Computer Science	Working towards expected outcomes Your child is not yet making the expected progress within this course.	Working at expected outcomes Your child is achieving the expected progress for this point within the course.	Working beyond expected outcomes Your child is exceeding the expected progress.
Year 13	Students working **towards** expected outcomes in Year 13 can:	Students working **at** expected in Year 13 can:	Students working **beyond** expected in Year 13 can:
Autumn 1 Data Structures	Arrays: Understands arrays store multiple values but struggles to manipulate them using indices. Tuples and Records: Knows that tuples and records group data but needs support in using them effectively. Queues: Can describe a queue as FIFO but struggles to implement and apply it. Lists: Understands lists store ordered data but struggles with different list types and operations. Stacks: Knows stacks are LIFO but needs help with implementing basic operations. Hash Tables: Aware that hash tables map keys to values but struggles with hashing and collisions. Graphs: Can identify graph components but struggles with types and algorithms. Trees: Knows trees use nodes and edges but needs guidance with practical tree operations.	Arrays: Can use arrays to store and manipulate data (e.g., sorting, searching). Tuples and Records: Can use tuples and records for grouping and organising related data. Queues: Can implement queues and use enqueue/dequeue in practical tasks. Lists: Can implement and manipulate singly/doubly linked lists (insertion, deletion). Stacks: Can implement stacks and apply LIFO in tasks like reversing strings or balancing parentheses. Hash Tables: Can implement hash tables, understand hashing, and handle collisions. Graphs: Can use graphs, represent them with adjacency lists, and apply BFS/DFS. Trees: Can implement binary trees and basic traversal (pre/in/post-order).	Arrays: Expert in using arrays, optimising for space and performance. Tuples and Records: Can model complex data structures with tuples/records. Queues: Can implement advanced queues (priority, circular) in real-world scenarios. Lists: Expert in advanced list types (e.g., skip lists) and optimising list operations. Stacks: Applies stacks in complex algorithms like DFS and expression parsing. Hash Tables: Expert in hash table optimisation, collision resolution, and real-world applications. Graphs: Deep understanding of graphs, applying advanced algorithms (Dijkstra, A*). Trees: Expert in balanced trees (AVL, B-trees) and optimising search operations.

Autumn 2 Boolean Algebra	Logic Gates: Can identify basic logic gates (AND, OR, NOT) but struggles to explain their behaviour or use in circuits. Boolean Expressions: Can write simple Boolean expressions but needs support simplifying or applying them in practical problems. Karnaugh Maps: Aware that Karnaugh maps are used to simplify expressions but struggles with filling them out or simplifying the expression. Adders: Can describe what an adder does but needs help understanding how they work in binary addition. D-type Flip-flops: Aware of the concept of flip-flops but struggles to explain their function or how they are used in sequential circuits.	Logic Gates: Can explain and apply basic logic gates in simple circuit diagrams and Boolean expressions. Boolean Expressions: Can simplify basic Boolean expressions using Boolean laws and apply them to solve problems. Karnaugh Maps: Can complete Karnaugh maps for expressions with 2-3 variables and simplify the Boolean expression. Adders: Understands the operation of half-adders and full-adders and can use them in basic binary addition problems. D-type Flip-flops: Understands the function of D-type flip-flops and can explain how they store a single bit of data in sequential circuits.	Logic Gates: Expert in using a range of logic gates (AND, OR, NOT, NAND, NOR, XOR) to design and analyse complex circuits and Boolean expressions. Boolean Expressions: Proficient in simplifying complex Boolean expressions using advanced techniques (e.g., Boolean theorems, De Morgan's laws) and applying them in circuit design. Karnaugh Maps: Can use Karnaugh maps for expressions with more than 4 variables and can optimise Boolean expressions efficiently. Adders: Can design and implement complex adders (e.g., 4-bit binary adders, carry-lookahead adders) and explain their operation in multi-bit addition. D-type Flip-flops: Deep understanding of D-type flip-flops and their application in sequential circuits, registers, and memory elements, including designing state machines.
Spring 1 Legal and Cultural Issues	Computing-related Legislation: Aware of basic computing laws (e.g., Data Protection Act) but struggles to explain their purpose or real-world applications. Ethical, Moral, and Cultural Issues: Can identify ethical or cultural concerns related to computing (e.g., social media usage) but needs support in understanding different perspectives. Privacy and Censorship: Aware that privacy and censorship are important issues in	Computing-related Legislation: Can explain key computing-related laws (e.g., GDPR, Computer Misuse Act) and their impact on society, businesses, and individuals. Ethical, Moral, and Cultural Issues: Understands and can discuss ethical dilemmas in computing (e.g., AI ethics, digital divide) and their moral and cultural implications. Privacy and Censorship: Can explain the concepts of privacy and censorship in digital spaces and how they affect users, businesses, and governments. Can discuss real-world	Computing-related Legislation: Deep understanding of a wide range of computing laws (e.g., intellectual property, freedom of information, international data laws) and can analyse their global impact and challenges. Ethical, Moral, and Cultural Issues: Critically evaluates ethical, moral, and cultural issues in computing, considering diverse viewpoints and proposing solutions to challenges (e.g., AI bias, digital rights). Privacy and Censorship: Can analyse the balance between privacy, censorship, and freedom of speech, and understand the implications of modern challenges

	computing but struggles to explain their impact or balance.	examples (e.g., data breaches, content moderation).	(e.g., government surveillance, data protection) on global and individual levels.
Spring 2 Computational Thinking	Thinking Abstractly: Can identify simple patterns or solutions but struggles to generalise or apply them to new problems. Thinking Ahead: Can think about the next steps in a task but struggles to plan multiple steps ahead or anticipate potential issues. Thinking Procedurally: Can follow basic instructions but struggles with creating step-by-step procedures for complex tasks. Thinking Logically: Can apply basic logic to problems but needs support in organising and structuring solutions. Thinking Concurrently: Aware that multiple tasks can run at the same time but struggles to understand or manage concurrent processes. Problem Recognition: Can recognise simple problems but needs support identifying the full scope or breaking it down into smaller parts. Problem Solving: Can solve simple problems but struggles with applying problem-solving strategies to more complex issues.	Thinking Abstractly: Can identify patterns and generalise solutions to new problems, simplifying complex issues. Thinking Ahead: Can plan ahead, anticipating challenges and considering multiple steps for problem-solving. Thinking Procedurally: Can create and follow clear step-by-step instructions for complex tasks or algorithms. Thinking Logically: Can apply logical reasoning to break down problems, organise thoughts, and structure solutions. Thinking Concurrently: Understands the concept of concurrency and can apply it to problems involving multiple tasks running in parallel. Problem Recognition: Can identify the core problem in a complex scenario and break it down into manageable sub-problems. Problem Solving: Can solve problems using structured approaches (e.g., algorithms, flowcharts) and apply solutions to new situations.	Thinking Abstractly: Deeply understands abstraction and can create models or frameworks to represent complex systems, recognising relationships and dependencies. Thinking Ahead: Can predict multiple outcomes and potential risks, creating detailed plans for complex problems while accounting for uncertainties. Thinking Procedurally: Can design sophisticated algorithms and systems, breaking problems into manageable procedures that are efficient and scalable. Thinking Logically: Applies advanced logic and reasoning to solve intricate problems, optimising processes and identifying patterns that are not immediately obvious. Thinking Concurrently: Expert in handling complex concurrent tasks and processes, understanding synchronisation, and efficiently managing resources. Problem Recognition: Can quickly and accurately identify problems in complex systems, anticipate challenges, and define solutions effectively. Problem Solving: Demonstrates exceptional problem-solving skills, applying advanced techniques (e.g., decomposition, algorithm design) and creating innovative solutions for real-world issues.