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 working beyond the expected progress for this point within the course.
Year 7	Students working **towards** expected outcomes in Year 7 can:	Students working **at** expected in Year 7 can:	Students working **beyond** expected in Year 7 can:
Autumn 1 E-Safety and Skills	Can identify some examples of safe or unsafe online behaviours but struggles to explain risks. Needs support navigating digital tools and may find basic tasks (saving files, using cloud tools, organising work) challenging. Beginning to recognise online threats such as scams or harmful content but cannot yet explain how to respond safely. Requires guidance to follow rules for creating secure passwords or protecting personal information.	Can explain safe online behaviour and identify common online risks (privacy issues, cyberbullying, scams). Uses school digital platforms confidently (file organisation, online tools, typing and formatting skills). Understands how to create secure passwords and protect personal data. Can evaluate online content for reliability with teacher prompts.	Confidently explains a wide range of online risks and how to avoid or report them. Independently applies strong digital organisation and productivity skills across tools/platforms. Can evaluate the trustworthiness of online sources and explain how misinformation spreads. Provides mature, informed advice to others on safe and responsible online behaviour.
Autumn 2 Bebras and Binary	Shows basic understanding during Bebras practice but struggles to apply logical reasoning or choose appropriate problem-solving strategies. Finds the Bebras competition challenging and requires guidance to interpret questions or identify patterns. Can recall that computers have evolved over time but struggles to explain key milestones in the history of computing.	Applies computational thinking (pattern recognition, logical reasoning, decomposition) to solve Bebras challenge questions with growing confidence. Participates effectively in the Bebras competition, demonstrating logical strategies and accurate reasoning. Can describe key developments in the history of computing and explain how technology has changed over time.	Creates a polished, well-structured Canva presentation that shows strong digital design skills, effective communication, and excellent research quality. Approaches Bebras problems using advanced computational thinking strategies such as abstraction or optimisation, demonstrating high-level reasoning. Performs exceptionally in the Bebras competition, showing resilience, flexibility, and creativity in problem-solving. Gives detailed explanations of major events in the history of computing and can link them to modern technologies or current trends.

	Beginning to understand that computers use binary but finds it difficult to convert numbers or explain binary place value. Requires step-by-step support to perform binary addition and often makes place-value errors.	Understands binary as a base-2 system and can convert small binary numbers to decimal with accuracy. Can complete binary addition tasks correctly when numbers are straightforward (e.g., no multiple carries).	Converts binary numbers confidently and can explain binary place value in detail, including how computers store and process data. Performs binary addition accurately even with more complex numbers, including multiple carries. Researches multiple computing pioneers and presents insightful explanations of how diverse individuals have shaped technological innovation.
Spring 1 Algorithms and Scratch	Can describe an algorithm at a basic level but struggles to explain how algorithms are used to solve problems. Attempts flowcharts but needs significant guidance to choose appropriate symbols or show the correct order of steps. Can create simple Scratch scripts using sequence but finds it difficult to incorporate selection or iteration independently. Requires support to use broadcast messages and events to coordinate actions between sprites. Finds debugging challenging and needs help identifying where a program is not working as intended. Needs guidance to break down a problem into smaller parts before starting a project.	Can explain what an algorithm is and use flowcharts to represent simple processes with correct symbols and order. Creates Scratch programs that use sequence, selection (if/else), and iteration (loops) effectively in appropriate situations. Uses broadcast and receive blocks to control multiple sprites or trigger events within a project. Can design a simple project by breaking it down into steps using decomposition. Tests programs systematically and can identify and fix straightforward errors. Produces interactive Scratch projects with clear structure and logical flow.	Gives clear explanations of algorithms and can design flowcharts for more complex processes, showing strong logical thinking. Creates well-structured Scratch programs that combine sequence, selection, iteration, and broadcasts creatively to produce sophisticated behaviours. Demonstrates strong problem-solving skills by breaking complex tasks into manageable components independently. Debugs efficiently, identifying logic errors and optimising code for clarity or performance. Adds enhanced features beyond the requirements (e.g., multiple sprites interacting, multi-stage gameplay, scoring systems, or custom blocks). Reflects thoughtfully on how to improve algorithms or program structure, showing mature understanding of computational thinking principles.
Spring 2	Can use basic Scratch blocks but struggles to apply sequence, selection, or iteration correctly without guidance.	Designs and builds a functional Scratch game that uses sequence, selection, iteration, and events to create interactive gameplay.	Develops a polished, creative Scratch game that uses complex interactions, multiple sprites, variables, and refined logic.

Scratch and Components	Finds it difficult to design a coherent game idea or break it into manageable steps. Struggles to use event handling effectively, leading to limited interactivity in their project. Needs support to debug their game or identify why certain behaviours aren't working. Can name a few computer components (e.g., keyboard, monitor) but struggles to explain their purpose or differentiate input vs output devices. Finds it difficult to understand how internal components such as the CPU, RAM, and storage contribute to system performance. Requires guidance to connect learning about hardware to how software runs on a computer.	Can break a game idea into steps using decomposition and implement these steps effectively. Uses event handling (e.g., broadcasts, key presses, collisions) to control game flow and sprite interactions. Debugs programs with growing independence, testing features and making corrections. Understands the purpose of key computer components such as the CPU, RAM, storage, motherboard, input and output devices. Can categorise devices as input or output and explain their role in a computer system. Understands in simple terms how internal components work together to run programs and process data.	Incorporates advanced features such as custom blocks, scoring systems, timers, level progression, or high-quality animations. Shows strong independence in debugging and enhancing their game, improving both functionality and user experience. Gives detailed explanations of hardware components, including how CPU speed, RAM capacity, storage type, and buses affect system performance. Demonstrates understanding of how hardware and software interact, describing how instructions are fetched, stored, and executed. Makes clear connections between system components and real-world devices, showing emerging GCSE-level reasoning. Reflects critically on both their game design and hardware knowledge, suggesting informed improvements.
<u>Summer 1</u> Encryption and Viruses	Can explain at a basic level that encryption is used to protect information but struggles to describe how ciphers work. Finds it difficult to apply the Caesar or Vigenère cipher without step-by-step guidance. Has limited ability to spot patterns in encoded messages and needs support to begin decrypting information.	Understands why encryption is important and can explain how simple ciphers such as Caesar and Vigenère work. Can encrypt and decrypt messages using both ciphers with developing confidence. Uses logical reasoning, pattern recognition, and trial-and-error techniques to solve codebreaking challenges.	Gives clear, detailed explanations of how different cipher techniques work and can compare their strengths and weaknesses. Encrypts and decrypts messages independently and can tackle more complex variations or multi-step ciphers. Demonstrates advanced logical reasoning and strong pattern-spotting skills when solving codebreaking challenges.

	Struggles to use logical reasoning to break down clues in the mystery investigation and may need frequent prompts. Can name some types of malware (e.g., viruses) but struggles to explain how they spread or the risks they pose. Needs support to understand safe computing practices and how to protect devices from security threats.	Works systematically during the mystery investigation to piece together clues and form sensible conclusions. Understands the differences between common types of malware such as viruses, worms, and trojans Can explain basic steps to stay safe online and protect devices using good cybersecurity practices.	Approaches the mystery investigation with creativity and persistence, explaining how clues connect and evaluating multiple possibilities. Provides mature explanations of how malware operates, including infection vectors, payloads, and prevention strategies. Shows strong awareness of cybersecurity principles and can advise others on safe computing behaviour, linking concepts to real-world examples.
<u>Summer 2</u> Python Turtle	Can write simple Python statements but struggles to use loops or functions without step-by-step guidance. Finds it difficult to control the turtle library accurately and may produce shapes that do not match the intended design. Has limited understanding of how sequencing and iteration work together to draw patterns or repeated structures. Needs support to break down a design into smaller coding steps and often becomes stuck when errors occur. Struggles to understand how functions can make code more efficient or reusable.	Writes Python code that makes effective use of sequencing and iteration to control the turtle graphics tool. Can draw a range of shapes and patterns by using loops to repeat instructions efficiently. Breaks down a design into clear coding steps and can independently correct simple errors. Understands how functions work and can create simple functions to avoid repeating code. Produces a mini art project that demonstrates creativity, planning, and logical structure.	Creates complex and visually impressive turtle graphics using nested loops, functions, parameters, and well-structured code. Demonstrates strong decomposition skills, breaking ambitious designs into manageable components and solving problems independently. Writes clean, modular, and efficient Python code, showing early GCSE-level thinking in structure and organisation. Enhances the mini art project with original ideas, experimenting with colour, movement, or mathematical patterns. Gives detailed explanations of how functions improve efficiency and how iteration can generate complex designs.