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 12	Students working **towards** expected outcomes in Year 12 can:	Students working **at** expected in Year 12 can:	Students working **beyond** expected in Year 12 can:
Autumn 1 Components of a Computer	Processor Components: Can identify basic components of a processor (e.g., ALU, CU, registers) but requires support in explaining their functions and interactions. Processor Performance: Has a basic understanding of how processor speed (clock speed, cache size) affects performance but struggles to explain the impact of different factors on overall performance. Types of Processor: Aware of the basic types of processors (e.g., single-core, multi-core) but lacks detailed knowledge of their advantages, disadvantages, and appropriate use cases. Input Devices: Can list common input devices (e.g., keyboard, mouse) but needs support in explaining how they work or their role in computing systems. Output Devices: Aware of basic output devices (e.g., monitor, printer) but struggles to explain their functionality in detail or their impact on user interaction. Storage Devices: Can identify common storage devices (e.g., HDD, SSD) but needs	Processor Components: Can explain the function of key processor components (e.g., ALU, CU, registers) and their role in executing instructions, with some confidence in how they work together. Processor Performance: Understands the factors influencing processor performance (e.g., clock speed, number of cores, cache size) and can discuss how these affect computational efficiency and speed. Types of Processor: Can explain the differences between processor types (e.g., single-core vs. multi-core) and understand their suitability for various tasks, such as gaming vs. multitasking. Input Devices: Can explain the function and use of various input devices and how they communicate with the computer (e.g., keyboard scanning, mouse tracking). Output Devices: Understands how output devices (e.g., monitors, printers) work, can explain resolution and colour depth for displays, and knows how output quality is influenced by the device. Storage Devices: Can compare different storage devices (e.g., HDD, SSD, optical media),	Processor Components: Has a detailed understanding of processor components and can analyse how different parts of the processor (e.g., control unit, ALU, cache) optimise instruction execution and overall system performance. Processor Performance: Expert in explaining how multiple factors (e.g., cache size, number of cores, clock speed) contribute to processor performance, including more complex concepts like pipelining and overclocking. Types of Processor: Can critically assess the use of different processor types (e.g., ARM, x86, multi-core processors) and their specific advantages in various computing environments (e.g., mobile vs. desktop vs. server). Input Devices: Can explain the technical details of input devices, including advanced input methods (e.g., biometric scanning, voice recognition) and their implications for user experience and security. Output Devices: Fully understands the technical aspects of output devices, including advanced concepts like resolution, refresh rates, and how the choice of output device impacts performance and user interaction. Storage Devices: Can compare modern storage solutions (e.g., cloud storage,) and explain their impact on

	help explaining the differences in speed, capacity, and suitability for various uses.	explaining the advantages and disadvantages of each in terms of speed, cost, and durability.	performance, data redundancy, and cost, and how storage devices fit into large-scale computing systems.
Autumn 2 System Software	OS Functions: Aware of the basic functions of an OS (e.g., managing hardware, providing user interface) but struggles to explain how these functions interact to manage system resources effectively. Types of OS: Can identify some types of operating systems (e.g., Windows, Linux) but needs support in explaining their specific features, uses, and differences. Nature of Applications: Understands that applications are software designed for specific tasks but needs guidance in understanding the relationship between system software and application software. Programming Language Translators: Knows that translators are used to convert high-level code to machine code but struggles to explain the differences between compilers, interpreters, and assemblers.	OS Functions: Can explain the key functions of an OS (e.g., managing memory, controlling I/O devices, providing security) and how the OS acts as an intermediary between the hardware and software. Types of OS: Understands the differences between various types of OS (e.g., single-user vs multi-user, real-time vs general-purpose) and can explain their use cases and characteristics. Nature of Applications: Can explain the relationship between system software and application software, and how applications rely on OS resources to function properly. Programming Language Translators: Can distinguish between compilers, interpreters, and assemblers, explaining how each one works and when each is used in the development process.	OS Functions: Has an in-depth understanding of OS functions, including memory management techniques (e.g., paging, segmentation), process scheduling algorithms, and handling interrupts and exceptions. Types of OS: Can critically assess the strengths and weaknesses of different OS types (e.g., embedded OS vs desktop OS, real-time OS vs general-purpose OS) and explain how these choices affect system performance and use cases. Nature of Applications: Fully understands the relationship between system software, application software, and how the OS provides services like file management, process control, and resource allocation for efficient application execution. Programming Language Translators: Can evaluate the advantages and disadvantages of compilers, interpreters, and assemblers, and explain their impact on code execution time, debugging, and portability across different systems.
Spring 1 Software Development	Systems Analysis Methods: Can identify basic stages of system development (e.g., analysis, design, implementation) but struggles with applying formal methods like flowcharts or DFDs to real-world scenarios. Writing and Following Algorithms: Understands the basic structure of an algorithm (e.g., sequence, selection, iteration) but requires guidance in writing and optimising algorithms for specific problems.	Systems Analysis Methods: Can explain and apply common system analysis methods (e.g., flowcharts, data flow diagrams) and understands their role in documenting and improving systems during development. Writing and Following Algorithms: Can write and follow algorithms using basic structures like loops and conditionals, and has begun optimising them for efficiency (e.g., reducing complexity).	Systems Analysis Methods: Can critically assess and apply advanced systems analysis methods (e.g., use case diagrams, entity-relationship models) to complex system design, identifying bottlenecks and optimisation opportunities. Writing and Following Algorithms: Proficient in writing efficient and optimised algorithms, demonstrating strong problem-solving skills, and applying algorithmic techniques (e.g., divide and conquer, dynamic programming) to solve complex problems.

	Programming Paradigms: Aware of different programming paradigms (e.g., procedural, object-oriented) but needs help in explaining how they differ and their practical applications. Assembly Language: Has a basic understanding of assembly language but struggles to write and interpret simple programs or understand the relationship between assembly and machine code.	Programming Paradigms: Understands and can explain the key differences between programming paradigms (e.g., procedural vs object-oriented) and is able to apply them when writing code. Assembly Language: Understands the basic syntax and structure of assembly language and can write simple programs to manipulate registers, memory, and perform basic I/O operations.	Programming Paradigms: Has a deep understanding of multiple programming paradigms and can evaluate the strengths and weaknesses of each, applying them appropriately to different types of projects (e.g., object-oriented for large-scale systems, functional for concurrent programming). Assembly Language: Can write and interpret complex assembly language programs, understanding the interaction between high-level languages and machine code, and optimising assembly code for performance.
Spring 2 Exchanging Data	Compression and Encryption: Aware of basic concepts of data compression and encryption but struggles to explain how they are applied in real-world scenarios (e.g., zipping files, securing communications). Database Concepts: Can identify basic database elements (e.g., tables, fields, records) but needs support in explaining how databases are used to store and manage data. Relational Databases and Normalisation: Understands that relational databases use tables to store data but has difficulty explaining the process of normalisation and its importance. SQL: Can write simple SQL queries (e.g., SELECT, WHERE) but struggles with more complex queries involving JOINs. Transaction Processing: Aware that transaction processing involves multiple steps but lacks understanding of ACID	Compression and Encryption: Can explain the difference between lossless and lossy compression and understands how encryption works to secure data but may need support in more complex examples (e.g., asymmetric encryption). Database Concepts: Can explain the basic structure of a database and how data is stored in tables, fields, and records, with an understanding of keys (primary and foreign keys). Relational Databases and Normalisation: Understands the importance of normalisation in relational databases and can apply first and second normal forms to eliminate data redundancy. SQL: Proficient in writing SQL queries to retrieve, filter, update, and delete data, and can use simple JOINs to combine data from multiple tables.	Compression and Encryption: Can critically analyse and apply various compression and encryption algorithms (e.g., Huffman coding, RSA encryption) and explain their strengths and weaknesses in different contexts (e.g., web security, file storage). Database Concepts: Can design and optimise complex database structures, explain the role of indexes, and manage relationships between tables using advanced techniques (e.g., recursive relationships, many-to-many relationships). Relational Databases and Normalisation: Expert in normalising databases to third normal form (3NF). SQL: Capable of writing advanced SQL queries involving subqueries, JOINs and complex filtering to retrieve and manipulate data efficiently. Transaction Processing: Deep understanding of transaction processing, including the application of the ACID properties in high-volume, multi-user environments, and can design systems that ensure data integrity during concurrent transactions.

	properties (atomicity, consistency, isolation, durability) and their importance in ensuring data integrity.	Transaction Processing: Understands the basic concept of transaction processing and can explain how ACID properties ensure data consistency and reliability in multi-step transactions.	
Summer 1 Network Security	Structure of the Internet: Can identify basic components of the internet (e.g., routers, servers) but needs support in understanding the overall structure and how data is transferred. Internet Communication: Understands that data is exchanged over the internet but struggles to explain protocols (e.g., TCP/IP, HTTP) or how communication happens between devices. Network Security and Threats: Aware of basic security threats (e.g., viruses, phishing) but struggles to explain how these threats affect network security or how to prevent them. HTML and CSS: Can write basic HTML to create simple web pages but struggles with using CSS to style and format content effectively. JavaScript: Understands basic JavaScript concepts (e.g., variables, functions) but has difficulty applying them to create interactive elements or dynamic web pages. Search Engine Indexing: Aware that search engines index web pages but struggles to explain the process or the role of SEO in improving page rankings.	Structure of the Internet: Can explain the general structure of the internet, including key components like routers, servers, and DNS, and how they work together to route data. Internet Communication: Understands key communication protocols (e.g., TCP/IP, HTTP, HTTPS) and can explain how they ensure reliable data transfer between devices. Network Security and Threats: Can identify common network security threats (e.g., malware, DDoS attacks) and explain basic preventive measures (e.g., firewalls, encryption). HTML and CSS: Proficient in writing HTML to structure content and using CSS to style it effectively, including creating layouts and applying responsive design techniques. JavaScript: Comfortable using JavaScript to create interactive web pages, handle events, and manipulate the DOM (Document Object Model) for dynamic content updates. Search Engine Indexing: Understands how search engines index content and how SEO techniques (e.g., keywords, meta tags) improve visibility and rankings.	Structure of the Internet: Has an in-depth understanding of the internet's architecture, including the roles of ISPs, backbone networks, and how data flows across the network using protocols like BGP (Border Gateway Protocol). Internet Communication: Can critically analyse the flow of data over the internet, explaining how different protocols (e.g., TCP, IP, HTTP, FTP) work together to ensure secure and efficient communication, including error handling and data transmission. Network Security and Threats: Deep understanding of network security practices, including encryption, firewalls, VPNs, and intrusion detection systems (IDS), and can explain how these tools mitigate advanced threats like man-in-the-middle attacks and ransomware. HTML and CSS: Expert in using HTML5 and CSS3 to create complex, responsive websites with advanced features like animations, transitions, and accessibility best practices. JavaScript: Proficient in advanced JavaScript techniques, including asynchronous programming (e.g., promises, async/await), using frameworks (e.g., React, Vue), and understanding JavaScript's role in full-stack development. Search Engine Indexing: Can explain the technical aspects of search engine indexing, including crawling,

	Client-Server and Peer-to-Peer: Can explain the basic difference between client-server and peer-to-peer networks but lacks an understanding of how each model works in practice or their pros and cons.	Client-Server and Peer-to-Peer: Can explain how client-server and peer-to-peer networks work in practice, including the roles of clients, servers, and peers in data exchange and resource sharing.	ranking algorithms, and the importance of site structure and user experience for SEO. Client-Server and Peer-to-Peer: Fully understands the advantages and limitations of client-server and peer-to-peer models, including their implementation in real-world systems (e.g., HTTP servers, file-sharing networks, blockchain) and their impact on scalability, security, and performance.
Summer 2 Data Types	Data Types: Can identify basic data types (e.g., integer, string) but needs support in explaining their uses and how they are represented in memory. Binary and Hexadecimal: Understands the concept of binary and hexadecimal but struggles with converting between number systems or explaining their use in computing. ASCII & Unicode: Aware that ASCII and Unicode are used for character encoding but has difficulty explaining how they work or converting between characters and their binary representations. Binary Arithmetic: Understands basic binary numbers but needs guidance in performing arithmetic operations (e.g., addition, subtraction) on binary numbers. Floating Point Arithmetic: Aware that floating-point numbers are used to represent real numbers but struggles to explain their structure or perform arithmetic with them.	Data Types: Can explain different data types (e.g., integer, float, string) and how they are used in programming, including their memory representation and storage requirements. Binary and Hexadecimal: Comfortable converting between binary and hexadecimal and understands their importance in computer systems for tasks like memory addressing and error detection. ASCII & Unicode: Can explain how ASCII and Unicode represent characters and is able to convert between characters and their corresponding binary values in both encodings. Binary Arithmetic: Can perform basic binary arithmetic (e.g., addition, subtraction) and understand how carryover works in binary addition. Floating Point Arithmetic: Understands the structure of floating-point numbers (e.g., mantissa, exponent) and can perform basic floating-point arithmetic, including rounding and handling precision issues.	Data Types: Deep understanding of various data types, including advanced types (e.g., arrays, structs, pointers) and how they are represented in memory, including how different types impact memory allocation and program performance. Binary and Hexadecimal: Expert in converting between binary, hexadecimal, and other bases (e.g., octal), and can explain the practical applications of these systems in low-level programming, networking, and error-checking algorithms. ASCII & Unicode: Fully understands the differences between ASCII and Unicode. Can convert between different encodings and explain their use in modern systems. Binary Arithmetic: Proficient in performing complex binary arithmetic, including operations with signed binary numbers, and understands how to handle overflow, underflow, and signed number representations (e.g., two's complement). Floating Point Arithmetic: Expert in floating-point representation and can explain the limitations of floating-point precision and how to handle rounding

	Bitwise Manipulation and Masks: Has a basic understanding of bitwise operations but struggles with applying them to solve practical problems or using masks for manipulating bits.	Bitwise Manipulation and Masks: Can perform basic bitwise operations (AND, OR, XOR, NOT) and understand how masks can be used to manipulate specific bits within a value.	errors and floating-point imprecision in real-world applications. Bitwise Manipulation and Masks: Can apply bitwise manipulation and bitwise masks to solve complex problems, such as setting, clearing, or toggling specific bits, optimising data storage, and working with low-level system programming tasks (e.g., hardware control, encryption algorithms).