The Island VI Form

Together we inspire | Together we discover | Together we flourish

Post-16 Curriculum - Computer Science

Subject Curriculum Overview

Subject	**Computer Science**
Qualification / Specification	H446
Awarding Body	OCR
Level	3
Curriculum Lead	Kamelia Howells
Teaching Staff	Gavin Gray
Guided Learning Hours	360
Year(s) Delivered	12 & 13
Last Review Date	June 2026
Next Review Date	June 2027

1. Curriculum Intent

Purpose of the Subject: The purpose of A-Level Computer Science is to develop abstract thinking, problem-solving, and algorithmic design skills. It moves students from being mere consumers of technology to creators, deeply understanding the mathematical and logical frameworks that underpin modern computing systems. It progresses from short scripts that are typical of GCSE to much larger programs.

Progression, Destinations, & Meeting Industry/University Needs: The curriculum acts as a critical bridge to higher education (Computer Science, Software Engineering, AI, and Data Science degrees) and industry pathways (Degree Apprenticeships, software development, and cybersecurity roles). By covering the OCR H446 specification, it directly meets university and employer demands by emphasizing:

- **Computational Thinking:** Deconstruction, abstraction, pattern recognition, and algorithm design.
- **Practical Programming Paradigm Mastery:** Rigorous, object-oriented, and procedural programming via the Non-Exam Assessment (NEA) and multiple instances of programming practice during lesson time.
- **Theoretical Foundations:** Deep understanding of systems architecture, data structures, boolean algebra, network topologies, and the legal/ethical implications of computing.

Success for Learners: Success means learners who are resilient debuggers, autonomous programmers, and analytical thinkers. They don't just memorize content for the H446 exams; they can independently architect a software solution from scratch for their NEA, articulate how hardware and software interact, and confidently apply logical reasoning to novel, real-world computational problems.

2. Foundational Knowledge

Area	Essential Knowledge / Skills	Why It Matters
Basic Programming Constructs	Proficiency in sequence, selection, and iteration using a high-level language (mostly Python in lessons due to network restrictions, but pupils can study other languages). Understanding basic data types (Integer, Real, String, Boolean).	Students cannot access the complex algorithmic thinking or the NEA project if they are still struggling with basic syntax and program flow.
Mathematical Logic & Data Representation	Understanding binary, hexadecimal, and basic arithmetic conversions. Familiarity with standard logic gates (AND, OR, NOT, XOR).	This underpins the entire Component 01 syllabus. Students must understand how computers represent data at a hardware level to grasp data structures and processor architecture later.
Problem Decomposition	The ability to take a complex problem and break it down into smaller, manageable, and programmable sub-problems (subroutines/functions).	Crucial for transitioning from writing short scripts to architecting larger systems. It builds the logical resilience needed for A-Level style exam questions.
Basic Algorithm Comprehension	Understanding standard linear and binary searching, as well as basic sorting (e.g., Bubble Sort).	Serves as the stepping stone to the more complex algorithms in Component 02, such as Dijkstra's algorithm, tree traversals, and Big O notation.
Mathematical Foundations & Functions	Comfort with standard algebraic manipulation, understanding functions (inputs and outputs), and basic set theory or coordinate systems.	A-Level Computer Science has a heavy mathematical undercurrent. Students need this foundation for Boolean algebra simplification, mapping arrays, and understanding functional programming concepts.
Basic Data Structures & Storage	Understanding the difference between static and dynamic data storage, and familiarity with standard arrays (1D and 2D) and records.	Before students can master advanced Abstract Data Types (like graphs, trees, and linked lists) in the main spec, they must thoroughly understand how data is sequentially indexed and manipulated in memory.

3. Powerful Knowledge

Powerful Knowledge Area	Why It Matters	Evidence of Understanding
The Von Neumann Architecture & Processor Operations	It demystifies the hardware. Understanding the Fetch-Decode-Execute cycle, registers, and interrupts ensures students know exactly how their code interacts with physical silicon.	Students can trace data movement through the CPU registers (MAR, MDR, PC, ACC) during a clock cycle and explain how Assembly language instructs hardware directly.
Computational Thinking & Abstraction	It enables students to strip away real-world complexities and model complex scenarios programmatically. It is the core mental framework of all software engineering.	Students can successfully translate a complex, messy problem description into clear pseudocode or a flowchart, identifying exactly what data needs to be hidden or generalized.

Algorithm Efficiency & Big O Notation	Moving beyond "does the code work?" to "how well does it scale?". Understanding time and space complexity prevents students from writing inefficient code that fails in enterprise environments.	Students can analyze a given algorithm (e.g., comparing Quicksort to Bubble Sort) and mathematically justify its efficiency using Big O notation for best, worst, and average cases.
Programming Paradigms (OOP & Procedural)	Exposes students to different architectural mindsets. Understanding Object-Oriented Programming (OOP) is critical, as it mirrors how modern industry software is built using encapsulation, inheritance, and polymorphism.	Students can independently write modular, reusable code using classes, methods, and private attributes, and can explain why OOP is advantageous for large-scale team projects.
Abstract Data Types (Graphs, Trees, Vectors)	Modern data isn't just stored in flat lists; it is relational and networked (like social media graphs or file directories). Mastering these allows for advanced data manipulation.	Students can manually trace or programmatically implement algorithms to traverse these structures (e.g., Depth-First/Breadth-First search or tree traversals) without breaking the logical rules of the structure.

4. Curriculum Implementation

Term / Unit	Key Content	Foundational Knowledge	Powerful Knowledge	Assessment Opportunities
Yr 12 - Autumn Fundamentals of Programming & Architecture	Procedural programming (syntax, variables, loops), Data representation (binary, hex), CPU architecture (Von Neumann, FDE cycle). Application generation Software development Types of programming Introduction to programming	Basic Programming Constructs; Mathematical Logic.	The Von Neumann Architecture & Processor Operations.	Baseline programming assessment; End-of-unit tests on individual units.
Y12 - Spring Databases, Networks & Component 2 Theory	Data Types. Compression, encryption, hashing. Relational databases, SQL. Networks. Web technology. Programming skills in a variety of paradigms (OOP, games, API's, apps)	Mathematical Foundations & Functions; Mathematical Logic.	Computational Thinking & Abstraction; SQL	Programming assessments. End-of-unit tests on individual units.
	Object-Oriented Programming (OOP)	Basic Data Structures & Storage	Advanced Data structures, knowledge of how to	End-of-unit tests on individual units.

Yr 12 - Summer Data structures. NEA Projects	Boolean logic and Algebra. Basic and Advanced Data Structures (Trees, Graphs, stacks, queues, linked lists), NEA Design & Development. NEA Programming Project Legislation and ethics		implement, traverse and perform specific actions (push, pop, peek, enqueue, dequeue)	Trial exam (component 1)
Y13 - Autumn Programming techniques Algorithms	Programming techniques. Algorithms (Search and sort, pathfinding) Computational methods. NEA Project	Searching and sorting algorithms. Computational methods/thinking.	Pathfinding algorithms (Dijkstra, A*) Algorithm Efficiency (Big O notation)	End-of-unit tests on individual units.
Y13 - Spring	Completion of NEA project. Review of previous units including exam question technique for each topic.	Mathematical Foundations & Functions; Mathematical Logic	Computational Thinking & Abstraction; Algorithm Efficiency.	Trial exams (component 1 and 2) Final NEA submission
Y13 - Summer Revision & Synoptic Practice	Targeted intervention, exam technique workshops, synoptic problem-solving, and full past paper practice.	All foundational areas.	Synthesis of all Powerful Knowledge areas.	Timed past paper tracking; Walking-talking mocks; Component 1 and 2 diagnostic quizzes.

5. Assessment Strategy

Assessment Type	Purpose	Frequency
Diagnostic	To assess baseline programming literacy, mathematical competency, and logical reasoning at the start of Year 12. This identifies students requiring immediate intervention or scaffolding before starting complex H446 theory.	At the start of the Year 12 course. At the start of more technical units (e.g., before introducing Boolean Algebra or Assembly language).

Formative	To monitor ongoing learning, address misconceptions in real-time, and gauge understanding of programming syntax and algorithmic tracing. This directly informs short-term lesson planning.	Regular coding challenges and debug exercises. In-class low-stakes quizzes (e.g., multiple-choice retrieval questions at the start of lessons). Live code reviews and feedback on NEA components.
Summative	To measure retention and application of knowledge across entire OCR specification modules. This provides formal data tracking points for predictions and mocks.	End-of-topic tests (e.g., CPU Architecture, Data Structures). Formal end-of-year mock exams in Year 12. Full Component 1 and Component 2 mock series in Year 13
Synoptic	To evaluate a student's ability to connect disparate parts of the spec—combining theoretical computational principles with practical software engineering, exactly as required by the NEA and Component 02 exam.	Mid-to-late Year 13, through comprehensive past paper practice. Evaluated continuously through the creation and documentation of the independent Non-Exam Assessment (NEA).