Module 10- Web Development

Module Overview

This module aims explaining web development concepts in detail

Module Objective

At the end of the module, you will be able,

- Understand Basics of Web Development
- Learn Basics of HTML, CSS and JavaScript

Web Development

Web Development in the most simplistic term means to create a website for the Internet.

There are plenty of websites out there and just like a painting is created by a painter, a website is essentially built by web developers and web designers.

This is a basic example of coding:

example.com

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">

    body {

        background-color:#f0f0f2;
        margin:0;padding:0;
        font-family:"Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif;
    }

    div {

            width:600px;
            margin:5em auto;
            padding:50px;
            background-color:#fff;
            border-radius:1em;
        }

    a:link,a:visited {
            color:#38488f;
            text-decoration:none;

        }

    @media (max-width:700px){body {
            background-color:#fff;
```

This looks a bit like ancient wingding language if you are a newbie, but this 'random' code is what powers a fully functional website. Take a look here for the live website of this code.

Go to any website and press Ctrl+U (for Windows) or Cmd+U (for Mac) to see what's called the 'source code' of a website.

While web developers are responsible for the coding and programming aspect of a website, the web designers design the theme, layout, images and decide the color scheme of the website, which is then used by a web developer in putting together the website.

It's highly beneficial to have a fair amount of designing knowledge like:

- Basic Photoshop skills.
- Typography (which is basically your font choice and layout of text).
- How you use colors.

What is a web page?

A web page is simply an electronic page that is a part of a book which in this case is a website.

A website is composed of several web pages. These web pages are a combination of text, images, videos and links. Web pages are often linked to each other through hyperlinks.

What is a hyperlink?

It is a link that directs you to a different page on the same website or a different website altogether.

If you move your cursor over the text it would often change to a 'finger' or sometimes the color of the text would change and an underline would appear.

What is a web browser?

Web browsers are software applications that you use to access everything on the World Wide Web (commonly known as the internet).

Examples of web browsers would be: Google Chrome, Apple Safari, Opera and Internet Explorer.

What is a search engine?

Search engines let a user locate relevant websites based on the phrases and keywords entered by the user.

Think of the World Wide Web (the internet) as a library of websites and you are looking for something specific, you then do search on a search engine.

Some common search engines are: Google, Bing and Yahoo.

What is an IP address?

An IP address is a unique numerical address that identifies every computing device like personal computers, tablets, smart phones etc. to communicate with other devices in the IP network.

It's basically a cellphone number for your laptop or like a number plate for a car.

How it works:

The server receives a request for a page sent by your web browser. The browser connects to the server using an IP address. In return, the server sends back the requested page back to your IP address.

To find your IP Address, check out:

whatismyipaddress.com

What is a domain?

A domain is your 'property' on the internet.

So for example, my 'home' is: www.studywebdevelopment.com – I own SWD, this is my domain. No two websites can have the same domain name.

This is the address people will use to visit your 'home' which is your website. An URL (Uniform Resource Locator,) carries the domain name.

A domain name consists of a top-level and a second-level domain. A top-level domain extension (TLD) is the part of the domain name located to the right of the dot (".") like .com, .net, and .org.

A TLD identifies something about the website associated with it, such as its purpose, the organization that owns it or the geographical area where it originates, like .com is mostly used for commercial purposes while .edu is used for educational institutions.

Just like you would go through a property agent to buy a property, you need to go through a hosting provider to buy a domain. I recommend Bluehost for purchasing your domain.

What is website hosting?

A web hosting provider offers a service that allows us to make our website accessible to anyone who goes to our domain on the internet.

So we need to register/purchase a domain from a hosting provider and then ALSO purchase a hosting service as well. These are two DIFFERENT things altogether. A domain is paid annually and hosting is mostly paid monthly/quarterly/yearly.

How to create a website?

You need to decide what type of website you'd like to have. Here's a simple and brief overview:

You can either use a CMS (Content Management System) which allows you to create, upload, edit and modify your content that is displayed on a website which is ideal for blogs.

It just makes things a lot easier since you don't really need coding knowledge, but it's a bonus if you do. You can choose from a variety of themes, both paid and free.

Some popular CMS's are: Wordpress, Drupal and Joomla.

If you don't want a CMS and you like to live dangerously and be adventurous (like most developers out there) then you need to start with a good text editor.

What is a text editor?

A text editor is a program where you write your code.

You can write code for HTML, CSS, JavaScript, Python, PHP, Ruby etc. It is essential that you choose a good text editor when you start. A good text editor can increase productivity and development speed considerably and it makes you more efficient.

Note– To master these text editors and to learn keyboard shortcuts, just go to YouTube and look for short tutorials.

Here are few popular text editors:

- Sublime is a really cool editor that is fast and simple. It's amazing for beginners and professionals.

- Notepad++ is very simple, friendly and clean. Some cool features are Syntax Highlighting and Syntax Folding, Search/Replace, Auto-completion and more.

- Atom-text-editor: Atom comes from Git and it's available for Mac, Windows and Linux. It is very promising with features like auto-completion, multiple panes, line numbers, multiple file support, search/replace features and more.

- Brackets is a lightweight and powerful text editor originally developed by Adobe. It comes with some really handy features like quick edit, live preview, color selector and more.

HTML

HTML is not a programming language; it is a markup language that defines the structure of your content. HTML consists of a series of elements, which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller, and so on.

For example, take the following line of content:

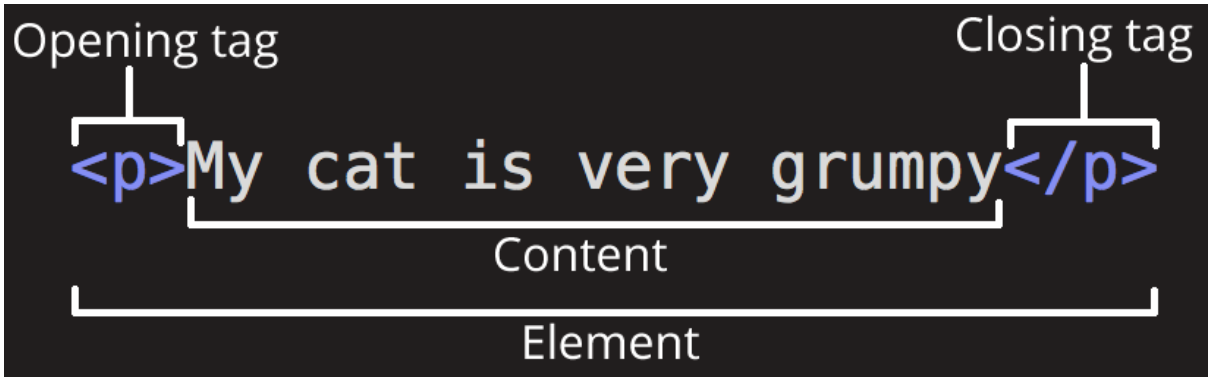
My cat is very grumpy

If we wanted the line to stand by itself, we could specify that it is a paragraph by enclosing it in paragraph tags:

** My cat is very grumpy **

A) Anatomy of an HTML element

Let's explore this paragraph element a bit further:



The main parts of our element are as follows:

- **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 — in this case where the paragraph begins.
- **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 — in this case where the paragraph ends. Failing to add a closing tag is one of the standard beginner errors and can lead to strange results.
- **The content:** This is the content of the element, which in this case is just text.
- **The element:** The opening tag, the closing tag, and the content together comprise the element.

Elements can also have attributes that look like the following:

Attributes contain extra information about the element that you don't want to appear in the actual content. Here, **class** is the attribute name, and editor-note is the attribute value. The **class** attribute allows you to give the element an identifier that can be used later to target the element with style information and other things.

An attribute should always have the following:

- A space between it and the element name (or the previous attribute, if the element already has one or more attributes).
- The attribute name, followed by an equals sign.
- The attribute value, wrapped by opening and closing quotation marks.

Note: Simple attribute values that don't contain ASCII whitespace (or any of the characters " ' ` = < >) can remain unquoted, but it is recommended that you quote all attribute values, as it makes code more consistent and understandable.

Nesting Elements

You can put elements inside other elements too — this is called **nesting**. If we wanted to state that our cat is very grumpy, we could wrap the word "very" in a **** element, which means that the word is to be strongly emphasized:

** My cat is very grumpy. **

You do however need to make sure that your elements are properly nested: in the example above, we opened the ** ** element first, then the **** element; therefore, we have to close the **** element first, then the ** ** element. The following is incorrect:

** My cat is very grumpy. **

The elements have to open and close correctly so that they are clearly inside or outside one another. If they overlap as shown above, then your web browser will try to make the best guess at what you were trying to say, which can lead to unexpected results.

Empty elements

Some elements have no content and are called empty elements. Take the <img> element that we already have in our HTML page:

<img src="images/firefox-icon.png" alt="My test image">

This contains two attributes, but there is no closing **</img>** tag and no inner content. This is because an image element doesn't wrap content to affect it. Its purpose is to embed an image in the HTML page in the place it appears.

B) Anatomy of an HTML document

That wraps up the basics of individual HTML elements, but they aren't handy on their own. Now we'll look at how individual elements are combined to form an entire HTML page. Let's revisit the code we put into our index.html example:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>My test page</title>
 </head>
 <body>
  <img src="images/firefox-icon.png" alt="My test image">
 </body>
</html>
```

Here, we have the following:

- **<!DOCTYPE html>** — The doctype. In the mists of time, when HTML was young (around 1991/92), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. However, these days no one cares about them, and they are just a historical artifact that needs to be included for everything to work right. For now, that's all you need to know.

- **<html></html>** — the **<html>** element. This element wraps all the content on the entire page and is sometimes known as the root element.

- **<head></head>** — the **<head>** element. This element acts as a container for all the stuff you want to include on the HTML page that isn't the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.

- **<meta charset="utf-8">** — This element sets the character set your document should use to UTF-8, which includes most characters from the vast majority of written languages. Essentially, it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later on.

- **<title></title>** — the **<title>** element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in. It is also used to describe the page when you bookmark/favourite it.

- **<body></body>** — the **<body>** element. This contains all the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

Images

Let's turn our attention to the **<img>** element again:

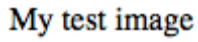
<img src="images/firefox-icon.png" alt="My test image">

As we said before, it embeds an image into our page in the position it appears. It does this via the **src** (source) attribute, which contains the path to our image file.

We have also included an alt (alternative) attribute. In this attribute, you specify descriptive text for users who cannot see the image, possibly because of the following reasons:

- They are visually impaired. Users with significant visual impairments often use tools called screen readers to read out the alt text to them.

- Something has gone wrong causing the image not to display. For example, try deliberately changing the path inside your src attribute to make it incorrect. If you save and reload the page, you should see something like this in place of the image:



The keywords for alt text are "descriptive text". The alt text you write should provide the reader with enough information to have a good idea of what the image conveys. In this example, our current text of "My test image" is no good at all. A much better alternative for our Firefox logo would be "The Firefox logo: a flaming fox surrounding the Earth."

Try coming up with some better alt text for your image now.

Marking up text

This section will cover some of the essential HTML elements you'll use for marking up the text.

Headings

Heading elements allow you to specify that certain parts of your content are headings — or subheadings. In the same way that a book has the main title, chapter titles, and subtitles, an HTML document can too. HTML contains 6 heading levels, <h1>–<h6>, although you'll commonly only use 3 to 4 at most:

<h1>My main title</h1>
<h2>My top level heading</h2>
<h3>My subheading</h3>
<h4>My sub-subheading</h4>

Now try adding a suitable title to your HTML page just above your **<img>** element.

Note: You'll see that your heading level 1 has an implicit style. Don't use heading elements to make text bigger or bold, because they are used for accessibility and other reasons such as SEO. Try to create a meaningful sequence of headings on your pages, without skipping levels.

Paragraphs

As explained above, elements are for containing paragraphs of text; you'll use these frequently when marking up regular text content:

** This is a single paragraph **

Add your sample text (you should have it from What should your website look like?) into one or a few paragraphs, placed directly below your <img> element.

Lists

A lot of the web's content is lists, and HTML has special elements for these. Marking up lists always consist of at least 2 elements. The most common list types are ordered and unordered lists:

- Unordered lists are for lists where the order of the items doesn't matter, such as a shopping list. These are wrapped in a <ul> element.

- Ordered lists are for lists where the order of the items does matter, such as a recipe. These are wrapped in an <ol> element.

Each item inside the lists is put inside an <li> (list item) element.

For example, if we wanted to turn the part of the following paragraph fragment into a list

** At Mozilla, we're a global community of technologists, thinkers, and builders working together ... **
We could modify the markup to this

** At Mozilla, we're a global community of **

<ul>
**
technologists
**
**
thinkers
**

**
builders
**
**
**

** working together ... **
Try adding an ordered or unordered list to your example page.

Links

Links are very important — they are what makes the web a web! To add a link, we need to use a simple element — <a> — "a" being the short form for "anchor". To make text within your paragraph into a link, follow these steps:

Choose some text. We chose the text "Mozilla Manifesto".

Wrap the text in an <a> element, as shown below:

<a>Mozilla Manifesto

Give the <a> element an href attribute, as shown below:

Mozilla Manifesto

Fill in the value of this attribute with the web address that you want the link to link to:

Mozilla Manifesto

You might get unexpected results if you omit the https:// or http:// part, called the protocol, at the beginning of the web address. After making a link, click it to make sure it is sending you where you wanted it to.

href might appear like a rather obscure choice for an attribute name at first. If you are having trouble remembering it, remember that it stands for hypertext reference.

CSS

CSS (Cascading Style Sheets) is what makes web pages look good and presentable. It's an essential part of modern web development, and a must-have skill for any web designer and developer.

Few ways to include CSS in our projects:

1. Inline CSS
First off, we can include CSS directly in our HTML elements. For this, we make use of the **style** attribute and then we provide properties to it.

<h1 style="color: blue"> Hello world! </h1>

Here we're giving it the property of **color**, and setting the value to blue, which results in the following:

We can also set multiple properties inside the **style** tag if we wanted. However, I don't want to continue down this path, as things start to get messey if our HTML is cluttered with lots of CSS inside it.

This is why the second method to include CSS was introduced.

2. Internal CSS
The other way to include CSS is using the style element in the head section of the HTML document. This is called internal styling.

```
<head>
  <style>
    h1 {
      color: blue;
    }
  </style>
</head>
```

In the style element, we can give the styling to our HTML elements by selecting the element(s) and provide styling attributes. Just like we applied the **color** property to the **h1** element above.

3. External CSS
The third and most recommended way to include CSS is using an external stylesheet. We create a stylesheet with a **.css** extension and include its link in the HTML document, like this:

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

```
</head>
```

In the code above, we have included the link of style.css file using the **link** element. We then write all of our CSS in a separate stylesheet called **style.css** so that it's easily manageable.

```
//style.css
h1 {
  color: blue;
}
```

This stylesheet can also be imported into other HTML files, so this is great for reusability.

CSS Selectors

As we discussed earlier, CSS is a design language which is used to style HTML elements. And in order to style elements, you first have to select them. You've already seen a glimpse of how this works, but let's dive a bit deeper into CSS selectors, and look at three different ways you can select HTML elements.

1. Element

The first way to select an HTML element is by simply using the name, which is what we did above. Let's see how it works:

```
h1 {
   font-size: 20px;
}
p {
   color: green;
}
div {
   margin: 10px;
}
```

The example above is almost self-explanatory. We are selecting different elements like **h1, p, div** and giving them different style attributes. The font-size controls the size of the text, color sets the text **color**, and **margin** adds spacing around the element.

2. Class

Another way of selecting HTML elements is by using the class attribute. In HTML, we can assign different classes to our elements. Each element can have multiple classes, and each class can also be applied to multiple elements as well.

Let's see it in action:

```
<div class='container'>
   <h1> This is heading </h1>
</div>
...
.container {
   margin: 10px;
}
```

In the code above, we have assigned the class of **container** to the div element. In the stylesheet, we select our class using **.className** format and giving it a **10px** margin.

3. ID

Like classes, we can also use IDs to select HTML elements and apply styling to them. The only difference between class and ID is that one ID can be assigned to only one HTML element.

```
<div>
   <p id='para1'> This is a paragraph </p>
</div>
...
#para1 {
   color: green;
   font-size: 16px;
}
```

The example above displays how we assign an ID to the paragraph element and later use the ID selector in the stylesheet to select the paragraph and apply the style to it.

Fonts & Colors

CSS provides us with literally hundreds of options for playing around with fonts and colors and making our HTML elements look pretty. We can choose from two types of font family names:

- **Generic Family:** a group of font families with a similar look (like 'Serif' or 'Monospace')
- **Font Family:** a specific font family (like 'Times New Roman' or 'Arial')

For colors, we can use predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

```
<div class='container'>
   <h1 class='heading1'>
      CSS is Coooooool!!!!
   </h1>
</div>
.............
.container {
   width: 500px;
```

```
  height: 100px;
  background-color: lightcyan;
  text-align: center;
}
.heading1 {
  font-family: 'Courier New';
  color: tomato;
}
```

As you can see in the above example, we have a div element with the class of **container .** Inside this div, there is an **h1** tag with some text inside it.

In the stylesheet, we select the container class and set its **width**, **height**, **background-color, and text-align**.

Finally we select the **.heading1** class — which is applied to the **h1** tag — and give it the attributes of **font-family** and **color**.

JavaScript

JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.

JavaScript itself is fairly compact yet very flexible. Developers have written a large variety of tools on top of the core JavaScript language, unlocking a vast amount of extra functionality with minimum effort. These include:

- Browser Application Programming Interfaces (APIs) — APIs built into web browsers, providing functionality like dynamically creating HTML and setting CSS styles, collecting and manipulating a video stream from the user's webcam, or generating 3D graphics and audio samples.

- Third-party APIs — Allow developers to incorporate functionality in their sites from other content providers, such as Twitter or Facebook.

- Third-party frameworks and libraries — You can apply these to your HTML to allow you to rapidly build up sites and applications.

A "hello world" example

First, go to your test site and create a new folder named 'scripts' (without the quotes). Then, within the new scripts folder you just created, create a new file called main.js. Save it in your scripts folder.

Next, in your index.html file enter the following element on a new line just before the closing </body> tag:

<script src="scripts/main.js"></script>

This is basically doing the same job as the <link> element for CSS — it applies the JavaScript to the page, so it can have an effect on the HTML (along with the CSS, and anything else on the page).

Now add the following code to the main.js file:

```
var myHeading = document.querySelector('h1');
myHeading.textContent = 'Hello world!';
```

Finally, make sure the HTML and JavaScript files are saved, then load index.html in the browser. You should see something like the following:

Note: The reason we've put the <script> element near the bottom of the HTML file is that HTML is loaded by the browser in the order it appears in the file. If the JavaScript is loaded first and it is supposed to affect the HTML below it, it might not work, as the JavaScript would be loaded before the HTML it is supposed to work on. Therefore, putting JavaScript near the bottom of the HTML page is often the best strategy.

What happened?

Your heading text has now been changed to "Hello world!" using JavaScript. You did this by first using a function called **querySelector()** to grab a reference to your heading, and store it in a variable called **myHeading**. This is very similar to what we did using CSS selectors. When wanting to do something to an element, you first need to select it.

After that, you set the value of the **myHeading** variable's **textContent** property (which represents the content of the heading) to "Hello world!".

Note: Both of the features you used above are parts of the Document Object Model (DOM) API, which allows you to manipulate documents.

Let us understand the core features of JavaScript:

Variables

Variables are containers that you can store values in. You start by declaring a variable with the var keyword, followed by any name you want to call it:

var myVariable;

Note:

A semicolon at the end of a line indicates where a statement ends; it is only absolutely required when you need to separate statements on a single line. However, some people believe that it is a good practice to put them in at the end of each statement

You can name a variable nearly anything, but there are some name restrictions (see this section about naming rules). If you are unsure, you can check your variable name to see if it is valid.

JavaScript is case sensitive — myVariable is a different variable to myvariable. If you are getting problems in your code, check the casing!

After declaring a variable, you can give it a value:

myVariable = 'Bob';

You can do both these operations on the same line if you wish:

var myVariable = 'Bob';

You can retrieve the value by just calling the variable by name:

myVariable;

After giving a variable a value, you can later choose to change it:

var myVariable = 'Bob';
myVariable = 'Steve';

Note that variables may hold values that have different data types:

Variable	Explanation	Example

String	A sequence of text known as a string. To signify that the value is a string, you must enclose it in quote marks.	var myVariable = 'Bob';
Number	A number. Numbers don't have quotes around them.	var myVariable = 10;
Boolean	A True/False value. The words true and false are special keywords in JS, and don't need quotes.	var myVariable = true;
Array	A structure that allows you to store multiple values in one single reference.	var myVariable = [1,'Bob','Steve',10]; Refer to each member of the array like this: myVariable[0], myVariable[1], etc.
Object	Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	var myVariable = document.querySelector('h1');

Comments

You can put comments into JavaScript code, just as you can in CSS:

**/*
Everything in between is a comment.
*/**
If your comment contains no line breaks, it's often easier to put it behind two slashes like this:

// This is a comment

Operators

An operator is a mathematical symbol which produces a result based on two values (or variables). In the following table you can see some of the simplest operators, along with some examples to try out in the JavaScript console.

Operator	Explanation	Symbol(s)	Example

Addition	Used to add two numbers together or glue two strings together.	+	6 + 9; "Hello " + "world!";

Subtraction, Multiplication, Division	These do what you'd expect them to do in basic math.	-, *, /	9 - 3;
			8 * 2; // multiply in JS is an asterisk
			9 / 3;

Assignment	You've seen this already: it assigns a value to a variable.	=	var myVariable = 'Bob';
Equality	Does a test to see if two values are equal to one another and returns a true/false (Boolean) result.	===	var myVariable = 3; myVariable === 4;

Not, Does-not-equal	Returns the logically opposite value of what it precedes; it turns a true into a false, etc. When it is used alongside the Equality operator, the negation operator tests whether two values are not equal.	!, !==	For "Not", the basic expression is true, but the comparison returns false because we negate it:
			
			var myVariable = 3;
			!(myVariable === 3);
			
			"Does-not-equal" gives basically the same result with different syntax. Here we are testing "is myVariable NOT equal to 3". This returns false because myVariable IS equal to 3:
			
			var myVariable = 3;
			myVariable !== 3;

Note: Mixing data types can lead to some strange results when performing calculations, so be careful that you are referring to your variables correctly, and getting the results you expect. For example, enter "35" + "25" into your console. Why don't you get the result you expected? Because the quote marks turn the numbers into strings, so you've ended up concatenating strings rather than adding numbers. If you enter, 35 + 25 you'll get the correct result.

Conditionals

Conditionals are code structures which allow you to test if an expression returns true or not, running alternative code revealed by its result. A very common form of conditionals is the **if ... else** statement. For example:

```
var iceCream = 'chocolate';
if (iceCream === 'chocolate') {
  alert('Yay, I love chocolate ice cream!');
} else {
  alert('Awwww, but chocolate is my favorite...');
}
```

The expression inside the **if (...)** is the test — this uses the identity operator (as described above) to compare the variable **iceCream** with the string chocolate to see if the two are equal. If this comparison returns true, the first block of code is run. If the comparison is not true, the first block is skipped and the second code block, after the else statement, is run instead.

Functions

Functions are a way of packaging functionality that you wish to reuse. When you need the procedure you can call a function, with the function name, instead of rewriting the entire code each time. You have already seen some uses of functions above, for example:

```
var myVariable = document.querySelector('h1');
alert('hello!');
```

These functions, **document.querySelector** and alert, are built into the browser for you to use whenever you desire.

If you see something which looks like a variable name, but has parentheses — () — after it, it is likely a function. Functions often take arguments — bits of data they need to do their job. These go inside the parentheses, separated by commas if there is more than one argument.

For example, the **alert()** function makes a pop-up box appear inside the browser window, but we need to give it a string as an argument to tell the function what to write in the pop-up box.

The good news is you can define your own functions — in this next example we write a simple function which takes two numbers as arguments and multiplies them:

```
function multiply(num1,num2) {
  var result = num1 * num2;
  return result;
}
```

Try running the above function in the console, then test with several arguments. For example:

```
multiply(4, 7);
multiply(20, 20);
multiply(0.5, 3);
```

Note: The return statement tells the browser to return the result variable out of the function so it is available to use. This is necessary because variables defined inside functions are only available inside those functions. This is called variable scoping.

Events

Real interactivity on a website needs events. These are code structures which listen for things happening in browser, running code in response. The most obvious example is the click event, which is fired by the browser when you click on something with your mouse. To demonstrate this, enter the following into your console, then click on the current webpage:

```
document.querySelector('html').onclick = function() {
   alert('Ouch! Stop poking me!');
}
```

There are many ways to attach an event to an element. Here we select the <html> element, setting its onclick handler property equal to an anonymous (i.e. nameless) function, which contains the code we want the click event to run.

Note that

```
document.querySelector('html').onclick = function() {};
```

is equivalent to

```
var myHTML = document.querySelector('html');
myHTML.onclick = function() {};
```

Trainer will take the participants to the computer lab and ask the participants to do practice on HTML commands taught in the class.

Trainer will take the participants to the computer lab and ask the participants to do practice on CSS commands taught in the class.

Trainer will take the participants to the computer lab and ask the participants to do practice on JavaScript commands taught in the class.
