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 10	Students working **towards** expected outcomes in Year 10 can:	Students working **at** expected in Year 10 can:	Students working **beyond** expected in Year 10 can:
Autumn 1 Algorithms Python Coding	Algorithms & Decomposition: Begin to recognise and break down simple problems into smaller parts, using pseudocode and flowcharts with guidance. Programming Concepts: Use IF/ELSE, For Loops, While Loops, and Lists in Python but needs help with troubleshooting and optimisation. Searching & Sorting: Start to use basic searching (e.g., linear search) and sorting algorithms (e.g., bubble sort) with some assistance. Abstraction & Flowcharts: Begin to understand abstraction and can create basic flowcharts but needs help with complex problems.	Algorithms & Decomposition: Can decompose problems into smaller parts and create pseudocode or flowcharts with minimal errors. Programming Concepts: Demonstrate proficiency in using IF/ELSE, For Loops, While Loops, and Lists in Python, and can debug code with little help. Searching & Sorting: Implement basic searching and sorting algorithms and understand basic efficiency concepts. Abstraction & Flowcharts: Use abstraction effectively and can create and translate flowcharts into working Python code.	Algorithms & Decomposition: Decompose complex problems and create clear pseudocode and flowcharts with minimal assistance. Programming Concepts: Show high levels of skill in using advanced Python features and optimising code for efficiency. Searching & Sorting: Understand and apply advanced searching and sorting algorithms, analysing their efficiency. Abstraction & Flowcharts: Abstract complex problems and create optimised, efficient algorithms with clear flowcharts and code.
Autumn 2 Data Representation	Number Bases & Conversions: Start to understand binary and hexadecimal systems; needs support converting between different bases. Units of Information: Evidence a basic understanding of bits, bytes, and simple units like KB, MB, but may struggle with larger units.	Number Bases & Conversions: Convert between binary, decimal, and hexadecimal and understands the process well. Units of Information: Understand bits, bytes, and common data units (KB, MB, GB) and can apply them in simple problems. Binary Arithmetic: Perform binary addition, subtraction, and multiplication confidently.	Number Bases & Conversions: Easily convert between all major number bases (binary, decimal, hexadecimal) and applies conversions in complex contexts. Units of Information: Fully understand all data units (bits, bytes, KB, MB, GB) and can calculate and convert between them accurately. Binary Arithmetic: Confidently perform advanced binary arithmetic (including multiplication, division, and bit-shifting).

	Binary Arithmetic: Perform simple binary addition or subtraction but needs help with more complex operations. ASCII & Unicode: Understand ASCII characters but may struggle with Unicode and its broader character set. Images & Sound: Begin to understand how images and sound are represented digitally but lacks full understanding of their encoding. Compression: Demonstrate an awareness of data compression and is developing their knowledge of how or why it is used in practice. Programming (Subroutines & Parameters): Can define simple subroutines but struggles with passing parameters and understanding their use.	ASCII & Unicode: Convert between ASCII and Unicode, understanding their differences and uses. Images & Sound: Understand how images and sound are represented digitally and can explain basic encoding techniques. Compression: Understand basic data compression methods and their uses for reducing file sizes. Programming (Subroutines & Parameters): Define subroutines and pass parameters effectively to modularise code.	ASCII & Unicode: Demonstrate in-depth knowledge of ASCII and Unicode, including extended characters and character encoding schemes. Images & Sound: Understand the detailed digital representation of images and sound, including pixel data, colour models, and sound sampling. Compression: Evidence a deep understanding of compression algorithms and can compare lossless vs. lossy methods and apply them in different scenarios. Programming (Subroutines & Parameters): Use subroutines and parameters to write modular, efficient code and can handle complex parameter passing (e.g., by reference or value).
Spring 1 Hardware and Software	Hardware & Software: Identify basic hardware components and understand the difference between hardware and software but needs support with more complex examples. Boolean Logic: Begin to understand AND, OR, NOT operations but struggles with applying them in problems. Systems & Application Software: Distinguish between system and application software but is unclear on their specific functions.	Hardware & Software: Understands and can describe the roles of hardware components and software types in a computer system. Boolean Logic: Apply basic Boolean logic to solve simple problems and understand truth tables. Systems & Application Software: Explain the differences between system and application software with examples (e.g., OS vs. word processors).	Hardware & Software: Evidence a deep understanding of hardware components and the relationship between software and hardware, including peripheral devices and their functions. Boolean Logic: Evidence a fluency in using Boolean logic in complex scenarios, including simplification of Boolean expressions and applying them to programming problems. Systems & Application Software: Explain in-depth the functions and purposes of system software (e.g., OS, utilities) vs. application software (e.g., word processors, games).

	Operating Systems (OS): Show awareness of what an OS does but needs help understanding its core functions like managing memory or running programs. Classification of Languages & Translators: Demonstrate they know the difference between high-level and low-level languages but needs assistance understanding the role of translators (compilers, interpreters). Architecture: Describe basic components of a computer system (CPU, memory) but struggles with understanding how they interact. Python Programming (Validation & Authentication): Use basic validation checks (e.g., checking for numbers or strings) but needs support with authentication methods.	Operating Systems (OS): Understands and apply key OS functions, including memory management, multitasking, and file management. Classification of Languages & Translators: Classify programming languages (high-level vs. low-level) and explain the functions of compilers and interpreters. Architecture: Explain the roles of the CPU, memory, and other components in a computer's architecture. Python Programming (Validation & Authentication): Implement validation checks (e.g., checking input formats) and understands basic authentication methods (e.g., password validation).	Operating Systems (OS): Analyse and explain the core principles of operating systems, including processes, memory management, and file systems. Classification of Languages & Translators: Fully understands the classifications of programming languages, the differences between compilers and interpreters, and can explain how translators work. Architecture: Explain and analyse the interactions between CPU, memory, storage, and input/output devices, understanding their roles in the computer architecture. Python Programming (Validation & Authentication): Implement advanced validation techniques and understand secure authentication methods, including user authentication and encryption.
Spring 2 Networks	PAN, LAN, WAN, Wired/Wireless: Identify the basic differences between PAN, LAN, and WAN and understands the concepts of wired and wireless connections but needs help with specific examples. Bus/Star Topologies: Describe bus and star topologies but struggles with their advantages and disadvantages in different network scenarios. Protocols: Show an awareness of protocols but unsure of how they work in different networking contexts (e.g., HTTP, FTP).	PAN, LAN, WAN, Wired/Wireless: Explain the differences between PAN, LAN, and WAN and can give examples of wired and wireless connections in real-world contexts. Bus/Star Topologies: Explain both bus and star topologies, their advantages, and drawbacks, and can apply them to basic network designs. Protocols: Identify common protocols (e.g., HTTP, FTP) and understand their role in transmitting data across networks. Security: Demonstrate and apply common security concepts (e.g., encryption, firewalls)	PAN, LAN, WAN, Wired/Wireless: Analyse and compare the benefits and limitations of PAN, LAN, and WAN in different network scenarios and understand the trade-offs between wired and wireless connections. Bus/Star Topologies: Assess and choose the best topology (bus, star, etc.) for specific networking needs and explain why. Protocols: Evidence and apply a deep understanding of network protocols and can explain how protocols (e.g., HTTP, FTP, TCP/IP) interact to ensure secure and efficient communication.

	Security: Understand the basic idea of security but needs help with specific security methods or tools. TCP/IP: Show an awareness of TCP/IP and its role in networking. Python Programming (Purpose of Algorithms): Can identify basic algorithms but struggles with determining their purpose or application in different situations.	and can describe methods to protect networks and data. TCP/IP: Understand the layers of TCP/IP and can explain its role in transmitting data across networks. Python Programming (Purpose of Algorithms): Identify the purpose of simple algorithms and explain how they solve problems or improve processes.	Security: Apply advanced security measures (e.g., encryption, multi-factor authentication) and explain how to protect networks from threats. TCP/IP: Explain the function of each layer in the TCP/IP model and how data is transmitted, including IP addressing and packet routing. Python Programming (Purpose of Algorithms): Evidence and determine the purpose of algorithms in complex situations and explain how different algorithms can be applied to solve a variety of problems efficiently.
Summer 1 & 2 Cyber-Security	Threats: Aware of basic cybersecurity threats but struggles to identify specific types or understand their impact. Social Engineering: Recognise basic forms of social engineering (e.g., phishing) but needs help identifying more complex tactics. Penetration Testing: Show a basic understanding of penetration testing but needs further explanation of its methods and tools. Malware: Identify common types of malware (e.g., viruses, Trojans) but is unclear on how they spread or how to protect against them. Detecting Threats: Demonstrate an Awareness of some methods for detecting threats but struggles with implementing or using detection tools. Python Programming (Errors and Testing): Identify common syntax errors and run basic	Threats: Understand common cybersecurity threats (e.g., viruses, ransomware) and their potential impact on systems and data. Social Engineering: Identify common social engineering tactics (e.g., phishing, baiting) and explain how they work to manipulate users. Penetration Testing: Evidence and demonstrate an understanding of the purpose of penetration testing and can explain some basic tools and techniques used in ethical hacking. Malware: Identify different types of malware, understand their behaviour, and know how to mitigate risks (e.g., using antivirus software). Detecting Threats: Understand basic methods of threat detection (e.g., antivirus scans, network monitoring) and can apply them to identify vulnerabilities. Python Programming (Errors and Testing): Debug simple errors and use basic testing techniques	Threats: Has a deep understanding of a wide range of cybersecurity threats (e.g., advanced persistent threats, zero-day vulnerabilities) and can assess their potential risks. Social Engineering: Can identify and defend against sophisticated social engineering techniques and explain the psychological tactics used to exploit individuals. Penetration Testing: Understand advanced penetration testing techniques and tools and can conduct simulated attacks to assess vulnerabilities. Malware: Analyse the behaviour of different types of malware and design strategies for prevention, detection, and response. Detecting Threats: Expert at using advanced threat detection tools (e.g., intrusion detection systems) and can analyse data to identify and respond to potential threats. Python Programming (Errors and Testing): Troubleshoot complex code issues, write efficient test cases, and

	tests but needs support in debugging and testing more complex code.	(e.g., unit tests) to ensure code works as expected.	implement automated testing to ensure reliability and performance.