Curriculum Map

Subject: Computer Science **Year Group: 10**

The Curriculum Map for Computer Science follows two parallel strands, split between Computational Thinking (CT) - the programming aspects covered in Topics 1 & 6 and the Principles of Computer Science (P) - the theory aspects covered by Topics 1 – 5

	Autumn 1	Autumn 2	Spring 1	Spring 2	Summer 1	Summer 2
Content	**Topic1 and 6:** Computation Thinking and Problem Solving **Key Areas:** 1.1 Programming: tools and strategies 1.2 Algorithms and programs 1.3 Data types and variables **Keywords:** Program, programming language, python, arithmetic operators, BIDMAS, algorithms, variables, data types, IDE, sequence, identifier	**Topic 1 and 6:** Computation Thinking and Problem Solving **Key Areas:** 1.4 Selection and relational operators 1.5 Repetition 1.6 One-dimensional data structure **Keywords:** Input, output, data structure, repetition, function, relational operator, selection operator	**Topic 2:** Data **Key Areas:** 2.1 Binary 2.2 Data Representation (Part 1) **Keywords:** Binary, nibble, bit byte, kilobyte, megabyte, signed and unsigned integers, two's complement, overflow error, arithmetic and logical binary shift	**Topic 2:** Data **Key Areas:** 2.2 Data Representation (Part 2) 2.3 Data storage and compression **Keywords:** Hexadecimal, ASCII, analogue and digital data, amplitude, sample rate, bit depth, sample interval, compression, lossless and lossy compression	**Topic 3:** Computers **Key Areas:** 3.1 Hardware 3.2 Software (Part 1) **Keywords:** Hardware, software, Von-Neumann architecture, CPU, RAM, ROM, cache, virtual memory, magnetic, optical, solid-state, operating system, paging	**Topic 3:** Computers **Key areas:** 3.2 Software (Part 2) 3.3 Programming Languages **Keywords:** GUI, device driver, utility software, low-level and high-level languages, instruction set, translators, interpreters, compilers
Skills	➢ Analytical skills ➢ Critical-thinking skills ➢ Problem-solving skills ➢ Programming skills					
Key questions	➢ Define the term 'program'	➢ Explain input and output	➢ Define what is meant by the terms 'nibble' and 'byte'	➢ Convert binary to the hexadecimal equivalent	➢ Define what is meant by the 'stored program concept'	➢ Identify different types of utility software

	➢ Identify types of programs used every day ➢ Identify types of programming languages ➢ Explain the integrated development environment ➢ Use arithmetic operators and BIDMAS ➢ Explain the importance of code layout ➢ Explain errors in programs ➢ Evaluate the use of variables in algorithms and programs	➢ Define the term 'runtime error' ➢ Explain primitive data types (integer, real, char, string) ➢ List flowchart symbols ➢ Represent an algorithm in a flowchart ➢ Explain how flowcharts are translated into code ➢ Define the terms 'array' and 'list' ➢ Explain how to access items in a list using indexing ➢ Create, append, delete items from a list ➢ Explain how the range function generates a sequence of numbers ➢ Use iteration 'for' to process every item in a one-dimensional data structure	➢ Convert between denary and binary numbers ➢ Differentiate between signed and unsigned integers ➢ Describe how positive and negative numbers are represented in two's complement ➢ Define what is meant by the terms 'binary' and 'bit' ➢ Explain why binary is used to represent data and program instructions in a computer ➢ Describe the effects of an overflow error ➢ Explain why arithmetic right shift differs from a logical right shift	➢ Explain why hexadecimal is used ➢ Describe how characters are encoded in ASCII ➢ Derive the code for an ASCII character from that of another ➢ Describe the limitations of ASCII ➢ Differentiate between analogue and digital data ➢ Explain the difference between image size and image resolution ➢ Define what is meant by the terms 'amplitude', 'sample rate', 'bit depth' and 'sample interval' ➢ Describe the process of converting analogue sound into binary data. ➢ Identify factors that affect the accuracy of the digital representation.	➢ Describe the hardware components used in the von Neumann architecture and explain their role in the fetch-decode-execute cycle ➢ Explain how the speed of the clock impacts on performance ➢ Explain how pipelining improves the performance of the CPU ➢ Explain the need for secondary storage ➢ Describe how data are stored on magnetic, optical and solid-state media ➢ Compare the capacity, speed and portability of magnetic, optical and solid-state storage devices ➢ Describe the role of the operating system in a computer system ➢ Identify tasks carried out by an OS	➢ Describe how an OS allocates each active process a share of CPU time ➢ Explain the role of a device driver ➢ Explain the features of a GUI user interface ➢ Define what is meant by the terms 'low-level language' and 'high-level language' ➢ Explain why each processor has its own unique instruction set ➢ Describe how writing a program in a low-level language differs from writing one in a high-level language ➢ Compare features of low-level and high-level languages and identify tasks for which each is best suited ➢ Explain the need for program translators ➢ Define what is meant by the terms 'compiler' and 'interpreter'

		➢ Define the term procedure and parameter ➢ Define the terms function, procedure, parameters, return value ➢ Decompose problems ➢ Write functions and procedures with/without parameters		➢ Give reasons for wanting to reduce file sizes (storage, streaming) ➢ Describe how compression affects file sizes ➢ Identify potential drawback of compressing files ➢ Explain the difference between lossless and lossy compression ➢ Describe the advantages/disadvantages of each type of compression	➢ Describe how the OS organises files and allocates space on a hard drive ➢ Describe how file permissions are used to control access to files ➢ Explain the levels of file access (read, write, delete, none) for a user ➢ Describe how an OS uses scheduling to give each active process a share of CPU time ➢ Describe the features of the round-robin scheduling algorithm ➢ Describe how the OS uses a paging algorithm to swap programs in and out of main memory	➢ Compare the way in which interpreters and compilers translate high-level code into machine code ➢ Describe the advantages/disadvantages of each approach ➢ Select and justify which method of translation to use for a given purpose
Assessment	**Formative Assessment:** Target questioning, quizzes, individual and group tasks **Summative Assessment:** Unit test End-of-term test					
Literacy/	Demonstrate and apply knowledge and understanding of the key concepts and principles of computer science					

Numeracy/ SMSC/ Character	Analyse problems in computational terms: - to make reasoned judgements - to design, program, evaluate and refine solutions