



Web Fundamentals - Session 1

Introduction - Course Overview

What will we learn

- How websites are made
- How to put up your own webpage
- Basic skills and tools that will be useful for any coding you do

We hope you will become curious about how the Internet works and will look at websites in a different way.

How courses will be run

The hardest thing about learning to program is knowing **where** to start and **what** to learn.

The course **aims** to provide a basic overview of the technologies used, along with the tools and resources to discover more.

The **focus** of this course is learning the basics of how and why things work and to provide the basis to build upon in future courses. We will **not** be able to cover everything in great depth or comprehensive detail.

Sessions will be as **hands-on and practical** as possible. Each week there will be a number of tasks to do in between the sessions to reinforce what you have learnt.

It's up to you whether you do the tasks or not, but the more you put in the more you will get out! If you're ever in doubt, [Google it](#), check [StackOverflow](#), ask the person beside you, or ask one of us!

Note:

DEMOS will be in blue with a salmon background - **Like this.**

TASKS will be in red with a pale-blue highlighted background - **Like this.**



Web Fundamentals - Session 1

Chapter 1 - What makes a website?

A **website** is a collection of linked webpages (and their associated resources); a set of related files that is compatible to your browser.

A **webpage** is just a file that your web browser is able to read and display, written in a markup language; **HTML**.

Associated (additional) **resources** can be embedded into your HTML file. These include, and are not limited to:

- page styles; **CSS** (Cascading StyleSheets)
- scripts - *for interactivity*; **JavaScript**
- media - *videos, music, etc.*

The website's main webpage (or *point of entry*) is referred to as the **homepage**, or a **landing page**. This is predefined, and its document name must be set as [index.html](#).

To **display** the page on the client-side device, a browser starts out by reading the HTML. This is why additional resources, such as CSS & JS, are written in dedicated documents and are “plugged-in” or linked into HTML documents, as we will see later.

Viewing HTML, CSS and JS of any given webpage online

Because the HTML, CSS and js are sent to your browser, it is easy for you to look at them. **There are no secrets in HTML, CSS or JS**. If there's a part of a webpage that you like, it's easy to find out how it is coded and use the technique yourself.

Every browser provides a way to look at the source of the page you're currently viewing, these are called **developer tools**. They are the best way to investigate a web page.

In Chrome you do **View > Developer > View Source**. This will show you the raw HTML but isn't always the easiest thing to look at.

Several browsers also provide developer tools, which allow you to *interactively* view the page source. In Chrome you can access these by doing **View > Developer > Developer Tools**.



Web Fundamentals - Session 1

Over the course you will be using them a lot on your own pages, especially when things aren't working exactly as you expect.

There are a few features of the Chrome developer tools that it is worth pointing out now.

Task:

1. Open any website in Google Chrome, or the [Code First: Girls website](#)
2. View the page source by doing one of the following:
 - o View > Developer > View Source
 - o Tools > View Source
3. Open the developer tools by doing one of the following:
 - o View > Developer > Developer Tools
 - o Tools > Developer Tools
4. Use the magnifying glass in the top left () to hover over bits of the page and find the related HTML.
5. Hover over the HTML code in the developer tools box and watch as different parts of the page are highlighted.
6. Try changing some of the CSS on the right hand side. To undo any changes just refresh the page.
7. Have a look on the [Resources](#) tab. See if you can find the CSS, javascript and image files used on this page.
8. Visit a few of your favourite websites and repeat this process.

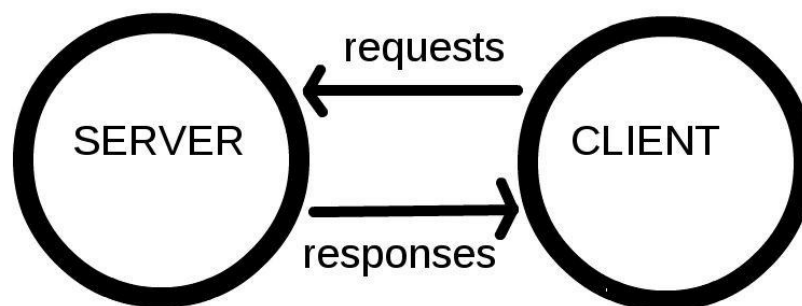


Web Fundamentals - Session 1

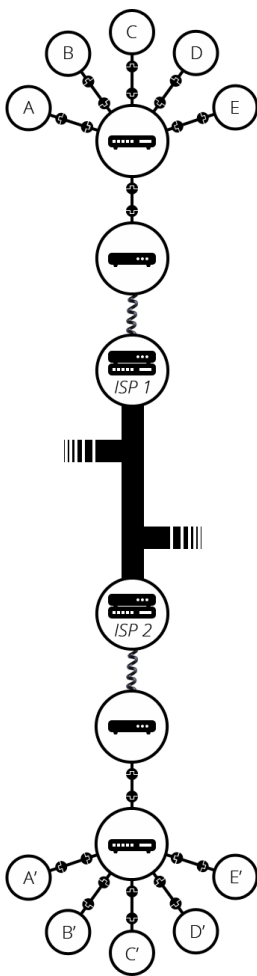
Chapter 2 - How the Internet Works

How does someone find your website? How do we publish our websites for other people to find?

Computers connected to the Web are called **clients** and **servers**



The computers that hold web pages, sites and applications are called **servers**. *Web servers are large, specialised computers that are (almost) permanently connected to the internet.*



The computers that allow people to access the web by providing software (browsers) for them to do so are called **clients**. They are the typical user's Internet-connected device.

The **Internet** is the technical infrastructure that makes the Web possible. You can think of the Internet as a massive network of computers that can send files to one another.

As more and more computers join the network, it gets a little bit too complicated to keep track of.

To solve this problem, we use special tiny computers called **routers**, which only have one job: to make sure that a message sent from one computer to another is done correctly.

By connecting computers to routers, and then routers to each other, we are able to scale infinitely.



Web Fundamentals - Session 1

To connect routers to each other, we make use of existing telephone infrastructure by using **modems**, and connect our network to an Internet Service Provider (**ISP**) - a company that manages special routers which all link together and can access special routers of other ISPs.

This is how the infrastructure of the Internet is created.

Images from Mozilla Web Developer Tutorials, more information [here](#).

Chapter 3 - How the Web Works

Many of our web interactions begin with a URL (*uniform resource locator*) being typed into our web browser address bar.

Let's look at an example: <http://www.bbc.co.uk/news/>.

This URL has several different parts to it:

- [http](#) : the *protocol* or *how* to fetch the information
- [www.bbc.co.uk](#) : the *host* or *where* to fetch it from
- [/news](#) : the *path* or precisely *what* information to fetch

When we type the URL into the address bar a request is sent over the internet and some information is returned to us.

The protocol describes how the information is transmitted. Other possibilities include [https](#) for secured communication, [ftp](#) for file transfer and [git](#) which you'll learn about later.

The host describes where the information should come from and the path tells that location precisely what information to send.

In general a URL can be more complicated than this. URLs can also contain *query parameters*, *fragments* and *port information*. We will leave these for now but will point them out when we meet them later. Instead we will focus on exactly what information is being sent and who is sending it.

Each computer on the internet has an address (an *IP address*) so that requests can be sent to it and files returned - much like a telephone number. The backbone of the internet is a network of *routers* that are responsible for routing files from one IP address to another.

DNS

IP addresses are a sequence of numbers and '.'s such as [212.58.244.67](#). These aren't very easy to remember. Instead the internet works on a domain name system, that matches domain names such as [bbc.co.uk](#) to IP addresses.



Web Fundamentals - Session 1

Task:

Type **74.125.224.72** into your browser's address bar. What happens?

One of the first things that happens when you type a URL into your browser is a *DNS lookup*: your computer contacts a DNS server (*Domain Name System server*), which is basically a massive address book of IP addresses. The DNS server converts the domain name from the URL (e.g. **bbc.co.uk**) into an address for a server (such as **212.57.244.67**).

Once the address of the web server has been found your request is forwarded on to it, the web server will interpret your request and send back one or more files.

Static vs dynamic sites






There are two main possibilities server-side: either your site is static or dynamic.

- In a **static** site, all pages are pre-prepared and the web server just sends them to the browser.
- In a **dynamic** site, pages are prepared on-the-fly, pulling information out of a database depending on what the user asked for.

Most of the sites you can think of will be dynamic sites (e.g. facebook.com, reddit.com, amazon.com, ..).

Server-side technologies

There are many options for building a dynamic server-side site. Common choices are:

 Ruby on Rails	 PHP	 Django	 node.js	 WORDPRESS
Web development framework written in Ruby (programming language)	Programming language popular in the early 2000s.	Web development framework written in Python (programming language)	Web development framework written in Javascript.	A blogging platform (now capable of much more) written in PHP.

Putting up a website

If you want to put up your own website at your own domain name you need two things:



Web Fundamentals - Session 1

1. A web server to serve your site
2. A domain name to point towards it

There are many options for web servers - you don't have to physically have your own one. Many companies that will offer **web hosting**, often providing you with space on a shared server. Later in the course we will use the free hosting offered by GitHub through GitHub Pages.

To buy a domain name you need to use a domain registrar. This is not necessary for our course, but it will be useful to know how they work.

Examples include 123-reg.co.uk, godaddy.com and namecheap.com. You pay the registrar to contact the body who manages a TLD (e.g. .com, .org., .co.uk) and put your server's IP address in their DNS address book.

Many domain registrars will try to sell you hosting and other site building tools when you buy a domain. It's important to remember that the domain name is completely separate from the hosting and you don't need to do them both through the same company - don't buy anything you're not going to use.

Chapter 4 - Creating an HTML page

One of the nice things about HTML is you don't need any fancy software to test it out on your laptop: all you need is a text editor and a web browser.

Task:

1. Create a folder called `coding_course` to hold all your work for the course.
2. Inside your `coding_course` folder create another folder called `first_site`.
3. Open Sublime Text.
4. Write "Hello"
5. Save the file as `index.html` in the `first_site` folder
6. Open `index.html` in Chrome. Depending on your version of Chrome
 - o File > Open
 - o Cmd-o (Mac) or Ctrl-o (Windows)

Why index.html?

In the old days of the web, navigating a website was a lot more like moving around the folder system on your laptop. You would go to a base site and there would just be a list of the files and folders available: an index. Because of this `index.html` is still the default file that a server will serve when you navigate to a folder in the web browser.

Task:

1. Go back to 'index.html' in Sublime Text.



Web Fundamentals - Session 1

2. Change the text to

```
<h1>Hello</h1>
```

1. Go back to 'index.html' in Chrome and refresh the page (Cmd-R)

This is your first line of HTML. It has an open tag and a close tag. They tell your browser to display the text in-between as a heading.



Web Fundamentals - Session 1

Chapter 5 - HTML Basics

Elements, Tags & Attributes

HTML uses a *predefined* set of **elements** for different **types of content**; they define the **semantic value** (or meaning) of their content. Elements include two matching tags and everything in between. They contain one or more "tags" which either contain or express content.

For example, the "<p>" element indicates a paragraph; the "" element indicates an image.

HTML attaches special meaning to anything that starts with the less-than sign ("<") and ends with the greater-than sign (">"). Such markup is called a **tag**.

Tags are enclosed by *angle brackets*, and the closing tag begins with a forward slash.

Make sure to **close** the tag, as some tags are closed by default, whereas others might produce unexpected errors if you forget the end tag. An example of a tag that closes by default is the image tag.

```

```

The start tag may contain additional information, also known as an **attribute**. Attributes usually consist of 2 *parts*, its **name** and corresponding **value**. In the example below, the attribute name is "class", and the class of the div is "main".

```
<div class="main">
```

Valid Code

To write **valid** code, you must properly nest start and closing tags, that is, write close tags in the opposite order from the start tags.

Valid code:

```
<div>
```

```
  <p>
```

```
    <em>I <strong>really</strong> mean that</em>.
```



Web Fundamentals - Session 1

</p>

</div>

Invalid code:

```
<div> <p> <em>I <strong>really</em> mean</div> that</strong>.</p>
```

HTML DOM

The **Document Object Model** specifies the hierarchical layout of the HTML document. It is an Application Program Interface (*set of routines, protocols and tools for building software applications*), or **API**, that represents and interacts with any HTML or XML (markup) document.

It is loaded in the browser, and represents the document as a node tree, with each node representing a part of the document. In other words, it tells the browser where to look for a specific thing in the document.

Every HTML5 document requires this layout:

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Page title</title>
```

```
  </head>
```

```
  <body>
```

```
    ...
```

```
  </body>
```

```
</html>
```

- The **!DOCTYPE** tells you what version of html you're using (html5, html4 ...):
 - With html5 it's simple - you just write **html**.
- Everything is wrapped in an **<html> ... </html>** tag
- Things within the **<head> .. </head>** are used to provide information about the page
- Only things within the **<body> ... </body>** tags are displayed on the page
- ... for example the text within **<title> ... </title>** will be displayed in the browser bar



Web Fundamentals - Session 1

HTML Syntax

[Demo here](#) - Talk through this!

Now you will use these ideas to create a richer web page.

Task:

1. Go to the github repository for this session: https://github.com/code61/learning_html
2. Download the code into your coding_course folder (by clicking 'Download ZIP' in the bottom right).
3. Open the whole folder in Sublime Text
4. Open the file `example.html` in Chrome and look around with the developer tools
5. Open the file `notes.html` in Sublime Text.
6. Change `notes.html` into valid html so that it looks like `notes_solution.jpg`



Web Fundamentals - Session 1

Session 1 Homework

Finishing off

Task: Finish the HTML exercise from the last section.

Preparation for next time

Task:

1. Read and **follow** [this guide](#) and [this guide](#) on how to use GitHub & version control - **This is vital.**
2. Complete the whole of Project 2 on the [General Assembly Dash](#) site.
3. (Optional) Do the lessons from the [Codecademy Web Track](#) Sections 4, 5 & 6.

Make a start on your own site

Task:

Use what you've learnt from the HTML exercise and the Dash projects to improve your `first_site/index.html`. Maybe add some content about yourself.

Don't worry if it doesn't look great yet - we'll be working on it more in the next few weeks. Just make sure it says something more than 'Hello'!

Extra Resources

[This video](#) talks about how the Internet works in 5 minutes

[A summary](#) of the different components of the Internet

[File organising](#) for your website

An article from Mozilla's Developer Guides: [Introduction to HTML](#)

[W3 Schools HTML Tutorial](#)

[HTML Terms Glossary](#)

[Web Monkey HTML Cheatsheet](#)



Web Fundamentals - Session 1

[Simple HTML Guide Cheatsheet](#)

[A HTML Validator that checks your HTML code](#)