Y10

Theoretical Concepts

Ethics & Legislation

T6

Ethics and legislation in Computer Science explore the impact of technology on society and the planet. Topics include ethical concerns like privacy, digital divide, and responsible data use; environmental issues such as e-waste, energy consumption, and sustainability in computing; and legal frameworks like the Data Protection Act, Computer Misuse Act, and copyright law. These themes promote understanding of responsible and lawful technology use in a modern world.

System Security & Software

T5

System software ensures hardware and software function smoothly, incorporating operating systems and utilities like file management and disk defragmentation. Operating systems handle tasks such as process, memory, device, and user management. System security focuses on protecting data and systems from threats such as malware, hackers, and social engineering. Countermeasures include strong passwords, firewalls, anti-virus software, encryption, regular updates, user access levels, and frequent, robust backups.

Networks and Protocols

T4

Networks can be LANs or WANs, with performance shaped by factors like bandwidth and connected devices. Client-server or peer-to-peer setups require hardware—such as routers, switches, and NICs—to function. The Internet spans global networks, aided by DNS, web servers, and the Cloud. Star and Mesh topologies have pros and cons. Wired or wireless connections rely on protocols, addresses, standards, and encryption, often arranged in layered structures.

Representing Data

T3

In computing, data is always stored in binary and measured in units from bits through petabytes. Conversions between decimal, binary, and hexadecimal allow arithmetic operations and representation of characters, images, and sound. File sizes depend on color depth, resolution, sample rate, and bit depth. Storage capacity calculations require understanding each unit's size. Compression, whether lossy or lossless, reduces file size while balancing quality and usability.

Recap Memory and Storage

T2

Computers need primary storage—RAM for working data, ROM for startup routines—and deploy virtual memory by shifting overflow data to secondary storage when RAM is full. Cache accelerates frequent tasks. Secondary storage options—optical, magnetic, or solid-state—are chosen for capacity, speed, portability, durability, reliability, and cost. Proper selection ensures efficient data handling. Each device's strengths suit particular scenarios, securing essential data while balancing overall performance and expense.

Recap Systems Architecture

T1

The CPU's primary role is to fetch, decode, and execute instructions using components like the ALU, Control Unit, registers (MAR, MDR, Program Counter, Accumulator), and cache, adhering to Von Neumann architecture. Data or addresses may be stored differently within these registers. Performance varies with clock speed, cache size, and core count. Embedded systems, possessing specialised functions and limited resources, are widely used in everyday devices.

Computer Science

Y10

Practical Skills

Controlled Assessment Part 2

When concluding a development cycle, the team finalises the system, verifying all components function properly. Rigorous testing is performed to identify flaws, ensuring the product meets user expectations. Afterward, evaluators measure outcomes against the original evaluation criteria, checking for fulfilment of requirements and user satisfaction. This confirmation phase helps refine the final solution, guaranteeing its reliability, usability, and adherence to specified goals with necessary adjustments.

T6

Controlled Assessment Part 1

T5

System development begins by analysing the problem, clarifying requirements, and defining objectives. A carefully designed solution architecture guides the creation of prototypes and initial builds. During implementation, teams frequently test features, isolate errors, and refine components. As the project unfolds, iterative improvements ensure alignment with user needs. Ultimately, this structured approach supports reliable, effective software solutions that comprehensively meet its intended goals.

Plan Design and Develop

System development is the process of turning an idea into a functioning solution, involving steps such as requirements analysis, design, implementation, testing, and evaluation. Exploring how each phase organises tasks, manages time, and ensures functionality. Helps to learn to refine solutions based on feedback, addressing errors, and ensuring systems meet user needs efficiently and effectively.

T4

Further Python Code

T3

More advanced Python covers deeper programming concepts beyond basic syntax, focusing on structured approaches such as defining and calling functions to manage complex tasks. It explores handling data storage using SQLite databases for persistent information, as well as reading and writing CSV files to manipulate tabular data. This skill set emphasises code organisation, reliability, and extensibility. This fosters problem-solving skills with real-world relevance.

Simple Python Code

Python fundamentals include essential coding concepts, such as using variables and their data types, applying operators, and leveraging control structures like loops and conditionals. learning to handle user input, process data, and generate output. Basic concepts of functions, file handling, and code readability also underpin effective problem-solving, enabling an accessible, practical understanding of core programming principles, and debugging methods.

T2

Introduction to Assembly

T1

Assembly language and the Little Man Computer (LMC) are crucial because they demonstrate how low-level instructions directly control hardware through memory manipulation. They illustrate core CPU operations like fetching, decoding, and executing commands, teaching the fundamental concepts behind high-level language translation. By exploring assembly code and LMC, one gains understanding of computer architecture, registers, and the inner workings of modern systems.

Computer Science

Y11

T6

Languages and IDEs

T5

High-level languages are easier to write and more user-friendly, while low-level languages offer direct control over hardware. Translators convert code between these levels, with compilers producing standalone executables, and interpreters translating on-the-fly. Each method has benefits and drawbacks. IDEs further assist development, providing editors, error diagnostics, run-time environments, and integrated translators. These tools streamline coding, debugging, and testing, making programming more efficient and effective overall.

Boolean Logic

T4

This topic covers basic logic operators AND, OR, and NOT, their truth tables, and how these gates are represented in logic diagrams. Students need to recognise each gate symbol, combine operators to create or edit complex diagrams, and interpret resulting outputs. Solving problems involves applying logical operators, understanding how inputs determine outputs, and producing correct truth tables. Multiple gates can be linked for complex scenarios.

Robust Programming

T3

Defensive design uses input validation, anticipating misuse, and authentication to confirm identity, with maintainability supported by subprograms, naming conventions, indentation, and commenting. Testing ensures reliability, involving iterative and final checks that detect syntax and logic errors. Programmers employ normal, boundary, invalid, and erroneous data to refine algorithms and validate modules. Test plans confirm correctness and help address potential flaws before release and thoroughly document outcomes.

Programming Fundamentals

T2

Effective software requires defensive design—validating inputs, anticipating misuse, and authenticating users—supported by maintainable code with consistent naming, subprograms, indentation, and comments. Rigorous testing ensures reliability, using normal, boundary, invalid, and erroneous data. A range of data types, string manipulation, file handling, arrays, records, SQL, and subprograms enable structured solutions, while random number generation enhances functionality, supported by thorough test plans, extensive documentation, and robust debugging.

Algorithms

T1

Computational thinking principles—abstraction, decomposition, and algorithmic thinking—help define and refine problems, while input-process-output structures guide solutions. Designers use structure diagrams, pseudocode, and flowcharts to build, correct, and refine algorithms. Syntax and logic errors must be identified and traced. Standard searches (binary, linear) and sorts (bubble, merge, insertion) have defined steps and requirements, and can be recognised and applied to data sets, with prerequisites considered thoroughly.

Computer Science