

Title:	Basic Programming Concepts
Level:	Entry Level 3
Credit value:	3
GLH:	27
Unique Reference Number:	A/651/8486
Sector Subject Area:	14.1: Foundations for Learning and Life.
Aim:	This unit introduces learners to the basic ideas behind programming and how computer programs work. Learners will explore common programming concepts using text-based or block-based environments, recognising how code is structured and how simple changes can alter outcomes. The unit supports confidence in logical thinking, problem solving and creative digital activity.
Assessment Type:	Assessment of this unit will be through an internally set and internally assessed portfolio of evidence.
Assessment Guidance:	Assessment decisions for skills-based learning outcomes must be made during the learner's normal work activity. Skills-based assessment must include direct observation as the main source of evidence and must be carried out over an appropriate period of time.

Learning outcomes	
<i>The learner will:</i>	
1.	Understand key terms and concepts used in programming.
Delivery content:	
The aim of this learning outcome is to provide the learners with the knowledge and skills to identify and describe common programming concepts used when creating digital solutions.	
The learner must:	
1.1 Identify common data types used in programming.	
1.2 Describe what a variable is and how it is used.	
1.3 Identify common operators used in programming.	
2.	Know how to use basic programming structures to achieve simple outcomes.
Delivery content:	

The aim of this learning outcome is to provide the learners with the knowledge and skills to use simple programming structures such as loops and arrays to produce an outcome.

The learner must demonstrate that they can:

- 2.1 Follow given instructions to **create or modify** short sections of code.
- 2.2 Use **loops** to repeat actions or processes.
- 2.3 Use or edit an **array** to store and retrieve data.

3 Understand how to identify and fix simple errors in code.

Delivery content:

The aim of this learning outcome is to provide the learners with the knowledge and skills to diagnose and correct errors within short code examples.

The learner must demonstrate that they can:

- 3.1 Identify examples of **syntax or logic errors** in short sections of code.
- 3.2 Correct a simple **error** following guidance.
- 3.3 Test that the corrected code performs as expected.

Scope of Training

The Scope of Training identifies areas that must be covered during the delivery of this unit. This is the minimum that is expected but tutors are expected to include other areas, knowledge of which will benefit their learners, based on location, types of work available and from the tutors own professional experience.

Requirements

Data Types:

Definition:

Categories of data a program can process (for example: string/text, integer/whole number, float/decimal, Boolean/true–false).

Teaching must include:

- Recognising what each data type represents.
- Matching simple values to suitable data types.
- Identifying data types in short code examples.

Teaching could include:

	<ul style="list-style-type: none"> • Showing type-related mistakes (for example, adding text to a number). • Displaying how types appear in block-based and text-based code.
Variables:	<p>Definition: A named area in a program that stores a value which can change while the program runs.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Finding and naming variables in short code. • Showing how a variable is set, read, and changed. • Linking variables to simple outputs (for example, score, attempts). <p>Teaching could include:</p> <ul style="list-style-type: none"> • Demonstrating variables in a block-based editor (for example, Scratch). • Tracing how a variable changes step by step.
Operators:	<p>Definition: Symbols or words that perform actions in expressions (for example: +, -, *, / for arithmetic; >, <, == for comparison; AND/OR/NOT for logic).</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Using arithmetic operators in simple calculations. • Using comparison operators to make true/false results. • Noticing how operators combine with variables and data types. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Exploring logical operators to control simple decisions. • Comparing operator blocks with text equivalents.
Create or modify:	<p>Definition: Producing a short new snippet of code or changing an existing snippet to achieve a simple stated outcome.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Following step-by-step instructions to complete a snippet.

	<ul style="list-style-type: none"> • Changing values, operators, or sequence to alter behaviour. • Completing partially written code to meet a given outcome. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Using a block-based editor to add, remove, or replace blocks (for example, Scratch, MakeCode). • Renaming variables or tidying layout for readability.
Loops:	<p>Definition: Structures that repeat one or more instructions a set number of times or while a condition is true.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Identifying a loop in short code examples. • Running or editing a simple count-controlled loop. • Explaining in plain language what the loop repeats. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Demonstrating condition-controlled loops. • Comparing loop blocks with text equivalents.
Array:	<p>Definition: An ordered collection that stores multiple values under one name, accessed by position (index).</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Identifying an array in short code examples. • Retrieving an item from an array by its position. • Changing or adding an item in a simple array. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Iterating through an array with a loop to produce an output. • Showing array lists in block-based environments.
Syntax:	<p>Definition: The rules of how code must be written so a computer can read it (for example: punctuation, brackets, indentation where required).</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Spotting common syntax errors in short snippets.

	<ul style="list-style-type: none"> • Correcting missing or extra characters (for example, quotes, brackets). • Re-running code after corrections to confirm it works. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Using editor messages to locate syntax problems. • Showing how block-based tools prevent some syntax errors.
<p>Logic errors:</p>	<p>Definition: Errors where the code runs but gives the wrong result because the steps or conditions are incorrect.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Noticing when outputs are unexpected. • Tracing steps to find where the logic goes wrong. • Changing the logic to produce the intended result. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Comparing two versions (wrong vs corrected) and describing the difference. • Adding simple print/output checks to trace program flow.
<p>Error:</p>	<p>Definition: A problem in code that stops it running correctly or makes it produce an unintended result (including syntax and logic errors).</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Distinguishing between syntax and logic errors at a basic level. • Applying a simple fix with guidance. • Testing after a fix to confirm the outcome is correct. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Keeping a short checklist for debugging steps. • Pair-checking code to find and fix small mistakes.