Lab 4: HTML5 / Creating a Form

Objectives

1. Understand the use of more ‘modern’ HTML5 semantic elements.
2. Enhance styling with CSS3.
3. Built a basic HTML form.
4. Styling a form.

Setting up your webspace

Download the ZIP file for this week’s lab. Extract the files and folders. In Sublime Text create a clean project by removing previous folders and then adding the new folder.

1. Select `Project > Remove all Folders from Project`.
2. Select `Project > Add Folder to Project` to find the *lab4* folder.
3. The files in the *lab4* folder will now appear on the left-hand side.

View the files

Your file structure should appear as follows:

```
f:\year1\web-dev\lab4\
f:\year1\web-dev\lab4\images
f:\year1\web-dev\lab4\styles
```

There will be six HTML files created for you *index.html*, *qualifications.html*, *skill-set.html*, *work-experience.html* that we have seen previously and two new files *contact-me.html* and *thank-you.html* that relate to the contact form we’ll create.

Using More Semantic HTML5 Elements

Our design is very dependent the on `<div>` tags. Such dependency can lead to what is known as 'div soup' where HTML is dominated by many nested `<div>` tags. With HTML5 we can reduce this dependency on `<div>` and use some of the newer semantic tags. This technique will make our pages easier to read for both developers as well as automated systems as the HTML will now be more descriptive.

Open the *index.html* page and locate the *div#header*.

```
<div id="header">
…
</div>
```

Replace this with the more semantic `<header>` element. Don't forget to change the closing </div>.

```
<header>
…
</header>
```

Next find the *div#nav* and change it to use `<nav>`. Again don't forget the closing pair.

Continue the process changing the *div.section* to <section> and the *div#footer* to `<footer>`.

Your structure of the *index.html* pages should now be as follows:

```
<div id="container">
     <header>
            <div id="logo">
                …
          </div>
         <nav>
            <ul>
                …
            </ul>
        </nav>
     </header>
     <div id="content">
          <section>
                    …
                </section>
     <div class="sidebar">
          …
     </div>
     </div>
     <footer>
          …
     </footer>
</div>
```

Update the Stylesheet

If you test your pages you will now find some styling is no longer applied. This is because we've changed the HTML.

For example by replacing the `<div id="header">` with `<header>`, the selector `#header` is no longer valid. Therefore change the previously used ID and class selectors to use HTML selectors.

For example in the CSS we have.

```
#header{
     background-color:#263248;
     border-radius:8px;
     overflow:auto;
}
```

Change this to:

```
header{
     background-color:#263248;
     border-radius:8px;
     overflow:auto;
}
```

You need to do this for all rules that reference `#header`, `#nav`, `.section` and `#footer`.

Tip: Remember, `.section` is a class selector.

Why Bother?

This exercise may seem a little academic as visually the page looks no different. However, what you have now is more semantic HTML – HTML with more implicity meaning.

Pages with better semantics (meaning) are more machine friendly. That could be search engine indexing bots or accessibility software. With the likely expansion of the Internet of Things adding meaning to your HTML will help future proof it.

Adding some CSS3 Styling

Beyond standard styling CSS3 can be used to add some visual elements such as:

- text-shadow
- box-shadow
- border-radius
- background:rgba()
- background gradient

For more details see:

http://www.mustbebuilt.co.uk/2012/09/21/css-three-is-a-magic-number/

Experiment adding some CSS3 styling to your pages. Things to try:

1. Add a Box Shadow to the header.
2. Add a text shadow to the `<h1>`.
3. Experiment with border radius values on images.

Adding a Contact Form

To create a HTML form we use the `<form>` tag.

Please note the `<form>` element is a long standing part of HTML but the form family of elements have been given some new functionality with HTML5.

A list of the original form elements can be found here:

http://www.mustbebuilt.co.uk/demo/html5/form-elements.html

Newer HTML5 features that may or may not be supported in your browser are listed here:

http://www.mustbebuilt.co.uk/demo/html5/form-elements-h5.html

Open the *contact-me.html* file. Find the HTML comment:

```
<!-- form here -->
```

This is where we'll add the form. Add the basic `<form>` tag:

```
<form>
</form>
```

The `<form>` has two key attributes that control how the data in the form is processed. The `method` attribute sets which http method is used to send the data and the `action` attribute where the form links to.

Note: To process the data we need a server technology we'll be just sending the user to the page *thank-you.html*.

Add the `action` attribute:

```
<form action="thank-you.html">
</form>
```

Adding a Single Line Text Input

The `<input>` tag has a number of different types and requires a number of different attributes.

The `<input type="text">` is perhaps the most commonly used. It is used to generate a single line text field.

```
<input type="text" name="yourName" id="yourName">
```

As with all form elements that return a value it is important to have a `name` attribute. This is used to create a name/value pair to submit the data.

Often with a form, a `<label>` element is used to attach a label to the form field. To connect a `<label>` to a particular form element the label is given a `for` attribute that must match the `id` attribute of the target form element ie:

```
<label for="yourName">Name:</label>
<input type="text" name="yourName" id="yourName">
```

Once a label and form element are connected it allows users to click on the label and give the form element the focus of the cursor.

A HTML5 attribute that can be added to the above is `placeholder`. This places some text in the field to act as a prompt to the user. The placeholder will clear as the user enters their own data.

```
<div>
<label for="yourName">Name:</label>
<input type="text" name="yourName" id="yourName"
placeholder="Please Add Your Name">
</div>
```

Add an input for email. The input type of `email` will help users on mobiles by offering up an email friendly keyboard.

```
<div>
<label for="email">Email:</label>
<input type="email" name="email" id="email"
placeholder="Email Address">
</div>
```

Add an input for a telephone number. The input type of `tel` will help users on mobiles by offering up a telephone number friendly keyboard.

```
<div>
<label for="tel">Telephone:</label>
<input type="tel" name="tel" id="tel" placeholder="Telephone
Number">
</div>
```

For multi-line text fields use the `<textarea></textarea>` element. Again, this element should have a `name` attribute in order to create a name/value pair for data submission.

We can also add a `<label>` and `id` and `placeholder` attributes.

```
<div>
<label for="message">Message:</label>
<textarea name="message" id="message" placeholder="Your
Message"></textarea>
</div>
```

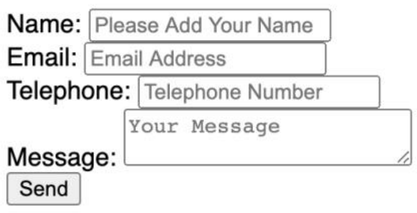
Now we need a submit button. This is an input type and is created as follows:

```
<div>
<input type="submit" value="Send">
</div>
```

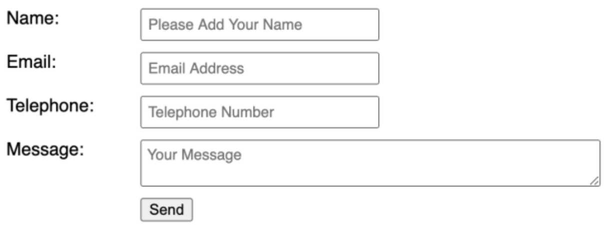
The `value` attribute is used to change the text that appears on the button.

Styling the Form

The form will initially appear with a ragged edge.



We could like it to appear as follows:



To achieve this can add some CSS to the *styles/main.css* to style the form.

First, space out the form elements vertically by creating a rule for the `<div>` in which each form element was enclosed:

```
form > div {
  margin: 10px 0;
}
```

As a `<label>` is an inline element we cannot give it a width unless we change its display type to `inline-block`. An inline-block element flows left to right but can have width and height applied.

```
label{
     display:inline-block;
     width:120px;
     vertical-align:top;
}
```

The `vertical-align:top` property is particularly useful on the label of the `<textarea>` to vertically aligned the label against it.

To style the `input type="text"` element can be targeted with an attribute selector ie:

```
input[type=text]{
     width:200px;
     padding:5px;
     font-family: Helvetica, Arial, sans-serif;
}
```

We can also resize the textarea with a standard HTML selector:

```
textarea{
     width:400px;
     padding:5px;
     font-family: Helvetica, Arial, sans-serif;
}
```

Reposition the submit button by adding a class to the `<div>`.

```
<div class="submitBtn">
     <input type="submit" value="Send">
</div>
```

Then create a rule as follows:

```
.submitBtn {
  margin-left: 120px;
}
```

Alternative Route with Flexbox

The above layout could also be achieved with flexbox.

Change the `form>div` rule to make all the `<div>` element in the form flex containers. The label and form elements will now ‘flex’.

```
form > div {
  margin: 10px 0;
  display: flex;
}
```

Change the rule on the label to set the `flex-basis` to a value of 120px.

```
label {
  flex-basis: 120px;
}
```

As the default behaviour of a flex container is to align elements to the top-left there is no need for `vertical-align`.