KEY STAGE 5 – YEAR 13 – A-Level Computer Science

CURRICULUM MAP

Autumn Term		Spring Term		Summer Term	
Autumn 1	**Autumn 2**	**Spring 1**	**Spring 2**	**Summer 1**	**Summer 2**
Key Themes	**Key Themes**	**Key Themes**	**Key Themes**	**Key Themes**	**Key Themes**
2.2 Problem solving and programming	**2.3 Algorithms** **3.4 Evaluation (20 marks)**	NEA Corrections and improvements	Exam Preparation NEA Corrections and improvements	Exam Preparation	
Assessment / Composite Tasks	**Assessment / Composite Tasks**	**Assessment / Composite Tasks**	**Assessment / Composite Tasks**	**Assessment / Composite Tasks**	**Assessment / Composite Tasks**
2.2 End on Unit Test	2.3 End on Unit Test Mock Exams	NEA	Mock Exams		

KEY STAGE 5 – YEAR 13 – A-Level Computer Science

SCHEME OF WORK

Autumn Half Term One: Key Theme – 2.2 Problem solving and programming					
	Intent (weekly outline)	Implementation (T and L Pedagogy/components used)	Impact	Powerful Knowledge (keywords and terminology)	Personal Development Links
Week 1	2.2.1 Programming techniques **3.4 Evaluation (20 marks)**	In this implementation, students apply key programming techniques such as sequence, iteration, and branching, using an IDE to develop and debug their programs. Additionally, computational methods like problem decomposition, divide and conquer, and abstraction are explored, with students applying advanced techniques like backtracking and data mining to solve real-world problems.	(a) Programming constructs: sequence, iteration, branching.	Sequence Iteration Branching Recursion Iterative Approach Global Variables Local Variables Modularity Functions Procedures Parameter Passing Pass By Value Pass By Reference Integrated Development Environment (IDE) Debugging Object-Oriented Programming (OOP) Classes Objects Inheritance Polymorphism	Personal development in computer science involves developing the ability to recognise problems and decompose them into manageable components, fostering critical thinking and problem-solving skills. By applying methods such as divide and conquer, abstraction, and advanced techniques like backtracking
Week 2			(b) Recursion, how it can be used and compares to an iterative approach.		
Week 3			(c) Global and local variables. (d) Modularity, functions and procedures, parameter passing by value and by reference. (e) Use of an IDE to develop/debug a program.		
Week 4			(f) Use of object oriented techniques.		
Week 5	2.2.2 Computational methods **3.4 Evaluation (20 marks)**		(a) Features that make a problem solvable by computational methods.	Computational Methods Problem Solvability Problem Recognition Problem Decomposition Divide and Conquer Abstraction	

Week 6			(b) Problem recognition. (c) Problem decomposition (d) Use of divide and conquer. (e) Use of abstraction. (f) Learners should apply their knowledge of: • backtracking • data mining • heuristics • performance modelling • pipelining	Backtracking Data Mining Heuristics Performance Modeling Pipelining Algorithm Design Recursive Approach Complexity Optimisation Solution Space Data Analysis Iterative Process Computational Efficiency Decision-Making	and data mining, learners enhance their ability to tackle complex challenges efficiently and effectively.
Week 7	**3.4 Evaluation (20 marks)**				

Autumn Half Term Two: Key Theme – 2.3 Algorithms					
	Intent (weekly outline)	Implementation (T and L Pedagogy/components used)	Impact	Powerful Knowledge (keywords and terminology)	Personal Development Links
Week 1	2.3.1 Algorithms	Students develop their understanding of algorithm design and efficiency through problem-based learning, where they analyse execution time and space complexity using Big O	(a) Analysis and design of algorithms for a given situation.	Algorithm Design Problem-Solving Efficiency Optimisation Computational Complexity	Personal development in computer science involves enhancing problem-solving and
Week 2	2.3.1 Algorithms		(b) The suitability of different algorithms for a given task and data set, in terms of execution time and	Execution Time Space Complexity Big O Notation Constant Time (O(1))	

Week 3	2.3.1 Algorithms	notation. Hands-on coding activities, visualisation tools, and peer discussions reinforce key concepts, allowing them to compare sorting, searching, and data structure algorithms in real-world scenarios.	space. (c) Measures and methods to determine the efficiency of different algorithms, Big O notation (constant, linear, polynomial, exponential and logarithmic complexity). (d) Comparison of the complexity of algorithms.	Linear Time (O(n)) Polynomial Time (O(n^k)) Exponential Time (O(2^n)) Logarithmic Time (O(log n)) Algorithm Efficiency Complexity Comparison	analytical skills by designing and evaluating algorithms for different situations. By understanding algorithm efficiency, complexity (Big O notation), and the suitability of various approaches like sorting algorithms and shortest path algorithms, learners develop logical thinking and the ability to optimize solutions for real-world applications.
Week 4	2.3.1 Algorithms		(e) Algorithms for the main data structures, (stacks, queues, trees, linked lists, depth-first (post-order) and breadth-first traversal of trees). (f) Standard algorithms (bubble sort, insertion sort, merge sort, quick sort, Dijkstra's shortest path algorithm, A* algorithm, binary search and linear search).	Stack Queue Tree Linked List Depth-First Search (DFS) Breadth-First Search (BFS) Post-Order Traversal Bubble Sort Insertion Sort Merge Sort Quick Sort Dijkstra's Algorithm A* Algorithm Binary Search Linear Search Sorting Algorithms Searching Algorithms Graph Traversal Pathfinding Algorithm Optimisation	
Week 5	**3.4 Evaluation (20 marks)**	For the NEA (Non-Examined Assessment), students will engage in	(a) Provide annotated evidence of testing the solution	Debugging Algorithm Efficiency Code Optimization Testing Framework	

Week 6	**3.4 Evaluation (20 marks)**	hands-on testing of their programming solutions, allowing them to refine their code and debug effectively. This practical approach reinforces problem-solving skills, encourages independent learning, and prepares students for real-world software development scenarios.	of robustness at the end of the development process. (b) Provide annotated evidence of usability testing (user feedback). (a) Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis. (a) Discuss the maintainability of the solution. (b) Discuss potential further development of the solution.	User Input Validation	
Week 7	**3.4 Evaluation (20 marks)**				

Spring Half Term One: Key Theme – NEA Corrections and improvements

	Intent (weekly outline)	Implementation (T and L Pedagogy/components used)	Impact	Powerful Knowledge (keywords and terminology)	Personal Development Links
Week 1	**3.1. Analysis of the problem (10 marks)**	The implementation of the **Programming Project** is supported through a **scaffolded approach**, where students progress from problem identification to designing, developing, testing, and evaluating	• Described and justified the features that make the problem solvable by computational methods, explaining why it is amenable to a computational approach. • Identified suitable stakeholders for the project	Problem Identification Software Development Algorithm Design Computational Thinking Project-Based Learning IDE (Integrated Development Environment) Debugging	The non-exam assessment (Programming Project) supports personal development by enhancing problem-

		their solution. **Project-based learning** encourages independence, while **regular teacher feedback, peer reviews, and debugging workshops** help refine their coding and problem-solving skills. Students use an **IDE for development and debugging**, applying **modular programming, object-oriented techniques, and data structures** to create efficient solutions. **Agile development principles** and **code reviews** are integrated to simulate real-world software development, enhancing their ability to plan, iterate, and optimise their final project.	and described them explaining how they will make use of the proposed solution and why it is appropriate to their needs. • Researched the problem in depth looking at existing solutions to similar problems, identifying and justifying suitable approaches based on this research. • Identified the essential features of the proposed computational solution explaining these choices. • Identified and explained with justification any limitations of the proposed solution. • Specified and justified the requirements for the solution including (as appropriate) any hardware and software requirements. • Identified and justified measurable success criteria for the proposed solution.	Object-Oriented Programming (OOP) Modular Programming Data Structures Agile Development Code Optimisation Testing and Evaluation Peer Review Code Documentation User Requirements Software Lifecycle Error Handling Version Control Real-World Application	solving, critical thinking, and independent learning skills, as students take ownership of designing, developing, and refining a complex software solution. It also fosters resilience and adaptability, as students troubleshoot errors, optimise their code, and apply computational thinking to real-world challenges.
Week 2	**3.2 Design of the solution (15 marks)**		Broken the problem down systematically into a series of smaller problems suitable for computational solutions, explaining and justifying the process.		

			• Defined in detail the structure of the solution to be developed. • Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem. • Described, justifying choices made, the usability features to be included in the solution. • Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) justifying and explaining any necessary validation. • Identified and justified the test data to be used during the iterative development of the solution. • Identified and justified any further data to be used in the post development phase.		
Week 3	**3.3 Developing the solution (25 marks)**		• Broken the problem down systematically into a series of smaller problems suitable for computational solutions, explaining and justifying the process.		

			• Defined in detail the structure of the solution to be developed. • Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem. • Described, justifying choices made, the usability features to be included in the solution. • Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) justifying and explaining any necessary validation. • Identified and justified the test data to be used during the iterative development of the solution. • Identified and justified any further data to be used in the post development phase.		
Week 4	**3.3 Developing the solution (25 marks)**		• Provided evidence of each stage of the iterative development process for a coded solution relating this to the break down of the problem		

			from the analysis stage and explaining what they did and justifying why. • Provided evidence of prototype versions of their solution for each stage of the process. • The solution will be well structured and modular in nature. • Code will be annotated to aid future maintenance of the system. • All variables and structures will be appropriately named. • There will be evidence of validation for all key elements of the solution. • The development will show review at all key stages in the process.• Provided evidence of testing at each stage of the iterative development process. • Provided evidence of any failed tests and the remedial actions taken with full justification for any actions taken.		

Week 5	**3.4 Evaluation (20 marks)**		• Provided annotated evidence of post development testing for function and robustness. • Provided annotated evidence for usability testing.		
Week 6	**3.4 Evaluation (20 marks)**		• Used the test evidence to cross reference with the success criteria to evaluate the solution explain how the evidence shows that the criteria has been fully, partially or not met in each case. • Provided comments on how any partially or unmet criteria could be addressed in further development. • Provided evidence of the usability features justifying their success, partial success or failure as effective usability features. • Provided comments on how any issues with partially or unmet usability features could be addressed in further development. • Considered maintenance issues and limitations of the solution.		

			• Described how the program could be developed to deal with limitations and potential improvements / changes. • There is a well developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.		

Spring Half Term Two: Key Theme – Exam Prep

	Intent (weekly outline)	Implementation (T and L Pedagogy/components used)	Impact	Powerful Knowledge (keywords and terminology)	Personal Development Links
Week 1	Sixth Form Mocks	For exam preparation, students will regularly complete past OCR A-Level examination papers under timed conditions to simulate the real exam environment. This approach helps develop exam techniques, improve time management, and familiarise students with the format and types of questions they will encounter.	The use of past examination papers enhances students' exam readiness by improving their time management and understanding of question formats. This targeted practice boosts their confidence and ensures they are well-prepared for the final exam.	Algorithm Data Structures Computational Thinking Big O Notation Recursion Boolean Logic Software Development Life Cycle (SDLC) Machine Code Cybersecurity Databases (SQL)	Problem-Solving Skills: Engaging with algorithms, coding tasks, and real-world scenarios helps students develop critical thinking and adaptability, which are transferable skills beyond the subject. Time Management
Week 2	Go through Mock Papers				
Week 3	May 2022 Paper 1				
Week 4	May 2022 Paper 2				
Week 5	May 2019 Paper 1				
Week 6	May 2019 Paper 2				

					and Self-Discipline: Working with past exam papers and preparing for assessments teaches students to manage their time effectively, enhancing their organizational skills. Collaboration and Communication: Group projects or peer discussions around complex topics like cybersecurity or database management promote teamwork and the ability to articulate technical concepts clearly.

Summer Half Term One: Key Theme – Exam Prep

KEY STAGE 5 – YEAR 13 – A-Level Computer Science

	Intent (weekly outline)	Implementation (T and L Pedagogy/components used)	Impact	Powerful Knowledge (keywords and terminology)	Personal Development Links
Week 1	Nov 2021 Paper 1	For exam preparation, students will regularly complete past OCR A-Level examination papers under timed conditions to simulate the real exam environment. This approach helps develop exam techniques, improve time management, and familiarise students with the format and types of questions they will encounter.	The use of past examination papers enhances students' exam readiness by improving their time management and understanding of question formats. This targeted practice boosts their confidence and ensures they are well-prepared for the final exam.	Algorithm Data Structures Computational Thinking Big O Notation Recursion Boolean Logic Software Development Life Cycle (SDLC) Machine Code Cybersecurity Databases (SQL)	Problem-Solving Skills: Engaging with algorithms, coding tasks, and real-world scenarios helps students develop critical thinking and adaptability, which are transferable skills beyond the subject. Time Management and Self-Discipline: Working with past exam papers and preparing for assessments teaches students to manage their time effectively, enhancing their organizational skills.
Week 2	Nov 2021 Paper 2				
Week 3	Nov 2020 Paper 1				
Week 4	Nov 2020 Paper 2				
Week 5	Summ				
Week 6					

					Collaboration and Communication: Group projects or peer discussions around complex topics like cybersecurity or database management promote teamwork and the ability to articulate technical concepts clearly.