HASMONEAN
אל תקרי בניך אלא בוניך

COMPUTER SCIENCE

What is Computer Science?

Computer Science is a rigorous, creative, and transformative academic discipline that combines logical problem-solving, mathematical theory, and practical software engineering. At its heart, Computer Science is the study of computation, information processing, and automated reasoning—exploring not only how digital systems operate, but how they can be architected to solve complex, real-world problems.

Through computational thinking, algorithm design, system architecture, data structures, and text-based programming, students learn to decompose intricate challenges, abstract away irrelevant detail, and construct efficient, robust solutions. Far beyond technical fluency, Computer Science empowers students to transition from passive consumers of technology into active creators, innovators, and ethical guardians of the digital infrastructure that shapes modern global society.

Why do we teach Computer Science at Hasmonean?

At Hasmonean, we believe that Computer Science is essential for equipping students with the intellectual tools, technical mastery, and ethical framework required to thrive in an increasingly technology-driven world. Computing fosters disciplined analytical thinking, logical resilience, and mathematical precision, providing a vital bridge between theoretical logic and practical application.

> **Core Values in Computer Science**
> *The Computer Science curriculum reflects our core school values of Aspire, Respect, and Engage:*
>
> *• Aspire: We challenge students to solve high-level algorithmic problems, master industry-standard text-based languages (Python, SQL, Assembly), and architect complex software systems, encouraging ambition and academic excellence.*
> *• Respect: Students critically evaluate technological legislation, cyber security, data privacy (GDPR), ethical artificial intelligence, and the societal and environmental impacts of technology, fostering deep respect for human rights, personal data, and digital safety.*
> *• Engage: Students actively engage in practical problem-solving, hands-on physical computing, system design, and real-world Non-Exam Assessment (NEA) projects, developing independence, perseverance, and collaborative confidence.*

Beyond discrete subject knowledge, Computer Science develops highly transferable cognitive skills—including decomposition, pattern recognition, systematic testing, and algorithmic efficiency—that directly strengthen achievement across STEM disciplines and prepare students for higher education, technological leadership, and lifelong learning.

How do we teach Computer Science at Key Stage 3?

Our Key Stage 3 curriculum is systematically sequenced across Years 7, 8, and 9 to build computational literacy, algorithmic confidence, and digital responsibility. Students progress logically from visual and physical control environments to formal text-based programming and computer systems theory.

Year 7: Digital Literacy, Data Foundations & Control Logic

- **Autumn (Term 1) – E-Safety & Cyber Security:** Establishes vital personal online safety, cyberbullying awareness, malware classification (viruses, spyware, ransomware), social engineering mechanics (phishing), and digital footprint management before engaging with broader computer systems.
- **Winter (Term 2) – Binary & Spreadsheet Data Modeling:** Transitions from digital citizenship to foundational data representation. Students master binary-to-denary conversions, ASCII character encoding, and functional spreadsheet modeling (SUM, AVERAGE, IF statements, lookup functions).
- **Spring (Term 3) – Computer Hardware & Networks:** Connects data representation to physical architecture. Examines CPU components, RAM/ROM, input/output devices, secondary storage, network types (LAN/WAN), protocols, and network topologies (Star, Bus, Ring).
- **Summer (Term 4) – Control Systems & Algorithmic Flowcharts:** Applies logic and hardware principles to visual algorithmic design using Flowol 4, modeling real-world automated control systems (traffic lights, zebra crossings, robotic systems).

Year 8: Visual Coding, Physical Computing & Python Foundations

- **Autumn (Term 1) – Block-Based Programming (NextBlock/Blockly):** Extends flowchart logic into block-based coding, mastering sequence, variables, user input, conditional selection (IF/ELSE), iteration (counted/conditional loops), and software debugging via a capstone mini-project.
- **Winter (Term 2) – Physical Computing with the BBC micro:bit:** Translates screen-based code into tangible physical outcomes, programming text displays, custom LED animations, event-driven button handlers, and motion sensors (accelerometers).
- **Spring (Term 3) – Text-Based Programming in Python:** Pivots from visual blocks to industry-standard Python text programming. Covers syntax rules, variable declaration, dynamic input, conditional branch logic (IF/ELIF/ELSE), and iterative constructs (WHILE and FOR loops).
- **Summer (Term 4) – Advanced Number Systems, Logic Gates & Legislation:** Extends mathematical foundations to Hexadecimal conversions and Boolean Logic Gates (AND, OR, NOT with truth tables). Evaluates computer legislation (Data Protection Act, Computer Misuse Act, Copyright Act, Online Safety Act) and societal/environmental impacts.

Year 9: Advanced Programming, Algorithmic Theory & Data Architecture

- **Autumn (Term 1) – Advanced Python & Data Structures:** Refines Python proficiency by introducing complex data structures (Lists/Arrays), subroutine modularisation (functions and procedures), and practical software synthesis via a capstone Fruit Machine project.
- **Spring (Term 2) – Fundamentals of Algorithms & Computational Thinking:** Formalises theoretical computational thinking (Abstraction, Decomposition, Algorithmic Thinking). Students master trace tables, pseudocode design, Searching algorithms (Linear/Binary Search), and Sorting algorithms (Bubble/Merge Sort).
- **Summer (Term 3) – Advanced Data Representations:** Consolidates mathematical data representation, mastering 8-bit binary addition, overflow conditions, logical bit shifts (left/right), Hexadecimal conversion, and character sets (ASCII vs Unicode), providing a seamless transition into Key Stage 4.

How do we teach Computer Science at Key Stage 4?

At Key Stage 4, students follow the AQA GCSE Computer Science (8525) specification. The curriculum is structured to complete all new theoretical and practical content by the end of Year 10, reserving Year 11 for rigorous synoptic revision, past paper mastery, and exam technique drills.

Component	Assessment Format	Weighting & Focus
Paper 1: Computational Thinking & Programming Skills	Written Exam 1 hr 30 mins (80 marks)	50% of GCSE Tests computational thinking, code analysis, tracing, pseudocode, searching/sorting, and Python programming logic.
Paper 2: Computing Concepts	Written Exam 1 hr 30 mins (80 marks)	50% of GCSE Tests data representation, computer systems, architecture, networks, cyber security, legal/ethical impacts, and SQL relational databases.

Year 10: Complete Specification Coverage & Applied Projects

- **Autumn (Term 1) – Media Representation & Data Compression:** Completes Unit 3 data representation, covering 8-bit binary arithmetic, bit shifts, image representation (pixels, color depth, resolution), sound sampling (bit depth, sample rate), and data compression techniques (RLE, Huffman Coding).

- **Winter (Term 2) – Hardware Architecture & Networking Infrastructure:** Delivers Unit 4 (CPU F-D-E cycle, primary memory, secondary storage, logic gates) and Unit 5 (LAN/WAN, network topologies, protocols, TCP/IP stack layers).
- **Spring (Term 3) – Advanced Programming & Capstone Software Projects:** Advances modular software development using Lists, Subroutines (parameters/returns), and external File I/O. Practical skills are embedded through multi-week projects (Celebrity Dogs and Tic-Tac-Toe).
- **Spring (Term 4) – Cyber Security, Tech Legislation & Ethics:** Covers Unit 6 cyber security threats (social engineering, malware, threat mitigation) and Unit 7 legal frameworks (GDPR 2018, Computer Misuse Act 1990, Copyright Act 1988) alongside ethical and environmental technology analysis.
- **Summer (Term 4) – Relational Databases & SQL:** Mastery of Unit 8 relational database design (primary/foreign keys) and SQL query construction (SELECT, WHERE, ORDER BY, INSERT, UPDATE, DELETE, multi-table JOINs).

Year 11: Synoptic Revision, Algorithm Optimisation & Exam Mastery

- **Autumn (Term 1) – Paper 1 Synoptic Revision & Code Prediction:** Intensive focus on Unit 1 algorithms and Unit 2 Python programming. Drills trace tables, error diagnosis, high-tariff pseudocode writing, and search/sort optimization.
- **Winter (Term 2) – Paper 2 Synoptic Revision & System Calculations:** Comprehensive revision of Units 3 to 6. Focuses on binary/hex math calculations, logic circuit simplification, subnetting concepts, and networking protocols.
- **Spring (Term 3) – Applied SQL, Legal Essay Structure & Full Mock Practice:** Targeted review of Unit 7 ethical essay writing and Unit 8 SQL syntax. Complete timed past paper practice, mark scheme analysis, command word drills, and misconception removal ahead of May examinations.

How do we teach Computer Science at Key Stage 5?

At Key Stage 5, students study the AQA A-Level Computer Science (7517) specification. The course introduces advanced computational theory, mathematical data structures, functional paradigms, low-level machine architecture, and systematic software engineering.

A central component of Key Stage 5 is the Non-Exam Assessment (NEA) Programming Project (20% of A-Level). Crucially, 20% of curriculum lesson time throughout Year 12 and Year 13 is explicitly dedicated to guiding students through the formal Software Development Life Cycle (SDLC)—from complex problem identification and 3NF relational database architecture to object-oriented coding, systematic testing, and evaluation.

Component	Assessment Format	Weighting & Focus
Paper 1: Practical Problem-Solving & Skeleton Code	On-Screen Computer Exam 2 hrs 30 mins (100 marks)	40% of A-Level Tests practical programming, data structures, algorithm efficiency (Big-O), recursion, theory of computation, and modification of AQA pre-released Skeleton Code.
Paper 2: Computing Concepts & Architecture	Written Exam 2 hrs 30 mins (100 marks)	40% of A-Level Tests data representation, hardware architecture, low-level assembly, networking, TCP/IP, web services, relational databases (3NF), and 12-mark evaluation essays.
Non-Exam Assessment (NEA): Programming Project	Substantive Practical Software Project Report & Code (75 marks)	20% of A-Level Independently assesses the systematic software life cycle (Analysis, Design, Technical Solution, Systematic Testing, Evaluation) solving a complex real-world problem.

Year 12: Advanced Theoretical Foundations & NEA Analysis/Design Draft

- **Autumn (Term 1) – Data Structures, Complex Algorithms & Big-O:** Front-loads dynamic Abstract Data Types (Queues, Stacks, Call Stack Frames, Hash Tables, Graphs, Trees, Vectors) and complex algorithms (Recursion, Big-O complexity, Dijkstra's algorithm, Depth/Breadth-First Search, Limits of Computation). NEA Allocation: Project initiation, client analysis, requirements gathering.
- **Winter (Term 2) – Programming Fundamentals, Relational Databases (3NF) & SQL:** Consolidates control structures, arrays, subroutines, file I/O, and robust exception handling with Unit 11 ER modeling, Normalisation (1NF to 3NF), and multi-table SQL. NEA Allocation: Schema design, pseudocode subroutines, UI mockups.
- **Spring (Term 3) – OOP, Functional Programming & Mathematical Data Representation:** Introduces Object-Oriented paradigms (encapsulation, inheritance, polymorphism, UML diagrams) and Functional Programming (pure functions, higher-order Map/Filter/Fold). Covers Unit 3 binary math, Two's Complement, fixed-point, vector graphics, and Huffman/Vernam ciphers. NEA Allocation: Class hierarchy and OOP design.
- **Spring (Term 4) – Theory of Computation & Language Automata:** Establishes computational thinking, algorithm hand-tracing, Finite State Machines (Mealy machines), Set

Theory, Regular Expressions (Regex), Universal Turing Machines, Backus-Naur Form (BNF), and Reverse Polish Notation (RPN). NEA Allocation: Formulating test plans.

- **Summer (Term 5) – Hardware Systems, Web Technologies & NEA Milestone 1:** Covers OS resource management, compilers vs interpreters, De Morgan's laws, TCP/IP stack layers, subnetting, public/private keys, RESTful web services, and JSON. NEA Deliverable: Formal submission of complete FIRST DRAFT containing Analysis and Design sections.

Year 13: Machine Architecture, Skeleton Code Mastery & Full NEA Completion

- **Autumn (Term 1) – Computer Architecture, Assembly & Skeleton Code Analysis:** Explores CPU buses, F-D-E cycle, register operation, ISR interrupts, and Assembly language programming. Introduces pre-released AQA Paper 1 Skeleton Code architecture. NEA Allocation: Intensive technical coding execution in Python/C#/Java.
- **Winter (Term 2) – Network Communication, Computing Legislation & NEA Milestone 2:** Mastery of serial/parallel transmission, CSMA/CA, network topologies, digital legislation (GDPR, CMA, CDPA), and AI ethics. NEA Deliverable: Mandatory submission of ALL SECTIONS OF THE REPORT COMPLETED AS A FIRST DRAFT (Analysis, Design, Solution, Testing, Evaluation).
- **Spring (Term 3) – Paper 1 Exam Drills, Skeleton Code Mastery & Final NEA Submission:** Intensive Paper 1 Exampro revision, trace tables, recursive algorithms, and Skeleton Code Section C/D exam tasks. NEA Deliverable: Final submission of polished NEA Project for internal moderation.
- **Spring (Term 4) – Paper 2 Synoptic Revision & Extended Prose Essay Mastery:** Targeted Paper 2 revision covering floating-point math, Boolean algebra simplification, SQL multi-table queries, subnetting, and structuring 12-mark AO1/AO2/AO3 essay responses.
- **Summer (Term 5) – Public Examinations & Synoptic Drop-in Support:** Final examination phase: Sitting AQA A-Level Paper 1 (On-Screen Skeleton & Theory - 2.5 hrs) and Paper 2 (Written Concepts - 2.5 hrs).