Curriculum Plans: Year 12 (Computer Science)

	Topic	Knowledge: By the end of the unit students will know:	Skills: What skills will students have developed by the end of this unit?	Key terms: What new key terms and vocabulary will be learnt in this unit?	Summative Assessment: How will pupils be assessed in this unit?
Term 1	SLR12 Web Technologies	• Understanding of the basic structure of a webpage using HTML, including common tags and attributes. • The role of CSS in web design and how it is used to style HTML elements. • Differences between client-side and server-side scripting languages. • Understanding the purpose of web servers and the relationship between clients and servers. • How web hosting and domain names work, and the importance of SEO in web development. • Basic understanding of HTTP and HTTPS protocols, and the role of encryption in web security. • The use of cookies and sessions to store user data across web sessions. • The function and importance of APIs (Application Programming	• Writing and editing HTML code to create functional web pages. • Using CSS to apply styling rules to HTML elements, both inline, embedded, and external stylesheets. • Implementing simple JavaScript code for client-side functionality. • Using JavaScript to interact with HTML forms and perform basic validations. • Applying SEO principles to improve website visibility in search engines. • Differentiating between HTTP and HTTPS and understanding how SSL/TLS works to secure communications. • Writing code that makes use of cookies and sessions to manage user state on websites.	• HTML • CSS • JavaScript • Client Side Processing • Server Side Processing • SEO • PageRank • Optimisation. • Search Engine Indexing • Web Server • HTTP/HTTPS • API (Application Programming Interface) • JSON • XML	**Written Assessment –** Mid topic MCQ test **Written Assessment –** End of topic test

		Interfaces) in web development. • - Key differences between JSON and XML for data interchange in web technologies.	• Implementing basic APIs to retrieve and display data in a web application. • - Writing and reading data in both JSON and XML formats for web applications.		
Term 1	SLR1 Structure and function of the processor	• Understanding the key components of the CPU, including ALU, CU, and registers. • How the fetch-decode-execute cycle works and the role of each register during this cycle. • Differences between Von Neumann and Harvard architectures and their practical uses. • The significance of clock speed, cache size, and number of cores in determining CPU performance. • The role of buses (address, data, and control) in communication within the CPU. • The use of pipelining to improve processor efficiency. • Basic assembly language and its relationship to machine code. • - How instruction sets define the operations a processor can perform.	• Describing the function of the CPU's components and how they contribute to processing. • Explaining and illustrating the fetch-decode-execute cycle, identifying the role of each register. • Comparing and contrasting different processor architectures in terms of structure and function. • Evaluating the performance of CPUs based on clock speed, cache, and number of cores. • Explaining how data is transferred between the CPU and memory using the different types of buses.	• CPU (Central Processing Unit) • ALU (Arithmetic Logic Unit) • Control Unit (CU) • Register • Program Counter (PC) • Memory Address Register (MAR) • Memory Data Register (MDR) • Von Neumann Architecture • Harvard Architecture	

			• Describing how pipelining speeds up the processing of multiple instructions in modern processors. • Writing simple assembly language programs and understanding how the CPU executes them. • Understanding the role of instruction sets in the execution of tasks by the CPU. • Implementing tasks in Little Man Computer (LMC) to simulate CPU processes.		
Term 1	SLR2 Types of processor	• Understanding the differences between CISC and RISC processors, including their advantages and disadvantages. • The role and applications of GPUs, particularly in tasks requiring parallel processing. • The importance of multicore and parallel processing in modern CPUs for performance improvements. • The concept of parallel processing and how it allows simultaneous execution of tasks.	• Comparing and analysing the use cases of CISC and RISC processors. • Explaining the role of GPUs in rendering and their increasing use in non-graphical tasks such as AI. • Evaluating the performance benefits and limitations of multicore processors. • Understanding and explaining how parallel processing boosts	• CISC (Complex Instruction Set Computer) • RISC (Reduced Instruction Set Computer) • GPU (Graphics Processing Unit) • Parallel Processing • Multi-core Processor • Co-processor • Quantum Processor	**Written Assessment –** End of topic test with MCQ on MS Forms

Curriculum Plans: Year 12 (Computer Science)

		• The differences between single-core and multi-core processors and their impact on computing performance. • The structure and functions of co-processors and their role in assisting the main CPU in specialized tasks. • Emerging technologies such as quantum processors and their potential future applications. • The concept of vector processors and their use in handling mathematical computations efficiently.	efficiency in systems such as gaming consoles. • Demonstrating the impact of increasing core numbers on processing speed and performance. • Identifying scenarios where co-processors are utilized to improve system efficiency. • Researching and explaining the implications of quantum processors on future computing systems. • Describing how vector processors operate and how they are used in certain computational fields.	• Vector Processor	
Term 1	SLR9 Compression, encryption and hashing	• The purpose of compression and its importance for reducing file sizes in images, audio, and videos. • Understanding and applying Run-Length Encoding (RLE) and Huffman Coding for compression.	• Differentiating between lossy and lossless compression techniques and their appropriate use cases. • Encoding and decoding data using Run-Length Encoding and Huffman Coding methods.	• Compression • Lossy Compression • Lossless Compression • Run-Length Encoding (RLE) • Huffman Coding	**Written Assessment –** End of topic test with MCQ on MS Forms

		• The differences between symmetric and asymmetric encryption. • The role of public and private keys in securing communications. • How hashing algorithms work, and their use in data integrity and password storage. • Identifying weaknesses in hashing algorithms, such as collisions, and understanding their impact. • Use cases of encryption and hashing in real-world scenarios, including file integrity and security.	• Applying symmetric encryption methods such as Caesar Cipher, and understanding key management. • Using public and private key pairs for secure communication in asymmetric encryption scenarios. • Implementing and explaining how hashing algorithms, such as MD5 and SHA-256, are used for securing data. • Detecting hash collisions and understanding the importance of strong hash functions in security. • - Applying encryption and hashing to secure sensitive information and maintain data integrity.	• Symmetric Encryption • Asymmetric Encryption, • Caesar Cipher • Public Key • Private Key • Encryption • Digital Signature • Hashing • Hash Function, • MD5 • SHA-256 • Checksum • Collision • Hash Table • Hash Function • Encryption	
Term 2	SLR3 Input, output and storage	• Understanding the different types of input, output, and storage devices and their roles in computing. • Distinguishing between primary, secondary, and tertiary storage, and when each is used.	• Identifying various input, output, and storage devices and explaining their functions. • Choosing appropriate storage devices based on a given scenario (e.g.,	• Input Device • Output Device • Storage Device • Primary Storage • Secondary Storage • Tertiary Storage	

		• The differences between volatile and non-volatile storage and their importance. • How storage devices like SSDs, HDDs, and optical drives work and their advantages/disadvantages. • The function and characteristics of RAM and ROM. • The concept of virtual memory and its role in extending RAM capacity. • Cloud storage and its benefits and drawbacks for both personal and organizational use. • How buffers work in managing the flow of data between devices. • - Differences between traditional and modern storage technologies such as magnetic disks and SSDs.	SSD for fast access, HDD for capacity). • Differentiating between RAM and ROM and explaining their uses in computer systems. • Evaluating the benefits and limitations of magnetic, optical, and solid-state storage. • Explaining the role of RAM in multitasking and ROM in bootstrapping a computer system. • Explaining the use of virtual memory when RAM is insufficient for running applications. • Comparing cloud storage options and discussing the trade-offs between local and cloud storage solutions. • Describing how input/output controllers and buffers manage data flow between hardware components. • - Demonstrating knowledge of how storage media impacts performance and reliability.	• Volatile Storage • Non-Volatile Storage • Solid-State Drive (SSD) • Hard Disk Drive (HDD) • Optical Disk • Flash Memory	

Curriculum Plans: Year 12 (Computer Science)

Term 2	SLR4 Operating systems - Systems software	• Understanding the function and purpose of operating systems, including memory and process management. • The role of the kernel and its responsibilities in process and resource management. • Differences between CLI and GUI, and their advantages and disadvantages in different environments. • The purpose and function of device drivers in managing hardware components. • How operating systems manage multitasking, scheduling, and interrupts. • The concept of virtual machines and their uses in running different operating systems or software. • Basic memory management techniques such as paging and segmentation. • Types of operating systems: distributed, embedded, real-time, multi-user, and multitasking. • The importance of system utilities for system maintenance and optimization.	• Describing the core functions of an operating system, such as file management and user interfaces. • Explaining how the kernel manages system resources like CPU time and memory allocation. • Using both CLI and GUI interfaces to interact with an operating system effectively. • Installing and managing device drivers to ensure proper hardware operation. • Describing multitasking, multithreading, and scheduling algorithms and their impact on system performance. • Setting up and using virtual machines for testing software in isolated environments. • Explaining how memory is allocated and managed using paging or segmentation. • Comparing different types of operating	• Operating System (OS) • Kernel • Command Line Interface (CLI) • Graphical User Interface (GUI) • Device Driver • Virtual Machine • Memory Management • Real-Time Operating System (RTOS), • Multitasking • Multithreading • Utility Software • BIOS • Boot Loader	**Written Assessment –** End of topic test with MCQ on MS Forms

		• - The function of BIOS and boot loaders in initialising system hardware and loading the OS.	systems and identifying their use cases. • Utilising system utilities like disk defragmentation, backups, and antivirus tools to maintain system performance. • - Explaining the role of BIOS and boot loaders during system startup.		
Term 2	SLR5 Application generation	• Understanding the difference between application software and system software. • Types of application software: word processors, spreadsheets, databases, multimedia software, and more. • The difference between off-the-shelf, custom-written, open-source, and proprietary software. • The role of translators (compilers, interpreters, assemblers) in converting source code to machine code. • The stages of compilation: lexical analysis, syntax analysis, code generation, and optimization. • The function of libraries and how they can be linked to programs during development.	• Identifying appropriate application software for specific tasks (e.g., word processing, databases). • Using various application software tools for document creation, data manipulation, and project management. • Evaluating the benefits and drawbacks of off-the-shelf versus custom-written software solutions. • Explaining how compilers, interpreters, and assemblers function and their uses in software development. • Understanding the compilation process and	• Application Software • Word Processing, Spreadsheet, Database Management System (DBMS) • Custom Software, Off-the-Shelf Software • Compiler, Interpreter, Assembler • Executable File, Linker, Loader • Library, Linker, Loader • Utility Program • - Open Source, Closed Source	

		• Understanding utility programs (e.g., antivirus, compression, backup) and their role in system maintenance. • The importance of open-source versus closed-source (proprietary) software in development and distribution.	how code is converted into executable files. • Using pre-built libraries in software development to improve efficiency and reduce development time. • Explaining the purpose of utility software and how it assists in maintaining system performance. • Evaluating the pros and cons of open-source versus closed-source software for different use cases.		
Term 3	SLR7 Types of programming language	• Differences between high-level and low-level programming languages, including their use cases. • Understanding machine code and assembly language and their relationship to hardware. • The advantages and disadvantages of compiled and interpreted languages. • Features of procedural programming, including sequence, selection, and iteration.	• Identifying when to use high-level vs. low-level languages based on task requirements. • Writing basic programs in assembly language and translating them to machine code. • Explaining the differences between compilers and interpreters and when to use each. • Writing programs using procedural	• High-Level Language, Low-Level Language • Machine Code, Assembly Language • Compiled Language, Interpreted Language • Procedural Programming • Object-Oriented	**Written Assessment -** Written Exam Assessed in the end of year exam

Curriculum Plans: Year 12 (Computer Science)

		• Basics of object-oriented programming (OOP), including concepts like classes, objects, inheritance, and polymorphism. • Functional programming concepts such as first-class functions, higher-order functions, and immutability. • Declarative programming and its applications, such as SQL and Prolog. • The role of concurrency and parallelism in modern computing. • - Different addressing modes: immediate, direct, indirect, and indexed addressing.	programming concepts such as loops and conditionals. • Implementing object-oriented programs using classes, objects, and inheritance. • Writing basic functional programs using functions as first-class citizens. • Solving problems using declarative programming by specifying what to solve, not how to solve it. • Applying concepts of concurrency and parallelism to improve program efficiency in multi-core systems. • - Using addressing modes correctly when writing assembly language programs.	Programming (OOP) • Functional Programming • Declarative Programming • Concurrency • Parallelism • Immediate Addressing • Direct Addressing • Indirect Addressing • Indexed Addressing	
Term 3	Introduction to the NEA	• Understanding the full system development life cycle (SDLC), including analysis, design, implementation, testing, and evaluation. • The importance of thoroughly defining and understanding a real-world problem that can be	• Conducting a full project lifecycle by researching, designing, coding, and evaluating a software solution. • Applying computational thinking techniques to decompose complex	• System Development Life Cycle (SDLC) • Analysis • Design • Implementation • Testing	

		solved using computational thinking. • Knowledge of various algorithms and data structures and their application to solving specific problems. • How to design a user-friendly interface (UI) that allows for effective interaction with the software. • Understanding of software testing techniques, including validation and verification to ensure the program meets its specification. • The importance of evaluating a completed project in terms of its effectiveness, efficiency, and user satisfaction. • Documentation skills for maintaining accurate project records, including the use of comments, flowcharts, and pseudocode. • - Legal, moral, and ethical considerations when developing software, including data privacy and security.	problems into manageable tasks. • Implementing appropriate algorithms and data structures in the software project to efficiently solve problems. • Designing and developing an intuitive user interface (UI) that meets the needs of the target audience. • Writing and conducting test plans to validate and verify the functionality of the software solution. • Producing detailed evaluations of the software solution, including feedback from end-users and potential improvements. • Documenting each phase of the project, including code comments, user guides, and technical documentation. • - Applying ethical guidelines and ensuring data protection	• Evaluation • Computational Thinking • Problem Decomposition • Abstraction • Algorithms • Data Structures • Efficiency • User Interface (UI) • User Experience (UX) • Testing • Validation, • Verification • Test Plan • Evaluation • User Feedback • Improvement Suggestions • Documentation • Flowchart • Pseudocode • Ethics • Data Protection • Privacy	

			throughout the development process.		