What do we mean by a Script?

Well, we're not talking about a play we might be reading for English Literature; nor are we talking about the little piece of green paper the doctor gives you when you need some medicine.

As I said right at the start, HTML is a great language for formatting the text, colours, layout and so on in a Web page (I hope you've seen that for yourself by now!) but it is just what its name says, it's a mark-up language. Fab for formatting stuff on the page, but it can't handle even the most basic decisions, maths or other forms of "thinking".

So, if we want to make our Web site a little bit more fun, or a little bit "cleverer", then we need to start to explore a new language, one we can actually write little lumps of program code in, and drop these lumps into our HTML, so they happen at the right place on our page. These lumps of code are called **scripts**, and we can write them in a **scripting language**. By far the most common scripting language on the Web is a thing called **JavaScript**.

Like HTML, JavaScript (or **JS** as it's sometimes called) is as old as the World Wide Web itself – that means it first appeared in 1995. Also, like HTML, it has changed a little bit over the years.

JavaScript is cool; it enables us to write Web sites that are a little bit clever; Web sites that can actually "do" things, as well as show things and look pretty. Sites where you can buy things; weather sites that can ask for your location to tell you what the weather's going to be like near you; the bit where you can log in to a site for personal service; sites that count their number of visitors, ir show a clock … all this, and a whole load more use scripting languages. Most use JavaScript.

Some things you need to know

Variables

When we write computer programs, they have to process data. We need to put each of those pieces of data into some sort of container, with a name, so that the computer (and the programmer) knows what's what. We call those containers **variables**.

That data can be numbers that we process like we do in maths, or pieces of text, which we can handle a bit like cutting and pasting in a word processor. So, we need more than one sort of variable.

When you've been programming for a while, you'll see there are quite a few types of variable. For now, we'll just think of two: numbers which we call **numeric variables**, and text, which we call **string variables**.

Strings and things

Let's have an example. We can create two variables. Let's give them the names Sam and Jordan.

If they are **numeric variables**, we can say `Sam=21` and `Jordan=19`. If we add them together, `Sam + Jordan` will give us a value of `40`. Just like simple maths.

If they are **string variables**, then we put the value in quotes `"` and `"`. If we add these up, they behave like words rather than numbers: If we say `Sam ="21"` and `Jordan="19"` then `Sam + Jordan` will give us `"2119"`.

If we do that with words, we could say `Sam="Web"` and `Jordan="Master"` now `Sam + Jordan` gives us `"WebMaster"`.

We can change the contents of a variable at any time simply by saying its name, using the equals sign, and stating the new value – in quotes if it's a string value.

Fixed or persistent data

Fixed data or **persistent data** is stated in the code and cannot change. As with variables, persistent data can be either string or numeric.

For example, I can say `Sam = Jordan + 4`. So long as Jordan is a numeric value, `Sam` will become `4` more. The `4` is a persistent value in our code.

Statements, commands and variables

A **statement** in a program usually consists of a **command** (an instruction word) and some action to be carried-out on one or more variables. In any programming language, commands are known as **reserved words**. This means you cannot use them as names for your variables.

You'll see we're going to use two JavaScript commands later. The first is a thing called var. This lets us create a variable, and put a value inside it.

For example:

> `var sam=21 ;` will create a numeric variable called `sam`, holding the value `21`
>
> `var jordan="red house over yonder" ;` will create a string (or text) variable named `jordan` containing the text `red house over yonder`.

Syntax

Syntax is a set of rules we need to follow when we write code. When you write in English, the syntax is usually referred to as grammar. You've already met syntax in HTML – the way we use start and end tags, for example. There are also rules of syntax we have to follow in JavaScript. You'll soon see how important the `;` character is in JavaScript code, for example.

Your first script:

In the last session, we looked at the `mailto` reference within an anchor point, to put a nice, convenient link to your email address in a Web site. There is a small problem with this. Some people and organisations (from big businesses right through to hackers) use robotic software, **bots**, to trawl the Internet, searching for email addresses which they can then **spam** with what most of us call **junk mail** or junk email.

So, we need to get a little bit cleverer than those bots. The bots recognise complete email addresses, something like:

```
myfirstname.mysurname@mydomain.com
```

They're trained to look for these. However, they're not trained to figure out what a script does. So, as some Web developers say, we can outsmart the bots with a little script.

Confuse a bot

Our contact email address at Resources4Learning.org is:

```
contact@resources4learning.org
```

That email address can be accessed by a `mailto` anchor point on our Web site. It's an anchor point that works well, when a human drags the mouse-pointer over it and clicks. We're happy for genuine humans with genuine questions to email us, but we don't want spam.

To solve this problem, the first thing we do is to break that email address into several pieces. Then we write a script to put it back together, but only when a human clicks on the hyperlink. In the script, I've gone one step further, to try to outsmart even the clever bots than might try to work out my code. I have used both variable and fixed (or persistent) data.

Here's how I'm going to break that address:

- A **variable** named `id1` will contain the text characters "`con`"
- I will leave the letter "`t`" as **persistent data**
- A variable named `id2` will contain the letters "`act`"
- I will leave the "`@`" character as persistent data
- A third variable, `domain` will contain the text "`resources4learning`"
- And I'll leave the "`.org`" part as persistent data

That should be good enough to confuse a bot.

Notice all those variables are strings. This is important so the chunks of text fit back together or **concatenate** one after the other into a new, longer string.

What does the code look like?

Here's the code from the email link paragraph of the Resources4Learning.org Web site.

```
<p style="text-align: center;">
<! -- rem_no_spam.please@resources4learning.org //-->
<!--->e-mail: <!-- nospam_here@notmydomain.com // -->

<script language="javascript" type="text/javascript">

  var id1="con" ;
  var id2="act" ;
  var domain="resources4learning" ;
  document.write("<a href=\"mailto:" + id1 + "t" + id2 + "@" +
  domain + ".org" + "\">Contact us</a>") ;

</script>
</p>
```

What's happening?

Ok – let's break that chunk of code down, a line at a time, and see what's going on:

```
<p style="text-align: center;">
```

Starts a new paragraph, and tells the browser to override the paragraph style in the style sheet, aligning the contents of the paragraph to the centre of the page width.

```
<! -- rem_no_spam.please@resources4learning.org //-->
```

Remember, any HTML tag beginning `<!` Is a remark or comment tag, to be read by humans but not by the browser. If the nosey bot tries to use that email address, it'll get rejected somewhere along the line anyway.

```
<!--->e_mail: <!-- nospam_here@notmydomain.com // -->
```

The word e-mail, cunningly disguised as e_mail, will throw a few more bots off the scent. A nice little non-existent email address might have a few bots thinking they've done their job – and harvested an email address which should get an error message sent back to the bot's owner. (We can be cunning here!)

```
<script language="javascript" type="text/javascript">
```

This tells the browser that we want it to run a script, written in JavaScript.

Now we're looking at JavaScript code, not HTML …

```
var id1="con" ;
```

The `var` **command** tells JavaScript we are declaring a **variable**. `id1` is the variable name; `=` shows that we're about to put a value in, the `""` quotes show we want it to be treated as a string variable, not a numeric one, containing the characters `con`. The final `;` character is critical – this shows we have finished our `var` statement. This is part of the **syntax** of JavaScript.

```
var id2="act" ;
```

This statement works the same way, putting the string characters `act` into a variable named `id2`.

```
var domain="resources4learning" ;
```

In the same way again, this puts the string `resources4learning` into a variable named `domain`.

```
document.write("<a href=\"mailto:" + id1 + "t" + id2 + "@" +
domain + ".org" + "\">Contact us</a>") ;
```

This line's the "biggie". The `document.write` command tells JavaScript to write whatever comes in the following brackets, the variables, back into the Web page within the user's browser (so the user can see it and click on it). Since JavaScript is already using quotes as part of its syntax, we have to use that `\` character where we would normally use `""` in HTML. The `+` symbols tell JavaScript to add the variables and static data together. Since they are all strings, they will be **concatenated** (made into on longer string, remember).

What JavaScript will pass back to the HTML part of the browser here, will look like:
```
<a href="mailto:contact@resources4learning.org">Contact
us</a>
```
This will enable the person reading it to click and send us an email, if they want to. This code only actually shows our email address within the browser on the reader's PC, not to any bots crawling around the Web.

```
</script>
```

We need to say when our script code ends, so the browser knows to get its brain back into HTML mode. **Now we're back to HTML code again.**

```
</p>
```

Since we write tidy code, we always end our paragraphs. ☺

Over to you!

Try to get a similar piece of code into a page on your Web site, with a link to your email address which is "bot safe".

Please try to use different variable names rather than just `id1` and `id2`. Apart from making it your own work, rather than just a thin copy of mine, this also helps you to understand how it works.

Top tip:

Be very careful of the syntax. One missing `;` for example, or a `;` where it should not be, and your script will not work. We call that kind of error a **bug** in your code.

Likewise, be sure to get your `"` and `"` characters in the right places, close your brackets …

Phew!

That was a biggie, with a **lot** to take in.

You're getting to need a second section to that coding toolbox now – as well as HTML as a mark-up language, you're also now starting to learn some new skills in programming using JavaScript.

Take a break if you need to, before starting to get that "hidden" email link sorted on your site – and don't feel bad if you need to read the whole of this handout again. When I first started using it, there were a few things I didn't "get" first time around, and that was with previous programming experience in other languages.

Have fun, stay safe until next session.

If you're hungry for more, the w3schools Web site has some great stuff.

https://www.w3schools.com/