

Title:	Programming for Beginners
Level:	1
Credit value:	3
GLH:	27
Unique Reference Number:	H/651/8514
Sector Subject Area:	14.1: Foundations for Learning and Life.
Aim:	<p>The aim of this unit is to provide learners with the knowledge and skills to use beginner-friendly programming tools to create, test, and improve a simple program.</p> <p>Learners will identify basic programming ideas, follow guided steps to build and run a program, find and correct simple issues, and recognise how programming supports problem-solving in everyday life, work, and learning.</p>
Assessment Type:	Assessment of this unit will be through an internally set and internally assessed portfolio of evidence.
Assessment Guidance:	<p>Assessment decisions for skills-based learning outcomes must be made during normal learning or work activity.</p> <p>Evidence may include direct observation, screenshots, discussion notes, shared online files, or simple reflective statements.</p> <p>Evidence must confirm that the learner has created, tested, and improved a simple program using a beginner-level environment.</p>

Learning outcomes

The learner will:

1. Understand basic ideas used in simple programming.

Delivery content:

The aim of this learning outcome is to provide learners with the knowledge and skills to recognise key concepts used in beginner-level programming.

The learner must:

<p>1.1 Identify at least two programming concepts.</p> <p>1.2 Describe one use for each concept identified.</p> <p>1.3 Identify one benefit of using programming to complete simple tasks.</p>
<p>2. Be able to create and test a simple program.</p> <p>Delivery content:</p> <p>The aim of this learning outcome is to provide learners with the knowledge and skills to follow steps to create and test a short program using beginner-friendly tools.</p> <p>The learner must demonstrate that they can:</p> <p>2.1 Create a simple program using a block-based or beginner coding tool.</p> <p>2.2 Run the program to test its behaviour.</p> <p>2.3 Identify one issue found during testing.</p> <p>2.4 Make one basic change to improve the program.</p>
<p>3. Know how simple programming supports problem-solving.</p> <p>Delivery content:</p> <p>The aim of this learning outcome is to provide learners with the knowledge and skills to recognise how programming helps to solve simple problems.</p> <p>The learner must:</p> <p>3.1 Identify one everyday problem programming can help with.</p> <p>3.2 Describe one change made during the improvement process and its effect.</p> <p>3.3 Identify one future use of programming skills in learning, work, or personal life.</p>

<p>Scope of Training</p> <p>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.</p>	
<p>Requirements</p>	
<p>Programming concepts:</p>	<p>Definition: Simple ideas used to create a program.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> Identifying at least two concepts such as sequence, input, output, repeat, or events.

	<ul style="list-style-type: none"> • Recognising how these concepts affect what a program does. • Using very simple examples appropriate for Level 1. • Connecting concepts to everyday problem-solving. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Using block-based coding tools such as Scratch or beginner mobile apps. • Demonstrating how changing one block affects the program. • Comparing different examples that use the same concept. • Looking at real-life examples of programming concepts (e.g., daily routines).
<p>Use:</p>	<p>Definition: A simple explanation of what the concept helps the program do.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Describing the purpose of each concept identified. • Linking concepts to actions in the program. • Using clear, accessible language. • Connecting use to realistic tasks. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Short examples such as repeating a sound, moving an object, or responding to a button press. • Drawing parallels between coding concepts and everyday processes. • Comparing different uses of the same concept. • Exploring how the concept helps solve a small problem.
<p>Benefit:</p>	<p>Definition: A positive outcome of using programming.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Identifying one benefit such as organisation, problem-solving, creativity, or efficiency. • Linking benefits to everyday or work-related tasks. • Recognising that simple programming builds digital confidence. • Using examples appropriate to the learner’s context. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Using training or simulation examples.

	<ul style="list-style-type: none"> • Discussing the benefits for personal hobbies. • Showing how programming supports other digital skills. • Comparing benefits of different beginner tools.
Simple program:	<p>Definition: A short program created using beginner-friendly tools.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Creating a program with only a few steps or blocks. • Using tools appropriate for Level 1 learners. • Ensuring the program has a clear action when run. • Making the program complete before testing. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Programs that move an object, play a sound, or react to user input. • Using templates or guided tasks. • Practising small parts before building the final program. • Exploring different beginner programming environments.
Program:	<p>Definition: The working version of the code created.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Running the program to see what it does. • Checking whether it behaves as expected. • Identifying when something needs changing. • Understanding that running is part of checking. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Demonstrating programs that behave differently from expected. • Comparing program output before and after edits. • Discussing why running programs is important in real workplaces. • Testing programs with peers.
Issue found:	<p>Definition: Any problem or unexpected behaviour identified during testing.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Identifying one clear issue.

	<ul style="list-style-type: none"> • Understanding how the issue affects the program. • Recognising that issues can be simple to fix. • Keeping examples appropriate for Level 1. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Common beginner issues (e.g., missing blocks, wrong values). • Tutor demonstrations of simple debugging. • Peer identification of issues. • Reflective discussions on problem-solving.
<p>Basic change:</p>	<p>Definition: A small update made to improve how the program works.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Making at least one simple edit such as adding a block, changing a value, or adjusting timing. • Checking whether the change improves the program. • Correcting errors where needed. • Keeping changes simple and appropriate for Level 1. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Before/after comparisons. • Editing programs created from templates. • Using peer feedback to identify changes. • Exploring how small changes affect the whole program.
<p>Everyday problem:</p>	<p>Definition: A simple task or challenge that programming could help with.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Identifying one everyday problem such as reminders, simple automation, or games. • Connecting problems to digital solutions. • Recognising that programming helps complete tasks systematically. • Understanding that problems vary by context. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Examples such as automating simple steps, checking inputs, or repeating actions.

	<ul style="list-style-type: none"> • Linking to workplace tasks such as checking totals or following steps. • Discussing common problems solved with simple digital logic. • Exploring examples learners might see around them.
<p>Change made:</p>	<p>Definition: The improvement added and its effect.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Describing the change in clear language. • Linking the change to its effect in the program. • Recognising how the change improved the behaviour. • Keeping explanations simple and accessible. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Visual comparisons. • Peer explanations. • Discussions on how improvements help users. • Exploring different improvement options.
<p>Future use:</p>	<p>Definition: A way the learner may use programming skills in the future.</p> <p>Teaching must include:</p> <ul style="list-style-type: none"> • Identifying one realistic future use. • Linking the future use to learning, work, or personal hobbies. • Recognising that programming skills develop over time. • Keeping suggestions achievable and relevant. <p>Teaching could include:</p> <ul style="list-style-type: none"> • Examples such as creating simple digital content, customising apps, or progressing to more advanced coding. • Discussing careers where problem-solving is important. • Exploring how programming supports confidence with technology. • Quick reflection activities on next steps.



PART OF **nocn** GROUP

© NOCN November 25