Developing Websites – Basic HTML

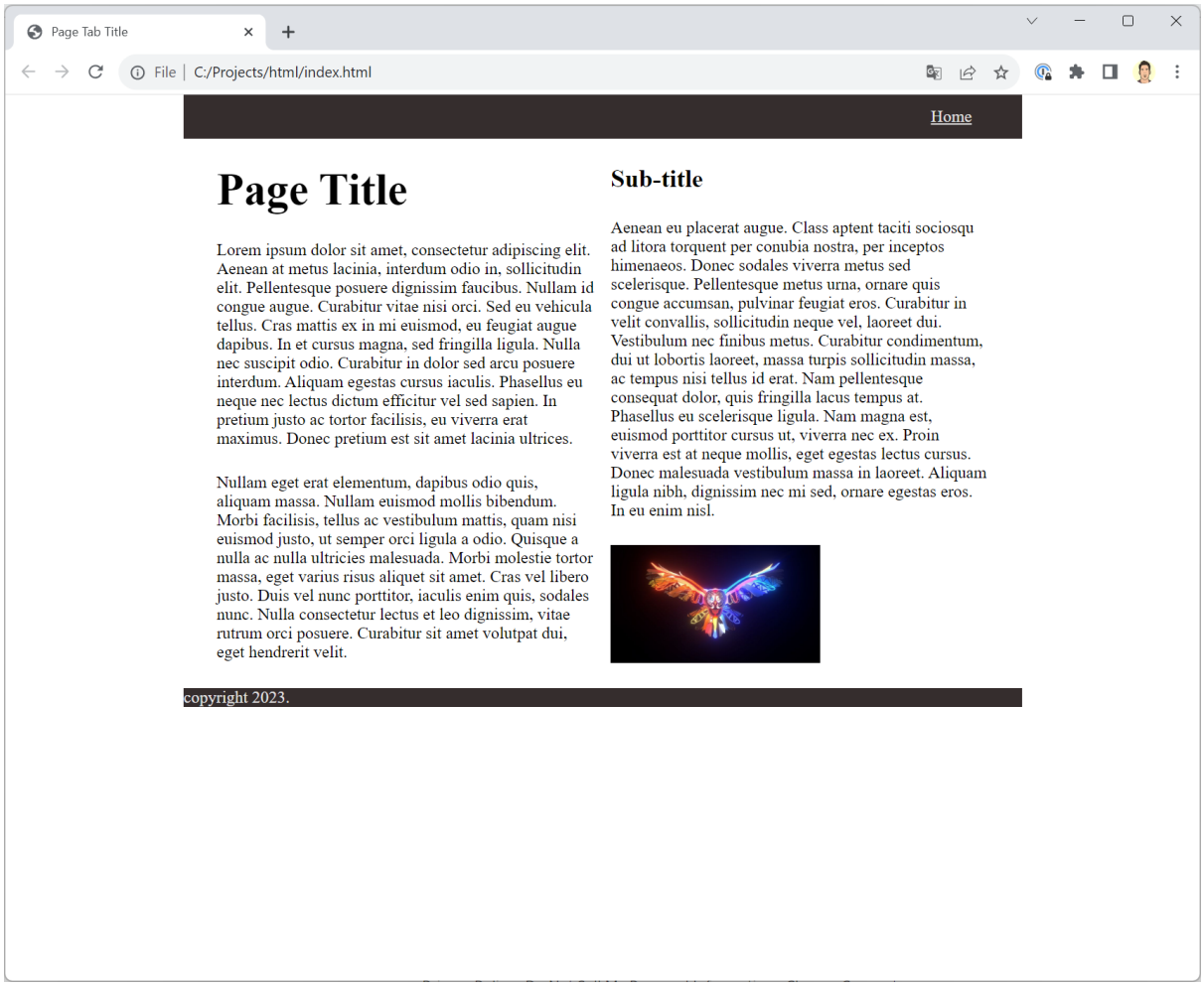
Introduction

You can make websites with very few tools. In this tutorial I'm going to be using Microsoft Visual Studio Code, which you can download here:

https://code.visualstudio.com/

but you can use any plain text editor, including good old Windows Notepad.

We're going to build a simple webpage that looks like this:

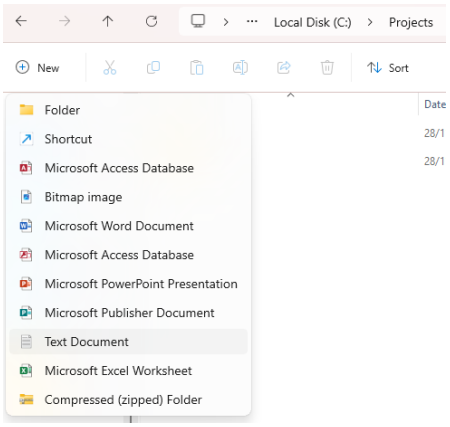


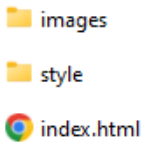
Files

Building websites is all about managing files. First you need to create a folder to contain all of the files that your website needs. Inside that folder, you should create 2 more folders: "images" and "style".

Inside the main project folder, you need to create a new Text Document:

Call it "index.html":





Do the same thing inside the "style" folder, but create a text file called "style.css".

Basic structure

HTML files are plain text files containing Markup Language tags (that's the ML part of HTML). All web pages start off with the same structure. Using your text editor of choice, enter the following:

```
<!doctype html>
<html>
    <head>

    </head>

    <body>



    </body>

</html>
```

All HTML files start with the *doctype* element. This tells the web browser what version of HTML the file is using. The *html* doctype is HTML version 5, which is the most modern version. The ensures that the browser will render the page correctly.

The *head* element of an HTML page is where the browser looks to see what else it might need to display this particular page.

The *body* element is where all of the actual page content is placed.

We'll create 3 new *section*s inside the body:

```
    <body>

        <section id="nav_bar">

        </section>

        <section id="content">

        </section>

        <section id="footer">

        </section>

    </body>
```

Notice that each *section* element has an *id attribute*. This is important later when we want to target each of these sections with some CSS.

```
13          <section id="content">
14              <section id="leftCol">
15
16              </section>
17
18              <section id="rightCol">
19
20              </section>
21          </section>
```

Inside the *section* that we've called *content* let's add two more nested sections:

Two new *sections* each with a unique identifier – values for *id* attributes must be unique for each page.

Content

Now we have a structure for our page, we can start to add some content. It's sometimes useful to have a big bunch of text to use as a placeholder when you're designing a new web page. I use this website:

https://www.lipsum.com/

it generates a bunch of nonsense text that sort of looks like words and allows you to see what text might look like on the page, without distracting you with actual words.

```
16          <section id="leftCol">
17              <h1> Page Title </h1>
18              <p>
19                  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean at met
20              </p>
21
22              <p>
23                  Nullam eget erat elementum, dapibus odio quis, aliquam massa. Nullam e
24              </p>
25          </section>
```

Don't try typing the Lorem ipsum bit out, there's lots of it! Just copy and paste it from the website.

Do something similar in the right column too:

```
25          <section id="rightCol">
26              <h2> Sub-title </h2>
27              <p>
28                  Aenean eu placerat augue. Class aptent taciti sociosqu ad litora tor
29              </p>
30          </section>
```

Navigation

Let's add a link to the navigation bar at the top:

```
11          <section id="nav_bar">
12              <a href="index.html">Home</a>
13          </section>
```

This isn't really much of a link. It just links back to the same page we're already on!

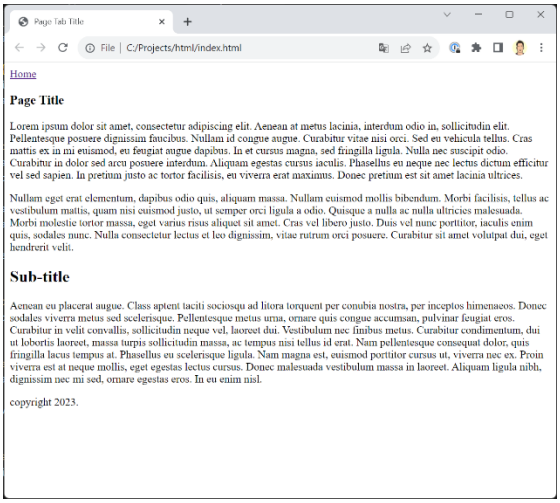
Footer

We don't need much here, let's just put a copyright notice in there. We own all our own work!

```
33          <section id="footer">
34              copyright 2023.
35          </section>
```

Giving it some style

The page we've created doesn't look like much. We need to add some code to the *style.css* file that we created earlier, and we need to link that file to our .html page so the browser knows that it needs to load it.



Up in the *head* section of the html file, we need to add 3 things. Only 1 of these is strictly necessary, but the other 2 are useful things to have:

```
3     <head>
4         <meta charset="utf-8">
5         <link rel="stylesheet" type="text/css" href="style/style.css">
6         <title> Page Tab Title </title>
7     </head>
```

The middle line there, the *link* element is what tells the browser to go and get the *style.css* file, and apply all the rules to this page when it renders it. The *title* element contains the text displayed in the page's browser tab. The *meta* element tells the browser what kinds of letters we're using. We don't really need to worry about that, just put it in there for now.

Now that our stylesheet is attached, we can start to style the elements we have in the page.

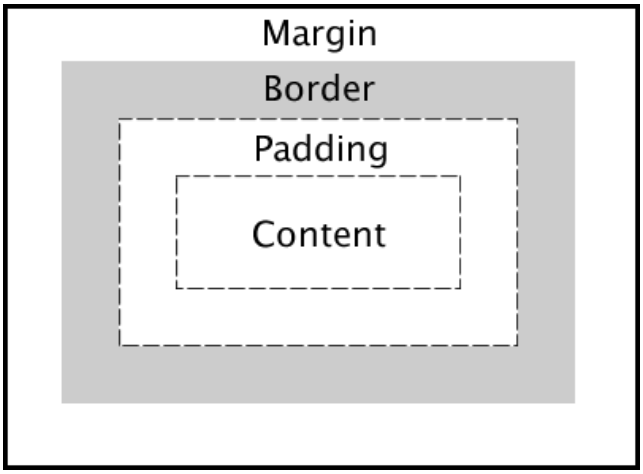
This code all goes in the *style.css* file:

```
1   * { margin:0px; padding:0px; }
2
3   body {
4       width:800px;
5       margin: 0px auto;
6   }
```

Line 1 here is something I like to call a *reset rule* the asterisk means that the rules apply to ALL of the elements on the page. This rule sets both the margin and the padding of all of the elements on the page to zero.

Line 3 is a rule that targets the *body* element on the page. It sets the width of the body element to 800 pixels wide and then sets the left/right margins to *auto*. This has the effect of making everything sit in the middle of the browser window.

Box Model Aside



The CSS Box Model describes how elements on a page are arranged. Each element – in our page so far we have *section* elements, we have *p*aragraph elements, *a*nchor elements and so on – have padding, a border and a margin. These determine how each element sits against its neighbour. This can get complicated, but by setting everything to zero with our reset rule, we only need to worry about it when we want to change the way something looks.

Navigation Bar

These rules will style our navigation bar at the top of the page:

```
8   #nav_bar {
9       background-color: #362d2d;
10      color: #e4e4e4;
11
12      padding: 12px;
13      padding-right: 48px;
14
15      text-align:right;
16  }
17
18  #nav_bar a {
19      color: #e4e4e4;
20  }
```

When we created our html page we created 3 main *sections* and gave each of them an *id*. One of those was *nav_bar*.

Line 8 here uses something called a *selector*, we start it with an *octothorpe* (they're not called hashtags!) which tells the browser to look for an element with the id specified.

Line 18 is another selector. It is targeting all *a* elements found *inside* the nav_bar *section*.

The rules themselves should be fairly self-explanatory.

Footer

```
69  #footer {
70      background-color: #362d2d;
71      color: #e4e4e4;
72  }
```

Footer is very simple too. We aren't really doing very much with it in this example, so it's just going to sit at the bottom and be the same colour as the navigation bar.

Columns

```
22  #content {
23      padding: 24px;
24
25      overflow:auto;
26  }
27
28  #content h1 {
29      font-size:32pt;
30  }
31
32  #content p {
33      margin-top:24px;
34  }
35
36  #leftCol {
37      width:48%;
38      float:left;
39
40      padding:0px 1%;
41  }
42
43  #rightCol {
44      width:48%;
45      float:right;
46
47      padding:0px 1%;
48  }
```

Again, we use the *id* selectors to target the appropriate elements on our page.

Most of the rules should be fairly self-explanatory again, except for maybe the *float* rule.

*Float*ing elements are actually a bit unfashionable these days, and for layouts any more complex that we have here we'd want to use the more modern *flex-box* layout.

float: left forces an element to line itself up against the left-hand-side of its parent object (our *content section)*, *right* floats it to the right. By setting the left and right column *section*s width to 48%, we leave enough space for them to sit side-by-side.

You might think that they should take up 50% of the parent element's width, but that would cause them to be too big, because we also add 1% padding to the left and right of each element.

These sorts of oddities are why once you get to wanting more complex layouts, flex-box is a much more powerful tool.

An image

The only remaining item we haven't covered is the image on the page.

We can add an image to an html page with the following element:

```
32
33                  <p><img src="images/owl.jpg" alt="An Owl"></p>
34              </section>
```

The image file needs to be in the *images* folder we created at the beginning of the project.

```
50  #content img {
51      max-width:200px;
52  }
```

It's also helpful to add a css rule to control the maximum width of an image, especially when we're using a fixed-width page as we are here.

And that's it. A *very* simple HTML template that can be used to make a quick webpage about something interesting!