Coding Basics

In this module we'll start to look at coding and programming.

We'll discuss the different types of programming languages and the difference between programs and scripts.

We'll also look a quite a few examples of code and commonly used programming structures such as conditional structures, looping structures and functions.

Finally, we'll write our first 'Hello World!' script in JavaScript.

Review of Previous Module

In the previous module we took a more in depth look at how web servers work and in particular how HTTP requests and responses are made up.

We also covered how to set up a local hosting environment.

It was quite a technical module, with a lot to take on board, so please don't hesitate to ask any questions that you may have before we move on...

Topics for this module

In this module we'll start to look at coding and programming and cover the following topics:

- Introduction to coding
- Programming languages
- Statements and expressions
- Comments
- Variables and data
- Conditional structures
- Looping structures
- Functions

Introduction to Coding

Coding, or **programming**, is the process of writing a series of instructions in the form of a **script** or **program** that are then carried out, or **executed**, by a computer.

In the same way that people around the world use different spoken languages to communicate with each other, programmers can use a range of different programming languages to communicate with computers.

The language that is used for any particular program might depend on factors such as:

The operating system used by the computer
The type of application being developed
etc

Programming languages

Some example programming languages are listed below, although this is by no means an exhaustive list:

- C
- C++
- C#
- Pascal
- Fortran
- Cobol
- Java
- PHP
- Python
- Ruby
- JavaScript

Programming languages

There are a great number of spoken languages in the world but people tend to use them for the same things:

- Greet each other
- Ask questions
- Provide answers
- Describe things
- Convey emotions
- etc

Programming languages

In the same way, although there are a number of different programming languages they tend to be used to do similar things:

- Store values
- Perform calculations
- Make decisions
- Repeat a set of instructions
- Ask for input
- etc

Programs vs Scripts

We hear developers talking about programs and scripts, and programming and coding – so what do these terms mean.

First of all we can assume that the terms programming and coding usually mean the same thing – the process of creating programs and scripts.

The terms programs and scripts usually mean slightly different things though, and are related to the fact that most programming languages fall into one of two categories:

- Complied languages
- Interpreted languages

In order to understand what these mean, let’s take a look at some programming concepts...

Programs vs Scripts

Not only are there a range of programming languages that developers can use, computers themselves speak a different language again – usually referred to as machine code.

In the early days of computing programmers had to write instructions in languages that were as close as possible to the machine code used by computers.

These are known to as assembly languages, or low-level languages as they are close to the level of the computer's language.

An example of some assembly language source code is shown on the next slide.

Programs vs Scripts

```
                         ;------------------------------------------------------------
                         ; zstr_count:
                         ; Counts a zero-terminated ASCII string to determine its size
                         ; in:   eax = start address of the zero terminated string
                         ; out:  ecx = count = the length of the string

                         zstr_count:                   ; Entry point
00000030 B9FFFFFFFF          mov  ecx, -1              ; Init the loop counter, pre-decrement
                                                       ;  to compensate for the increment
                         .loop:
00000035 41                  inc  ecx                  ; Add 1 to the loop counter
00000036 803C0800            cmp  byte [eax + ecx], 0  ; Compare the value at the string's
                                                       ;  [starting memory address Plus the
                                                       ;  loop offset], to zero
0000003A 75F9                jne  .loop                ; If the memory value is not zero,
                                                       ;  then jump to the label called '.loop',
                                                       ;  otherwise continue to the next line
                         .done:
                                                       ; We don't do a final increment,
                                                       ;  because even though the count is base 1,
                                                       ;  we do not include the zero terminator in the
                                                       ;  string's length
0000003C C3                  ret                       ; Return to the calling program
```

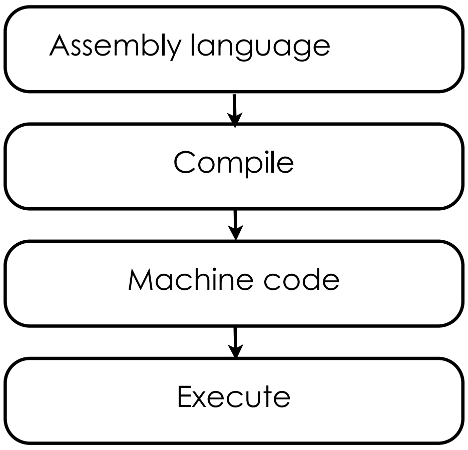
Programs vs Scripts

Although assembly code is close to the computer's own language it still can't be understood by the computer yet and it has to be translated, or **compiled**, into code that the computer can **execute.**

The final code that the computer **executes** is usually referred to as **machine code.**

The processes involved are illustrated on the next slide.

Programs vs Scripts



Programs vs Scripts

As you will have seen from the example source code listing, assembly language is not that easy to understand from a human perspective – a bit like reading Latin.

Over the years, a number of programming languages have evolved that use words and characters – referred to as a language's **syntax** – that are more familiar to humans, and therefore easier to write. These are known as **high-level languages.** Some examples are:

- C
- Java
- PHP
- Python
- JavaScript

An example of some PHP is shown on the next slide

Programs vs Scripts

Example PHP code:

```
$temperature = getTemperature();

if ($temperature > 20) {
  print “It’s hot!”;
}
else if ($temperature > 15) {
  print “It’s warm!”;
}
else if ($temperature < 10) {
  print “It’s cold!”;
}
```

As you can see, PHP's source code is much easier to understand than assembly language. Not only that, but it is very similar to other high level languages.

Programs vs Scripts

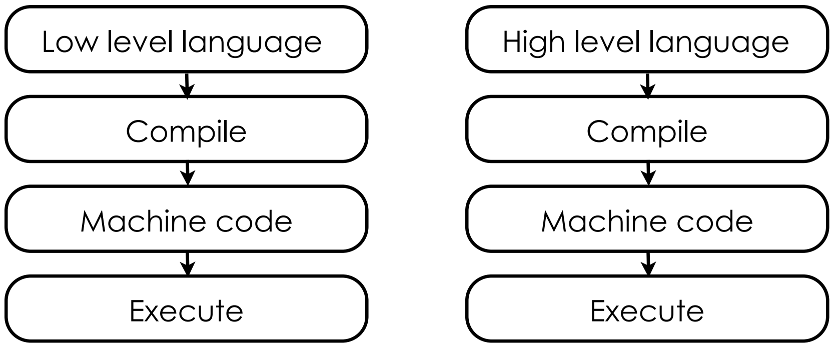
So far we've discussed two types of programming language: low-level and high level.

However, a few slides back we said that programming languages fall into two other categories: compiled and interpreted languages.

Let's look at what we mean by that:

Programs vs Scripts

Whether a program is written in a low-level language or a high-level language – at some point the code that is written by the programmer, which is known as the **source code**, has to be **compiled** in **executable code.**



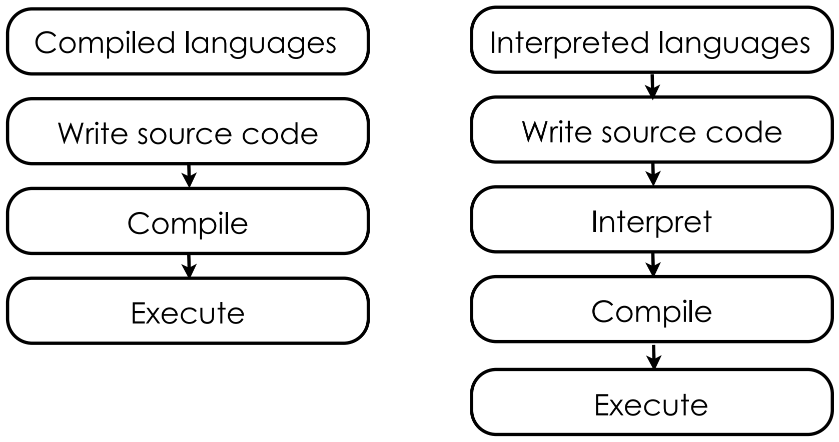
Programs vs Scripts

When we develop an application with some languages we have to go through the complete process of writing the source code and then compiling it into executable code.

We can then install the executable code on the computer and it is ready to go. **It doesn't need to be compiled again** unless we change the original source code.

With other languages we only take the process as far as writing the source code, and then we install it on the computer on which it is going to be used. A special program, known as an interpreter, then carries out the process of translating our source code into machine code for that computer. So our source code is **interpreted and complied each time the program is used**.

Programs vs Scripts



Programs vs Scripts

As a general rule, we refer to languages that must be compiled into executable code before they are used by the computer as **programming languages.**

Languages that have their source code interpreted and complied at run time are normally referred to as **scripting languages.**

Programs are therefore the final compiled code ready to be executed, and **scripts** are the saved sourced code that needs to be interpreted, compiled and then executed each time.

Programs vs Scripts

Some examples of compiled and interpreted languages are shown below:

Compiled languages	Interpreted languages
C	PHP
C#	Python
C++	Perl
Cobol	ASP
Fortran	Ruby
Java	JavaScript

Statements and Expressions

A programming statement is a single instruction to do something.

That something might be to print out some information, perform a calculation or retrieve a previously calculated value from memory.

A program is therefore made up of one or more statements.

Traditionally, the first program that a developer writes in any new programming language has a single statement that displays the words 'Hello World'.

Some examples of statements to display 'Hello World' in various languages are shown on the next slide:

Statements and Expressions

'Hello World' in various languages:

Language	Statement
PHP	print 'Hello World';
JavaScript	console.log('Hello World');
C++	cout << "Hello World";
C#	console.WriteLine("Hello World");

Statements and Expressions

Each statement is made up of a number of parts. These usually include one or more expressions.

An expression is a combination of values, operators, variables and functions that are combined and processed to produce a result.

In the example PHP statement below, the expression is the string of text **‘Hello World’**

```
print ‘Hello World’;
```

And in the following statement the expression is **2 + 4 / 5**

```
print 2 + 4 / 5;
```

Comments

In addition to statements, programming languages can include comments.

Comments are notes that are included as part of the source code but which are ignored when the program is executed. They are typically used to document the code and provide details of the author, version number, etc.

The syntax used for comments varies between programming languages, but the three most common types are as follows:

```
# Single line comment

// Single line comment

/* Multi-line comment */
```

Variables and Data

Programming languages use values to perform calculations, make decisions and control processing. These values are referred to as data.

The types of data that a particular programming language can use may vary slightly, but in general they can be categorized as follows:

Text (also known as strings)
Numbers (integers, floats)
Dates and times
Boolean values (true or false)

These values can be used in **expressions** and stored for later use in **variables.**

Variables and Data

Variables are used in a programming language to hold, or store, values (data)

Each variable is given a unique name and then a value can be stored (assigned) in it.

The value that has been assigned to a variable can be recalled for use later in the program by using the name of the variable.

Here's an example from PHP

```
$name = 'Mary';
// some other code here...
echo 'Hi ' . $name . ' how are you today?;
```

Let's look at that in more detail on the next slide

Variables and Data

Here’s that example from PHP again:

```
$name = ‘Mary’;
// some other statements here...
print ‘Hi ’ . $name . ‘ how are you today?’;
```

The first statement **assigns** (stores) the **string** (text) ‘Mary’ to a **variable** called **$name**. In PHP variable names always start with a ‘$’ character.

Then some other **statements** are **executed**

The final statement **prints** (**outputs**) an **expression** that joins (**concatenates**) three pieces of text together. First the **literal string** ‘Hi ’, then whatever text is stored in the **$name variable** and finally the **literal string** ‘ how are you today?’

Variables and Data

Given the following PHP code, what do you think the final output will be?

```
$today = ‘Monday’;
$place = ‘Italy’;
$otherPlace = ‘Spain’;


if ($today == ‘Saturday’) {
  print “It’s ” . $today . ‘ so I must be in ’ . $place;
} else {
  print ‘I must be in ’ . $otherPlace;
}
```

Conditional structures

Statements can be enclosed in structures that are used to control when a statement, or group of statements, are executed.

Structures can be used for making decisions based and a conditional expression or for repeating a one or more statements a number of times.

Structures that are used to make decisions are generally referred to as conditional structures, and those for repeating statements as looping structures or iterative structures.

The most commonly used conditional structure that is used in many programming language is the if structure.

The next slide looks at this structure in more detail.

Conditional structures – if

Most programming languages have an if structure that can be used for making decisions. The syntax may vary slightly between languages but the general format is as follows:

```
if (expression) {
  do something
}
```

The expression is evaluated to determine if it is true or false. In most languages the expression is enclosed in parenthesis.

If the expression is true then any statements enclosed in the curly braces '{ }' that follow the expression are executed. If the the expression is false then they are not.

The execution of the program then continues after the closing '}'

Conditional structures – if

The if structure can usually be extended to provide an option to execute a different statement. or set of statements, if the expression is false.:

```
if (expression) {
  do something
} else {
  do something else...
}
```

If the expression is true then any statements enclosed in the curly braces '{ }' that follow the expression are executed. If the expression is false then any statements in the curly braces following the **else keyword** are executed instead.

Looping structures

Another very common structure in programming languages is the looping structure or iterative structure.

This is used to repeat a single statement, or a group of statements, multiple times.

The two most common forms of looping structures are:

- for loops
- while loops

We'll look at each of these on the next slides...

Looping structures – for loop

The general syntax of a for loop structure is as follows:

```
for ( statement A; expression B; statement C ) {
  do something
}
```

When the program first encounters the for loop it executes statement A – this is only executed once at the start of the loop.

Then the program will evaluate expression B. If it is true then any statements between the curly brackets will be executed. The program then executes statement C.

Finally, the program will go back to the start of the structure and see if expression B is still true. If so it will repeat the process. This continues until expression B is false

Looping structures – while loop

The general syntax of a while loop structure is as follows:

```
while ( expression ) {
  do something
}
```

The program first evaluates the expression.

If it is true then any statements between the curly brackets will be executed.

Once any statements have been executed the program will evaluate the expression again. If it is still true then the statements will be repeated. This continues until the expression is false.

Functions

It will often be the case when writing a program that certain groups of statements are used in a number of different places in the code.

```
$name = ‘Mary’;
$age = 25;
if ($age >= 21) {
  print ‘Hi, ’ . $name . ‘ come and buy a drink!’;
} else {
  print ‘Sorry, ’ $name . ‘ come back when you are 21...’;
}
$name = ‘Mike’;
$age = 17;
if ($age >= 21) {
  print ‘Hi, ’ . $name . ‘ come and buy a drink!’;
} else {
  print ‘Sorry, ’ $name . ‘ come back when you are 21...’;
}
```

Functions

In situations where a group, or block, of code is being used in several places in a program it can be useful to incorporate that code into a function.

A function acts as a container for one or more statements. The statement can then be executed by using the name of the function to call it.

In addition, most programming languages allow values to be passed in to a function each time it is called.

Let’s rewrite that code from the previous slide using a function...

Functions

```
function checkAge($name, $age) {

  if ($age >= 21) {
    print 'Hi, ' . $name . ' come and buy a drink!';
  } else {
    print 'Sorry, ' $name . ' come back when you are 21...';
  }

}

checkAge('Mary', 25);

// other statements here...

checkAge('Mike', 17);
```

In this example the code is stored in a function. When the program encounters the function's name further down it executes the code and uses the values that are passed in.

Exercise – Hello World

OK, now it's time to write our first bit of code. We're going to use JavaScript as that is what we'll be concentrating on for the remaining modules.

JavaScript's syntax is very similar to the examples that we have already looked at, so it should be pretty easy to get started.

Create a new HTML document named 'hello-world.html' and use the following for the body of the document:

```
<body>
  <script type="text/javascript">
    document.write('Hello World!');
  </script>
</body>
```

Then open the page in a browser...

Module Summary

In this module we looked at some additional CSS selectors, such as the first-child pseudo class and also at attribute selectors.

We discussed inline and block-level elements, the CSS box model, borders and outlines and padding and margins.

We also covered specificity and media types.

Labs

Your instructor will upload the labs for this module to Basecamp after the session ends.

Questions

Do you have any questions before we move on to the next module

?

Don't worry, if you think of something later then just ask!