GCSE

WJEC GCSE Computer Science

Approved by Qualifications Wales

Guidance for Teaching: Unit 2

Teaching from 2025
For award from 2027

This Qualifications Wales regulated qualification is not available to centres in England.

Made for Wales.
Ready for the world.

WJEC would like to thank the students and teachers of Fitzalan High School, Cardiff and Llanwern High School, Newport for their participation in the creation of these covers.

Contents

Introduction

The WJEC GCSE Computer Science has been approved by Qualifications Wales and is available to all centres in Wales. It will be awarded for the first time in Summer 2027, using grades A* to G.

Aims of the Guidance for Teaching

The principal aim of the Guidance for Teaching is to support teachers in the delivery of WJEC GCSE Computer Science and to offer guidance on the requirements of the qualification and the assessment process. The Guidance for Teaching is **not intended as a comprehensive reference**, but as support for teachers to develop stimulating and exciting courses tailored to the needs and skills of their learners. The guide offers possible classroom activities and links to useful resources (including our own, freely available digital materials and some from external sources) to provide ideas for immersive and engaging lessons.

Additional ways that WJEC can offer support:

- sample assessment materials and mark schemes
- professional learning events
- examiners' reports on each unit
- direct access to the subject officer
- free online resources
- Exam Results Analysis
- Online Examination Review.

Qualification Structure

WJEC GCSE Computer Science consists of two units. The qualification is unitised and does not contain tiering. There is no hierarchy to the order the units should be taught.

	Unit title	Type of Assessment	Weighting
Unit 1	Understanding Computer Science	Digital examination	50%
Unit 2	Computer Programming	On-screen examination based on a pre-released brief	50%

Assessment

Summary of Assessment Unit 2

Unit 2: Computer Programming **On-screen examination based on a pre-released brief*:* 2 hours** **50% of qualification** **80 marks**
Set and marked by WJEC. The pre-released brief will include a scenario. A new scenario will be set by WJEC each year. The pre-released brief will be available via the WJEC Portal. The assessment will feature tasks based on the scenario.

Overview of Unit 2

Computer Programming
(50% of the qualification)

The purpose of this unit is to:
- explore the concept of programming
- develop programming skills using Python as the specified language
- encourage iterative problem solving and design
- develop the use of data modelling skills
- give learners the opportunity to build appropriate user interfaces.

The unit will be based on the following:
- investigation
  - decomposition
  - abstraction
  - pattern recognition
- design
  - algorithms
  - data modelling
- implementation
  - programming
  - user interfaces
- testing
- refinement, including evaluation.

Learners will require access to Python 3. The version of Python 3 required will be specified on the scenario issued to learners and on the examination instructions.

The materials will be released, written using the standard Python IDLE IDE and will use TKINTER library for graphical user elements.

Centres may use an IDE of their choice for teaching programming knowledge and skills. However, learners are required to use the Python IDLE IDE version specified in the pre-release for the Unit 2 examination.

In this unit, learners will develop knowledge, skills and understanding in:	
2.1	Investigation
2.2	Design
2.3	Implementation
2.4	Testing, refinement, evaluation

Unit 2 Assessment objectives and weightings

AO1	Demonstrate knowledge and understanding of the key concepts and principles of computer science.	-
AO2	Apply knowledge and understanding of key concepts and principles of computer science.	25%
AO3	Analyse problems in computational terms: • to make reasoned judgements • to design, program, evaluate, and refine solutions.	25%

Unit 2 Teacher Guidance

2.1 Investigation		
Content Amplification		**Teacher Guidance**
2.1.1 Investigation	• use a systematic approach to problem solving including the use of decomposition and abstraction	Decomposition involves breaking down a complex problem into smaller, more manageable parts. Abstraction simplifies a problem by removing unnecessary details.
	• use abstraction effectively to model selected aspects of the external world in an algorithm or program	Learners should apply abstraction to remove irrelevant information from a real-world scenario and model the simplified version into an algorithm or program. It is advantageous for abstraction to occur before decomposition.
	• use abstraction effectively to appropriately structure programs into modular parts with clear, well-documented interfaces	Abstraction allows for implementation of a structured and efficient solution. Modular components of programs should be implemented with self-contained subroutines. Interfaces should be clear and unambiguous. When documenting interfaces, all documentation should be as unambiguous as possible.
	• use data modelling skills to determine requirements	Learners should be able to understand the requirements of the program, including data, relationships and constraints. Leaners should be able to determine the data which is required to be input, processed and output from the requirements along with the most appropriate methods of storing the data.
	• analyse a set of requirements.	The pre-released brief will include a scenario. A new scenario will be set by WJEC each year. This will form context of the examination and will feature tasks based on the scenario. From these requirements learners should understand the requirements, constraints and assumptions of the scenario.
	• consider a programming solution that meets a set of requirements.	Centres must ensure to use only the specified version of Python found within the task. The same version will be used in the examination. Learners should create their own solution to the problem to ensure they are familiar with the requirements of the unit. To meet requirements learners should be able to understand the requirements, create success criteria and be able to program solutions to these criteria.

2.2 Design		
Content Amplification		**Teacher Guidance**
2.2.1 Design	• design and document the input and output facilities required to produce an effective user interface	Learners should consider effective use of any GUI designs in preparation of implementing a user interface. Understanding the constraints of the Tkinter library for input and output facilities.
	• design and document all required data structures	Learners should be able to design suitable data structures, such as lists, tuples, and data dictionaries. Learners should know which is most appropriate to use.

	• using pseudo code, design and document the following routines:
	  • validation and verification
	  • data handling and processing
	  • authentication
	  • processing
	  • User interface designs
		Learners should be familiar with the use of pseudo code and must refrain from using any Python source code in its place.
		
		Learners should know how to design and use a range of appropriate validation and verification techniques including:
		presence check
		range check
		length check
		format check
		lookup table
		check digit
		double keying/double entry.
		
		Learners should be able to design suitable authentication techniques to ensure access to correctly authorised users only and to deny unauthorised users.

	• create and modify data validation and verification routines.	Learners should be able to read and understand pre-exiting validation and verification routines. They should also be able to adjust or develop routines which ensure accuracy, consistency and integrity before data is processed.

2.3 Implementation		
Content Amplification		**Teacher Guidance**
2.3.1 Implementation	Design, write, test and refine Python 3 code using the following skills:	Centres and Learners must ensure to use only the specified version of Python found within the task. The same Python IDLE IDE version will be used in the examination. Learners will be supplied with a working unannotated program based on the initial task. They will be required to annotate, add and amend functionality to this program during the examination. Learners should be familiar with basic Python library functions such as Tkinter.
	• create new and extend existing functions or methods	Functions should be small, focused, self-contained and perform a single task. These are general-purpose. Learners should utilise many of the in-built functions available to Python3. Functions should be standalone blocks of code and not associated with any object or class. Methods are functions associated or used with objects or classes, with the data stored in the object.
	• create new, edit and extend existing Python 3 modules	Modules are files containing functions, classes and variables for organisation intended to be used multiple times.
	• use variables, operators, inputs, outputs and assignment	Assign, identify and use variables to store data that can change or be amended. Learners should understand where constants and variables are appropriate and use appropriately. Learners should also be able to create self-documenting code using sensible variable and constant names.
	• use a variety of data types, including integer, Boolean, real, character and string	my_integer = 6 my_boolean = True my_real= 6.02 my_character = ‘W’

		my_string = 'WJEC'

	• use programming constructs to control the flow of a program, including:
	  • iteration (conditional and count controlled loops)
	  • selection
	  • sequence
		Iteration (Loops):
		• *For* : Used to repeatedly execute a block of code for a given number of times. The iterations can be set manually, or taken from an input, or from a list, etc.)
		• *While*: Used to repeatedly execute a block of code as long as a specified condition remains True.
		• Iteration loops a set of instructions a number of times and must be eventually terminated. Counts and rogue values can be used to terminate iteration loops.
		
		Selection:
		• Use of *If* , *Else* , *Elif* , *Switch* Statements to allow for making decisions or executing code based on certain conditions.
		• A question is asked, and depending on the answer, the program takes one or more courses of action.
		
		Sequence:
		• An action, or event, leads to the next ordered action in a predetermined order. Allows for storing and manipulating collections of data.

	• use basic file handling, including:
	  • open a file
	  • read from a file into a variable
	  • read from a file into a list (which function similarly to arrays in other programming languages)
	  • write to a file
	  • write to a file from a list (which function similarly to arrays in other programming languages)
	  • close a file
		Learners should be able to use basic file handling including opening, reading, writing, amending and closing files.
		
		Learners should also be able to create, save and retrieve appropriate data structures for the given scenario.

	• create new and extend data structures and fixed length records to store data	
	• use string manipulation	String manipulation means to modify or gain information from a string. For example: add, delete or edit a string, e.g. upper, lower, capitalise. Learners should also be able to find data within a string and count the occurrences of a given value within a string.
	• create new and extend lists, tuples and dictionaries	**Lists / Multi-Dimensional Lists**: Python allows different datatypes in the same list, e.g. int, string, real, list. Each element (item) in the list can be accessed by the number associated with its position (its index). By default, the index starts at 0. Lists are ordered (in the order created) and can be changed (mutable). Lists are written using square brackets '[ ]' **Tuples**: Python allows different datatypes in the same tuple, e.g. int, string, float, list. Each element (item) in the tuple can be accessed by the number associated with its position (its index). By default, the index starts at 0. Tuples are ordered (cannot change once created) and NOT changeable (immutable). Tuples are written using brackets '( )'. **Dictionaries**: Dictionaries in Python are used to store data in 'key:value' pairs. Dictionaries are ordered and changeable. They do not allow for duplicates. Dictionaries are written using curly brackets '{ }'
	• use lists (which function similarly to arrays in other programming languages)	Lists/Multi-Dimensional Lists: Python allows different datatypes in the same list, e.g. int, string, real, list. Each element (item) in the list can be accessed by the number associated with its position (its index).
	• use slicing	Slicing refers to the process of extracting a portion of a sequence (like a list, tuple, or string) by specifying a range of indices. Slicing allows for the creation of a new sequence but leaving the original sequence unmodified.
	• use mathematical and logical operations	Learners should be familiar with Mathematical Operations such as addition *(+)*, subtraction *(-)*, division *(/),* multiplication (*) and modulus (MOD). Learners should be familiar with logical operations such as *AND*, *OR*, *NOT* and *XOR*.

	• create new and modify existing intuitive Graphical User Interfaces (GUI) using the Python 3 built in Tkinter libraries, including:
	  • windows
	  • frames
	  • widgets
	  • buttons
	  • geometry managers
		Tkinter needs to be imported to be used within a program. Learners should be able to create windows, with suitable titles and geometry.
		
		Windows are top-level containers that hold all other frames and widgets within it.
		
		Frames provide a container that can hold widgets.
		
		Widgets provide a visual representation of data and can be items such as buttons, labels, text or checkboxes allowing for data input or modification.
		
		Buttons allow for interaction by a user in triggering an action when clicked.
		
		Geometry managers define how widgets are positioned within the main window. The three most common methods are *Pack, Grid,* and *Place.*

	• create and modify authentication management routines	Authentication involves an act of confirmation or verification before proceeding. For example, comparing user input credentials against stored credentials to determine validity.
	• create code for the solution that is self-documenting and uses meaningful identifiers	Learners should implement and produce code that is clear, concise and easy to understand without need for additional comments. Appropriate programming conventions and practices should be followed with suitable names given to variables, functions, classes etc. that accurately reflect their purpose.
	• use a programming style that is consistent, including indentation and appropriate use of white space	Learners should use indentation and white space to visually represent the structure of code. Indentation is a requirement in Python to indent a block of code. Similar to the use of curly brackets { and } in other programming languages. Indents are typically found upon beginning functions, loops and statements. Indentation and use of white space improves and ensures readability and maintainability of code.

	• use subroutines with well-defined interfaces	Learners should ensure that all input, output and behaviours are clearly specified and understood.
	• annotate code so that it is accessible to a competent third party.	Learners should use annotation and comments for additional commentary, understanding or explanation for another user or developer who is not familiar with the code. Annotation conventions state that developer comments should either be on the line before, after or at the end of the line of code in which is commented.

2.4 Testing, refinement, evaluation		
Content Amplification		**Teacher Guidance**
2.4.1 Testing	• design and document an effective testing strategy that will ensure that the final solution meets the requirements	Learners should design suitable test cases to ensure that the program functions as intended, producing expected results.
	• design test data to include examples of typical, extreme and erroneous content	Typical data represents a suitable and expected input or data type. Extreme data represents a limit or boundary of an expected input range. Erroneous data represents an unsuitable or an unexpected input.
	• implement a test plan using typical, extreme and erroneous data where appropriate	The following example when inputting an integer between 1 and 32768: • Typical data could be 16184 • Extreme data could be 1 or 32768 • Erroneous data could be *A* or @
	• present test outcomes with detailed and informed commentaries	Learners should document test case outcomes giving informed commentaries on their why the tests are success or failures.
	• demonstrate testing and refinement of code during development or in response to change in the requirements provided	Learners should document the ongoing process of testing and improving code in response to any changes.
	• explain the outcome of testing using given data or data that the learner has designed.	Learners should document expected results against actual results, providing analysis and accuracy of the provided test data.

2.4.2 Refinement and evaluation	• demonstrate understanding of the requirements and any changes required by revised requirements • demonstrate understanding of changes by presenting evidence of developmental changes • demonstrate testing and refinement of code to improve efficiency • discuss the strengths and weaknesses of differing approaches to meeting the requirements • demonstrate an understanding of the technical terminology/concepts used during software development • evaluate the outcomes of the original given code • evaluate the outcomes of the changes made by additional requirements • explain the solution or changes made to a solution.	Learners will be required to; clearly express the initial requirements for the software development project, identify and explain any changes in the requirements that were introduced during the development process. They will also be required to discuss how these changes impacted the overall design and implementation of the software. To demonstrate understanding of changes by presenting evidence of developmental changes, learners will need to explain the reasoning behind the changes made and how they addressed the revised requirements to improve the software's functionality. Learners need to be able to describe the testing strategies and techniques used to evaluate the code's performance and identify areas for improvement. They will need to be able to provide or annotate examples of changes made to the code discussing the impact of these refinements on the program. Learners should be able to discuss the strengths and weaknesses of differing approaches to meeting the requirements, explaining alternative design or implementation approaches that were considered during the development process giving an analysis of the advantages and disadvantages of each approach explaining and critically evaluating the reasoning behind the chosen solution. Learners need to demonstrate an understanding of the technical terminology/concepts used during software development by using appropriate technical terminology and concepts related to programming, and computer science. They should be able to explain the relevance and application of these concepts within the context of the scenario. To evaluate the outcomes of the original given code, learners need to assess the performance, functionality, and overall effectiveness of the initial solution identifying limitations, issues, or areas for improvement in the original code.

		Leaners need to evaluate the outcomes of the changes made by additional requirements by assessing the impact of the changes made to address the updated requirements. They are required to analyse the improvements of the introduced code in the updated solution. And provide an evaluation of the final solution. Learners should explain the solution, or changes made to a solution, by clearly communicating the final solution and the rationale behind the decisions they made. They need to describe the specific changes or refinements made to the original solution to address the updated requirements and demonstrate their ability to communicate the technical details and reasoning behind their solution.

Clarification on working with others in Unit 2

When working on the pre-release, learners can work in groups, pairs or individually to increase their understanding, knowledge and skills in programming. However, the examination will require learners to answer the questions individually.

Learning Experiences

Learners should be encouraged to consider the following learning experiences and skills to further develop their understanding, appreciation and awareness of the subject content. Information in the table below provides opportunities for teachers to integrate the learning experiences into delivery.

Learning Experience	Exemplification of Learning Experience
undertake 'unplugged' learning and development activities to help bring to life computational concepts	We will provide teachers with a number of suggested 'unplugged' activities in the Guidance for Teachers including Inside the CPU: Unplugged' and a card activity to introduce the concept of data types and their characteristics in relation to Computer Science. We have changed 'visualise' to 'bring to life' to ensure no bias against blind or visually impaired Learners.
engage with examples and achievements in programming that have provided solutions to real-world problems	Learners will have the opportunity to undertake this throughout the specification as detailed in the content amplification rationale, for example throughout Unit 2; we will also include examples in the Guidance for Teaching to ensure a diversity of representation, for example Sophie Wilson who co-designed the instruction set for the ARM architecture and Dr Gladys West was a pioneer in the early stages of GPS development.
engage in discussions about the potential impacts of the latest Artificial Intelligence (AI) tools and technologies, and developments in this field	Learners will have the opportunity to undertake this throughout the specification including 1.2.3 and 1.5.2. Further guidance will be provided in the Guidance for Teaching including discussion starters relating to bias in AI regarding race and gender.
use design-thinking to understand how a range of approaches could solve Computer Science problems	Learners will have the opportunity to undertake this throughout the specification including in Unit 2; further guidance will be provided in the Guidance for Teaching, explaining what design thinking is and how the non-linear, iterative process can support learners in this unit.
develop collaboration and teamwork skills by working with, and learning from, others	Learners will have the opportunity to develop their collaborative and teamwork skills throughout their course of study and particularly in Unit 2 as they work together to analyse Computer Science problems, design, implement, test and refine effective programs.
gain awareness and appreciation of some of the different careers and work-related areas that draw upon Computer Science	Learners will have the opportunity to undertake this throughout the specification including 1.3.1, 1.7.1 and 1.8.1 and Unit 2. Further guidance will be provided in the Guidance for Teaching and will include links to organisations such as Technocamps, and the careers pages of organisations that support a diversity of representation in Computer Science careers.

Opportunities for embedding elements of the Curriculum for Wales

<table>
<tr><th colspan="4">Curriculum for Wales Strands</th></tr>
<tr><td colspan="4">Cross-cutting Themes</td></tr>
<tr><td rowspan="4">Local, National & International Contexts</td><td colspan="3">There are many opportunities to include Local, National and International Contexts in GCSE Computer Science. These opportunities are important to learners as it helps them understand the relevance of Computer Science in real-world scenarios, foster global awareness, and encourage responsible digital citizenship, enabling them to see how technology can address local challenges, contribute to national innovations, and engage with international issues

Below are some examples of how Local, National & International Contexts can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.1.1
Investigation</td><td>To consider programming solutions that meets a set of requirements.</td><td>Activity 1: Bespoke Programming Challenge
Instruct learners to design and implement a programming solution that explores opportunities for local, national and international contexts. For example, design and implementation of a population calculator, history, flag or capital city quiz.</td></tr>
<tr><td>2.2.1
Design</td><td>Learners should be able to design solutions and their component parts, and use data modelling skills</td><td></td></tr>
<tr><td>2.3.1
Implementation</td><td>Learners should be able to use the Python programming language to write programs and build appropriate user interfaces</td><td>The pre-release brief set will be inclusive and consider exploration and insight of others’ experiences and perspectives. Learners will have the opportunity to develop insight and understanding of other's experiences and perspectives when considering the social, ethical, environmental and professional impacts of Computer Science.</td></tr>
</table>

<table>
<tr><td rowspan="4">Sustainability</td><td colspan="3">There are many opportunities to include Sustainability in GCSE Computer Science. These opportunities are important to learners because they promote awareness of environmental issues, empowering students to develop technology solutions that address sustainability challenges, and encourage responsible practices in coding and data management, ultimately fostering a sense of stewardship for the planet as they prepare for future careers in an increasingly eco-conscious world.

Below are some examples of how Sustainability can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.1.1
Investigation

2.2.1
Design

2.3.1
Implementation</td><td>To consider programming solutions that meets a set of requirements.

Learners should be able to design solutions and their component parts, and use data modelling skills

Learners should be able to use the Python programming language to write programs and build appropriate user interfaces</td><td rowspan="2">For example, consideration of environmental impacts and comparison between paper-based and digital data systems.

Activity 1: Specific Programming Activity
Opportunities to solve problems regarding energy consumption, carbon footprint and global warming. For example, an Eco-Friendly Carbon Footprint Emissions Calculator.

Activity 2: Case Study Analysis
Learners analyse a specific case involving a company’s decision to change from a paper-based data system to digital. Doing so, investigating and designing suitable procedures to changeover digitally.</td></tr>
</table>

<table>
<tr><td rowspan="3">Relationships and Sexuality Education</td><td colspan="3">There are many opportunities to include Relationships and Sexuality Education (RSE) in GCSE Computer Science. These opportunities are important to learners because they help them understand the implications of technology on relationships, promote respectful communication and to foster critical thinking about consent, privacy, and personal safety in digital interactions, ultimately supporting their overall well-being in a connected world.

Below are some examples of how RSE can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.3.1 Implementation</td><td>Learners should be able to use the Python programming language to write programs and build user interfaces that are appropriate for all.</td><td>Learners must ensure to implement software, especially algorithms that do not have biases or discrimination against certain groups. Learners will have the opportunity to develop insight and understanding of other's experiences and perspectives when considering the social, ethical, environmental and professional impacts of Computer Science.

Activity 1: Ethical Programming Workshop
Learners can participate in a workshop where they are taught ethical programming practices. Following this, they must create a simple program and document how they ensured it was free from bias. The documentation should include examples of potential biases and how they were mitigated.

Activity 2: Case Study Presentation
Learners will be tasked with preparing a case study on a software application that has been scrutinised for bias. They will present their findings to the class, focusing on its impact on specific communities and drawing connections to societal issues. The pre-release brief set will be inclusive and take into account exploration and insight of others’ experiences and perspectives.</td></tr>
</table>

<table>
<tr><td rowspan="3">Human Rights Education and Diversity</td><td colspan="3">There are many opportunities to include Human Rights Education and Diversity in GCSE Computer Science. These opportunities are important to learners because it encourages awareness of social justice issues, fostering understanding and respect for diverse perspectives, promoting equitable access to technology, and empowering learners to use Computer Science as a tool for advocacy and positive change, ultimately preparing them to be responsible and informed global citizens.

Below are some examples of how Human Rights Education and Diversity can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.3.2
Design

2.3.1
Implementation</td><td>Learners should be able to design solutions and their component parts.

Learners should be able to use the Python programming language to write programs and build appropriate user interfaces.

Unit 2 will explicitly allow development opportunities for learners to engage with languages, develop problem solving, logical thinking, decision making and reflective thinking. The pre-release brief will give learners the opportunity to overcome set challenges with programming, and, as a group take measured steps to solve the given problem(s).</td><td>Learners will have the opportunity to develop insight and understanding of other's experiences and perspectives when considering the social, ethical, environmental and professional impacts of Computer Science. Learners must ensure to develop software, especially algorithms that do not have biases or discrimination against certain groups.

Learners should have development opportunities to engage with languages, develop problem solving, logical thinking, decision making and reflective thinking. The pre-release brief will give learners the opportunity to overcome set challenges with programming, and, as a group take measured risks in software development prior to the final examination.

Activity 1: Code Scrutiny and Review
Learners will be given a sample piece of code that implements an algorithm. They will review the code and identify any potential biases that may arise from the algorithm’s logic or data used. Learners can suggest modifications to reduce bias.</td></tr>
</table>

			The pre-release brief set will be inclusive and consider exploration and insight of others' experiences and perspectives.
Careers and Work-Related Experiences	**There are many opportunities to include Career and Work-Related Experiences (CWRE) in GCSE Computer Science. These opportunities are important to learners because they provide practical insights into various career paths within industry, enhancing employability skills, fostering connections with each other, and allowing learners to apply theoretical knowledge in real-world contexts, ultimately preparing them for future employment and helping them make informed decisions about their career aspirations. Below are some examples of how CWRE can be embedded into teaching and learning:**		
	Specification Reference	***Amplification***	***Example***
	2.2.1 Design 2.3.1 Implementation	Learners should be able to design solutions and their component parts. Learners should understand how to write, correct, test and interpret the function of algorithms that solve real-life problems. Learners will have the opportunity to take risks, develop resilience and learn from experience, and develop creativity in algorithm design and the selection, identification and justification of appropriate data structures for different situations. Learners can learn about careers of the present and future when discussing professional impacts. This also applies for rights at the workplace, whilst also considering ethical and legal impacts.	Understanding algorithms is fundamental in many careers. By engaging with algorithms that solve real-life problems, learners can see how these skills are applied in industries such as software development, data science, finance, and healthcare. Writing, correcting, testing, and interpreting algorithms fosters critical thinking and analytical skills. These competencies are highly valued in the workplace, as employers seek individuals who can approach complex problems logically and systematically. **Activity 1: Real-Life Algorithm Design Challenge** Learners will design an algorithm to solve a specific real-life problem. Tasks can be specifically applied to appropriate workplaces and sectors such as construction, education, healthcare, transportation or agriculture. Challenges should be made to design and solve problems that should support and improve these sectors. For example: Construction: to calculate area or perimeter of a determined building site.

<table>
<tr><td></td><td></td><td></td><td>Education: to create a revision tool or testing resource for a student.

Transportation: to create a public transport route planner

Healthcare: to design a healthcare plan for an individual based on specific criteria entered.

Activity 2: Debugging Workshop
Learners learn to debug and fix coding and algorithmic errors, that can be provided from exemplar scenarios and problems as discussed in Activity 1.</td></tr>
<tr><td colspan="4">Cross-curricular Skills - Literacy</td></tr>
<tr><td rowspan="3">Listening</td><td colspan="3">There are many opportunities to include Literacy in GCSE Computer Science. These opportunities are important to learners because they enhance their ability to comprehend and communicate concepts, improve critical reading skills and foster effective written communication. Ultimately equipping learners with essential skills for success in both academic and professional settings.

Below are some examples of how Literacy can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.1.1
Investigation

2.4.2
Design</td><td>Discipline-specific vocabulary is developed throughout the entire course. Recall and response techniques can be developed throughout the entire specification content.</td><td>The Unit 2 pre-release brief encourages extensive research before attempting to solve the problem allowing learners to read a range of content and evaluate its fitness for purpose and effectiveness.

Activity 1: Group Discussions
The unit 2 pre-release brief will allow learners to consider different approaches to problem solving and consider the relevance of the differing approaches they have heard to solving the set task.</td></tr>
</table>

	Specification Reference	***Amplification***	***Example***
Reading	2.1.1 Investigation	Discipline-specific vocabulary is developed throughout the entire course. Recall and response techniques can be developed throughout the entire specification content.	The Unit 2 pre-release brief encourages extensive research before attempting to solve the problem allowing learners to read a range of content and evaluate its fitness for purpose and effectiveness.
	Specification Reference	***Amplification***	***Example***
Speaking	2.1.1 Investigation 2.2.1 Design	Learners should be able to analyse Computer Science problems, using decomposition, abstraction, pattern recognition and algorithms to determine solutions. Learners should be able to design solutions and their component parts, and use data modelling skills	**Activity 1: Group Discussions** The unit 2 pre-release brief will allow learners to consider different approaches to problem solving and consider the relevance of the differing approaches they have heard to solving the scenario. Learners are given the opportunity to effectively respond to others via group discussions in which they must communicate and respond to each other whilst discussing ethical considerations, where diverse perspectives are encouraged. The unit 2 pre-release brief will allow learners to consider different approaches to problem solving and consider the relevance of the differing approaches they have heard to solving the set problem. Subject specific terminology will be explicitly assessed throughout the unit. Feedback from peers is encouraged whilst working on the pre-release brief allowing for learners to respond to others feedback.

			The Unit 2 pre-release brief encourages extensive research before attempting to solve the problem allowing learners to read a range of content and evaluate its fitness for purpose and effectiveness. Units will be presented and assessed digitally, and learners will have the opportunity to draft and refine work as part of the development work based on the pre-release materials for Unit 2.
	Specification Reference	***Amplification***	***Example***
Writing	2.2.1 Design	Learners should be able to use the Python programming language to write programs and build appropriate user interfaces.	Work is presented and assessed digitally, and learners will have the opportunity to draft and refine work as part of the development work based on the pre-release materials. **Activity 1: Annotate Pre-Existing Code** Learners are to read examples of pre-written code, understand what is expected of the program and to effectively annotate code so that it is accessible to a competent third party. **Activity 2: Interface Mock-ups** Learners to produce free-hand sketches of interface mock-ups as a suitable guide and plan before commencing any implementation. These should include appropriate geometry, suitable containers, labels, text boxes, combo boxes or other suitable input or output, buttons, calculations or visible aspects that aim to be included on the form.

<table>
<tr><th colspan="4">Cross-curricular Skills - Numeracy</th></tr>
<tr><td rowspan="3">Developing Mathematical Proficiency</td><td colspan="3">There are many opportunities to include Numeracy in GCSE Computer Science. These opportunities are important to learners because they develop essential numeracy and mathematical skills required for problem-solving, logical reasoning, and data analysis. This enables learners to understand algorithms, evaluate computational efficiency, and apply reasoning in programming activities.

Below are some examples of how Numeracy can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.2.1
Design

2.3.1
Implementation</td><td>Plan, Design, Implement and Solve explicit mathematical problems.

Learners should be able to use the Python programming language to write programs and build appropriate user interfaces</td><td>Activity 1: Bespoke Programming Challenge
Learners to design and implement a programming solution that explores mathematical functions and develops proficiency. For example, design and implementation of a basic functional calculator, outputting the Fibonacci sequence, a tax rate calculator or a staff member payslip.

Unit 2 will have explicit mathematical problems to solve as part of the set brief each year.</td></tr>
<tr><td rowspan="3">Understanding the number system helps us to represent and compare relationships between numbers and quantities</td><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.2.1
Design

2.3.1
Implementation</td><td>Plan, Design, Implement and Solve explicit mathematical problems.

Learners should be able to use the Python programming language to write programs and build appropriate user interfaces</td><td>Activity 1: Data Capture Opportunities
Learners are given an opportunity to interpret data by designing effective data capture forms to collect and store data as part of their programmed solution.

Unit 2 will have explicit mathematical problems to solve as part of the set brief each year.</td></tr>
</table>

Learning about geometry helps us understand shape, space and position and learning about measurement helps us quantify in the real world	***Specification Reference***	***Amplification***	***Example***
	2.2.1 Design	Plan, Design, Implement and Solve explicit mathematical problems.	**Activity 1: Bespoke Programming Challenge** Learners to design and implement a programming solution that can solve geometry including area and perimeter of different shapes.
	2.3.1 Implementation	Learners should be able to use the Python programming language to write programs and build appropriate user interfaces	Unit 2 will have explicit mathematical problems to solve as part of the set brief each year.
Learning that statistics represent data and that probability models chance help us make informed inferences and decisions	***Specification Reference***	***Amplification***	***Example***
	2.2.1 Design	Plan, Design, Implement and Solve explicit mathematical problems.	**Activity 1: Bespoke Programming Challenge** Ask your learners to design and implement a programming solution to a digital game. For example, randomisation of a dice roll, rock/paper/scissors. Games that can utilise probability models and analysis.
	2.3.1 Implementation	Learners should be able to use the Python programming language to write programs and build appropriate user interfaces	Unit 2 will have explicit mathematical problems to solve as part of the set brief each year.

<table>
<tr><th colspan="4">Cross-curricular Skills - Digital Competence</th></tr>
<tr><td rowspan="2">Citizenship</td><td colspan="3">There are many opportunities to include Digital Competence in GCSE Computer Science. These opportunities are important to learners because they equip learners with the skills to effectively use digital tools and technologies, promote critical thinking about information, enhance their ability to navigate digital environments safely, and prepare them for an increasingly technology-driven world.
Below are some examples of how Digital Competence can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.2.1 Investigation

2.2.1 Design

2.3.1 Implementation</td><td>Plan, Design, Implement and Solve explicit mathematical problems.

Learners should be able to use the Python programming language to write programs and build appropriate user interfaces</td><td>During the development phase of the pre-release materials learners will have the opportunity to collect information reflecting on copyright usage of others work and logging sources. This will not be formally assessed.

Activity 1: Validation and Verification Planning
Learners have opportunities to plan and create suitable and appropriate validation and verification techniques to keep data and software secure and accurate.

Activity 2: Bespoke Programming Challenge
Learners to design and implement a programming solution for a digital quiz that explore digital citizenship topics, such as online safety, respectful communication, and data privacy.</td></tr>
</table>

	Specification Reference	***Amplification***	***Example***
Interacting and Collaborating	2.2.1 Design 2.3.1 Implementation 2.4.1 Testing	Plan, Design, Implement and Solve explicit mathematical problems. Learners should be able to use the Python programming language to write programs and build appropriate user interfaces. Learners should be able to use the Python programming language to test and refine effective programs and build appropriate user interfaces	**Activity 1: Paired Programming** Give learners an opportunity to collaborate on a coding approach by working simultaneously on the same code. Through frequently switching roles, learners can ensure both engagement in hands-on coding and strategic thinking. This method promotes shared learning, improves code quality, and helps develop communication, critical thinking and problem-solving skills through teamwork, as both partners collaborate to debug and improve the code, learning from each other's approaches and solutions.
	Specification Reference	***Amplification***	***Example***
Producing	2.3.1 Implementation	Learners should be able to use the Python programming language to write programs and build appropriate user interfaces	**Activity 1: Bespoke Programming Challenge** Learners should have an opportunity to produce bespoke and functional programming solutions to a given brief through solving a specific problem. For example, games, quizzes or calculators.

	Specification Reference	***Amplification***	***Example***
Data and Computational Thinking	2.2.1 Design	Understand how to write, correct, test and interpret the function of algorithms that solve real-life problems.	**Activity 1: Algorithm Design** Learners should design algorithms for real-world scenarios. Ensure they include a range of algorithmic process, such as sequence, selection, and iteration. They then test each other's algorithms with inputs to ensure it functions effectively. **Activity 2: Bespoke Programming Challenge** Learners to produce a programming solution to a given brief and solving a specific problem. Examples of which can be used similarly to those discussed earlier. These programming tasks give learners the opportunity to develop their computational thinking throughout Unit 2 including problem solving and modelling and to develop data and information literacy.
	2.3.1 Implementation	Learners should be able to use the Python programming language to write programs and build appropriate user interfaces	
	2.4.1 Testing	Learners should be able to use the Python programming language to test and refine effective programs and build appropriate user interfaces	

<table>
<tr><th colspan="4">Integral Skills</th></tr>
<tr><td rowspan="3">Creativity and Innovation</td><td colspan="3">There are many opportunities to include Creativity and Innovation in GCSE Computer Science. These opportunities are important to learners because they encourage imaginative thinking and problem-solving, empowering learners to develop unique solutions to real-world challenges in preparation for a rapidly evolving digital landscape where creativity is essential for driving technological advancement and innovation.

Below are some examples of how Creativity and Innovation can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.1.1
Investigation

2.2.1
Design

2.3.1
Implementation</td><td>Through exploratory research, leaners can explore various methodologies for solving problems. By investigating emerging trends in computer science, such as artificial intelligence or cybersecurity, to inspire innovative thinking in their programming solutions.

Allowing learners to identify and define problems creatively, including sessions where students think ‘outside the box’ to find unique software problems and solutions.</td><td>During their work on the pre-release brief, learners will have the opportunity to research new solutions and generate new ideas as to how to solve the set problem. Learners will be able to compare alternative solutions to the problem and justify the methods used in their approach.</td></tr>
</table>

		Learners should be challenged to design solutions that are functional and innovative. This can include creating new algorithms, developing unique user interfaces, or designing unique applications. Learners can create prototypes of designs that will allow them to experiment with different ideas and iterate on their designs creatively. By promoting groupwork to implement and solve problems, this can spark new ideas and lead to more innovative solutions.	

<table>
<tr><td rowspan="2">Critical Thinking and Problem Solving</td><td colspan="3">There are many opportunities to include Critical Thinking and Problem Solving in GCSE Computer Science. These opportunities are important to learners because they cultivate essential skills for analysing complex problems, developing systematic approaches to finding solutions, enhancing logical reasoning abilities, and preparing students to tackle real-world challenges effectively, which are crucial for success in both academic and professional environments.

Below are some examples of how Critical Thinking and Problem Solving can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td></td><td>2.2.1
Investigation

2.2.1
Design

2.3.1
Implementation</td><td>Learners should be able to break down complex problems into smaller, manageable parts allowing them to understand the problem thoroughly before attempting to solve them.

Learners should be able to gather relevant information and data from various sources, including using online resources, textbooks, and peer discussions to form an understanding of the problem.

Learners should be able to create algorithms to solve problems, enhancing their ability to think logically and systematically.</td><td>Data and computational thinking are consistent and essential within Unit 2 and being able to effectively use computational thinking techniques to solve real-world scenarios through the development of algorithms is fundamental.

Activity 1: Algorithm Design and Implementation
Learners design algorithms for real-world scenarios. Ensure they include a range of algorithmic process, such as sequence, selection and iteration. They then test each other's algorithms with inputs to ensure it works effectively.

During their work on the pre-release brief, learners will have the opportunity to research new solutions and generate new ideas as to how to solve the set problem. Learners will be able to compare alternative solutions to the problem and justify the methods used in their approach.</td></tr>
</table>

		Learners should have the ability to use and interpret flowcharts to represent the steps involved in solving a problem. By implementing their algorithms learners can practice reinforcing their problem-solving skills. Through debugging and testing, learners should display the ability to test their code and debug errors. Through this iterative process, learners will develop resilience and attention to detail.	

<table>
<tr><td rowspan="3">Planning and Organisation</td><td colspan="3">There are many opportunities to include Planning and Organisation in GCSE Computer Science. These opportunities are important to learners because they help develop essential skills in project management, time management, and resource allocation, enabling learners to effectively organise their work and set achievable goals.

Below are some examples of how Planning and Organisation can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.1.1

2.2.1</td><td>Investigation
Learners should be able to analyse Computer Science problems, using decomposition, abstraction, pattern recognition and algorithms to determine solutions.

Design
Learners should be able to design solutions and their component parts, and use data modelling skills.</td><td>Unit 2 pre-release materials give learners the opportunity to develop a programmed solution to a problem over an extended period of time. This will allow learners to set goals and monitor the results of different programming attempts. Learners will have opportunities to reflect and adapt their programming approach and manage their time and resources to complete tasks.</td></tr>
</table>

<table>
<tr><td rowspan="3">Personal Effectiveness</td><td colspan="3">There are many opportunities to include Personal Effectiveness in GCSE Computer Science. These opportunities are important to learners because they foster self-awareness, motivation, and resilience, helping learners to develop effective study habits, set personal goals, and reflect on their progress.

Below are some examples of how Personal Effectiveness can be embedded into teaching and learning:</td></tr>
<tr><td>Specification Reference</td><td>Amplification</td><td>Example</td></tr>
<tr><td>2.4.2 Refinement and Evaluation</td><td>Learners should be able to use the Python programming language to test and refine effective programs and build appropriate user interfaces.</td><td>Learners should have the opportunity to develop solutions independently becoming more confident and independent programmers. Learners will evaluate their learning by reflecting on their programming mistakes whilst developing a solution and plan areas that they need to further develop.

Activity 1: Development and Reflection Log
Document any challenges faced in algorithmic design and how they overcame them. Reflect on any problem-solving strategies and what they’ve learned from each task or assignment. Set personal goals for improving specific skills.</td></tr>
</table>

Glossary for Unit 2

Term	Definition
Abstraction	Simplifying a complex system by focusing on the essential details and ignoring the irrelevant ones.
Additional Requirements	New or extra needs that are added to the original requirements, often leading to changes in the software.
Algorithm	A step-by-step set of instructions to solve a problem or complete a task.
Annotation	Adding notes or comments to code to explain what it does.
Approaches	Different methods or strategies used to solve a problem or meet the requirements in software development.
Assignment	Setting a value to a variable.
Authentication	Verifying the identity of a user or system.
Boolean	A data type that can hold only two values: True or False.
Button	Clickable widgets that perform an action when pressed. They can display text or images.
Character	A single letter, digit, or symbol. When stored on a computer system, they are stored as a binary number.
Code	The set of instructions written in a programming language that tells the computer what to do.
Concepts	Key ideas or principles that are fundamental to understanding software development.
Conditional Controlled Loop	A loop that continues to execute as long as a certain condition is true.
Control Flow	The order in which individual statements, instructions, or function calls are executed or evaluated in a program.
Count Controlled Loop	A loop that runs a specific number of times.
Data Handling	The process of collecting, storing, and managing data.
Data Structure	A specific way of organising and storing data in a computer so it can be processed efficiently.
Data Types	Different kinds of data that can be used in programming, such as integers, Booleans, and strings.
Decomposition	Breaking down a complex problem or system into smaller, more manageable parts.
Development	The process of creating and improving software.
Developmental Changes	Modifications or improvements made to the software during its development process.
Dictionaries	A collection of key-value pairs, where each key is unique.

Documented	Written or typed details, usually to explain how something works or how it was created.
Efficiency	How well the software uses resources like time and memory; better software performs tasks faster and uses less memory.
Erroneous Data	Incorrect or invalid data used to test how a program handles errors.
Evaluation	The process of assessing or judging the quality, effectiveness, and impact of the software or the changes made to it.
Extreme Data	Data at the upper or lower limits of what a program can handle.
File Handling	The process of reading from and writing to files.
Fixed Length Record	A record in which each data field has a pre-determined set length.
Frame	Rectangular widgets used to organise other widgets visually and at the coding level. They act as containers for other elements.
Functions	A block of code designed to perform a specific task, which can be called when needed.
Geometry Manager	Tkinter provides three geometry managers (pack, grid, and place) to control the layout and positioning of widgets within a window or frame. Each manager has its own rules for arranging widgets.
Graphical User Interfaces	(GUI) Visual interfaces that allow users to interact with electronic devices using graphical elements like windows, icons, and buttons.
Identifiers	Names used to identify variables, functions, classes, and other objects in code.
Indentation	The use of spaces or tabs to visually structure code and show the relationships between different code blocks.
Input	Data or information that is entered into a system for processing.
Integer	A positive or negative whole number.
Interface	The way in which different components of a system communicate with each other.
Iteration	Repeating a set of instructions a certain number of times or until a condition is met.
Libraries	Collections of pre-written code that can be used to perform common tasks, saving time and effort.
List	A collection of items stored in a particular order.
Logical Operations	Operations that return true or false, such as AND, OR, and NOT.
Manipulation	The act of changing or handling data to achieve a desired result.
Mathematical Operations	Calculations like addition, subtraction, multiplication, and division.

Methods	Functions that are defined inside a class in object-oriented programming.
Model	A representation or simulation of a real-world process or system.
Modify	To change or alter something.
Modular	Designed with individual components that can be independently created and used in different systems.
Modules	Independent, self-contained units of code that can be combined with others to create a program.
Operators	Symbols that perform operations on variables and values, like addition (+) or comparison (==).
Outcomes	The results or effects of the code or the changes made to it.
Output	Data or information that comes out of a system after processing.
Problem Solving	The process of finding solutions to difficult or complex issues.
Processing	Performing operations on data to transform, analyse, or manipulate it.
Program	A set of instructions written in a programming language that a computer can execute.
Programming Constructs	Basic elements used to build programs, such as loops, conditionals, and sequences.
Pseudo Code	A way of writing algorithms using plain language instead of actual programming code.
Read	The process of getting data from a file or input device.
Real	A number which can include a fraction or a decimal point.
Record	An organised collection of related fields, this holds data on one person or object.
Refinement	The process of making small improvements to the code to make it work better or more efficiently.
Requirements	Specific capabilities that a system must have or perform.
Revised Requirements	Changes or updates made to the original needs or conditions that a software program must satisfy.
Routines	A set of instructions that perform a specific task, often used repeatedly in a program.
Selection	Choosing which set of instructions to execute based on a condition, often using if-else statements.
Self-Documenting	Code written in a way that is easy to understand without needing additional comments or documentation.
Sequence	The order in which instructions are executed in an algorithm or program.
Slicing	Selecting a part of a list, string, or other sequence data type.
Solution	The final implementation or method used to solve a problem in the software.

Strengths	The positive aspects or advantages of a particular approach or method.
String	A sequence of characters.
Subroutines	Smaller sections of code designed to perform a specific task, similar to functions.
Systematic	A methodical and organised approach to solve a problem or task.
Test Data	Data used to test a program to see if it works as expected.
Testing	The process of running a program to find and fix bugs or errors, and to ensure it works correctly.
Testing Strategy	A plan for testing a system to ensure it works correctly and meets requirements.
Third Party	Software or components created by someone other than the original developer of the system or application.
Tuples	A fixed-size collection of elements, often of different types.
Typical Data	Normal data that is expected to be used by a program.
User Interface	The part of a computer system that allows a user to interact with it.
Validation	An automatic check used to determine if data entered into a system is sensible and meets certain criteria.
Variable	A storage location in a program with a name and a value that can change.
Variables	Named storage locations in a program that hold data which can be changed during program execution.
Verification	A check used to determine if the data entered into a system is correct.
Weaknesses	The negative aspects or disadvantages of a particular approach or method.
White Space	Any character or series of characters that represent horizontal or vertical space in text, like spaces, tabs, and newlines, used for readability.
Widget	General term for all GUI elements in Tkinter, including buttons, labels, entry fields, and more.
Window	A top-level container in Tkinter that represents a main application window. It can hold other widgets and frames.
Write	The process of sending data to a file or output device.