PL/U141030B.pdf](https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/UXPIN_PL/U141030B.pdf)
- [https://ddf46429.springboard.com/uploads/resources/1548810145\\_Ebook\\_\\_UX\\_Fundamentals\\_\\_Learn\\_the\\_Basics\\_of\\_IA\\_UX\\_and\\_UI\\_Design.pdf](https://ddf46429.springboard.com/uploads/resources/1548810145_Ebook__UX_Fundamentals__Learn_the_Basics_of_IA_UX_and_UI_Design.pdf)
- <https://medium.com/tag/ux>
- <https://www.interaction-design.org/courses>
- <https://www.mockplus.com/blog/post/what-is-interaction-design-and-how-it-works>

- <https://hackernoon.com/why-ux-design-must-be-the-foundation-of-your-software-product-f66e431cc7b4>
- <https://www.nngroup.com/>
- <https://www.designprinciplesftw.com/collections/don-normans-principles-of-design>
- <https://badhtml.com/>

***Examples & Breakout sourced from [www.interaction-design.org](http://www.interaction-design.org), [www.nngroup.com](http://www.nngroup.com)  
[www.medium.com](http://www.medium.com) and [www.centric.eu](http://www.centric.eu).***