Armfield Academy – Department of Computer Science

Year 10 Curriculum Overview

Half Term 1			
Date	2.1.1: Computational Thinking 2.1.2: Problem Solving and Algorithms 2.1.3: Algorithms (searching & sorting)		
	Lesson 1	Lesson 2	Lesson 3
Week 1	What is computational thinking and why is it crucial in solving problems with computers?	How do abstraction and decomposition help us simplify complex problems?	Can you identify patterns and write a sequence of steps (algorithms) to solve everyday problems? Application: Use past paper questions on abstraction, decomposition, and algorithm design. Students work independently for 30 minutes, then peer-mark in pairs.
Week 2	What is a flowchart and how can we represent an algorithm visually?	What is pseudocode and how do we use it to write clear, logical algorithms?	How can we trace an algorithm using a trace table and spot logical errors? Application: Complete a mock task with flowcharts and pseudocode to design a login system. Include trace table evaluation and logic correction.
Week 3	What are linear and binary search algorithms and how do they differ in performance?	How can we write pseudocode for a binary search and trace its steps?	How can we apply binary search logic to real-world data sets (e.g., name lists or leader board scores)? Application: Past paper problems on linear and binary search. Students apply searching algorithms to sample datasets, then justify which one is most efficient and why.
Week 4	What is a bubble sort and how does it work step-by-step?	What is a merge sort and how does divide-and-conquer make it efficient?	What are the differences between bubble and merge sorts in terms of speed and memory? Application: Students complete sort-based tasks from previous OCR assessments, including comparing and evaluating efficiency in different scenarios.
Week 5	Can you write your own bubble and merge sort in pseudocode and correct an error in one?	How can we use dry runs and trace tables to test the accuracy of a sorting algorithm?	Can you design a short interactive task to demonstrate sorting visually (e.g., paper-based cards or online tool)? Application: 30-minute timed mini-assessment: write, trace, and compare two sorting algorithms with written justifications.
Week 6	What makes an algorithm efficient, and how can we compare algorithms for time and space complexity (qualitatively)?	How can we modify existing algorithms to solve slightly different problems (e.g., searching by letter, not word)?	Can you spot inefficiencies in given pseudocode and suggest optimisations? Application: Students tackle a mixed algorithm challenge involving both searching and sorting with a focus on real exam problem-solving strategies.
Week 7	How can exam questions be tackled using structured thinking and planning first?	What common mistakes are made in algorithm-based exam answers, and how can you avoid them?	Can you independently solve a multi-step exam-style question involving computational thinking, searching, and sorting? Final Application: Mock exam paper (30 mins) covering all three subtopics (2.1.1, 2.1.2, 2.1.3). Students apply all they've learned and review in groups.
Half Term 2			
Date	2.2.1: Programming Fundamentals 2.2.2: Data Types & Casting and 2.2.3: Additional Programming Techniques Using Python		
	Lesson 1	Lesson 2	Lesson 3
Week 1	What are variables, constants, inputs, outputs, and assignments, and how do we use them in a program?	How do sequence, selection (IF), and iteration (loops) control how a program runs?	Can you write a simple interactive program that uses input/output, IF statements, and loops to simulate a quiz or number guesser? Application: Students complete 2 exam-style coding questions using sequence, selection, and count-controlled loops.

Week 2	How do comparison and Boolean operators (==, !=, <, >, AND, OR, NOT) affect the logic of a program?	What are the common arithmetic operators (+, –, , /, MOD, DIV, ^) and when do we use them in code?	Can you write a calculator or grading program that uses arithmetic and logical operators with IF/ELSE structures? Application: Students solve logic-based problems using nested IF statements and arithmetic in pseudocode or Python.
Week 3	What are the different data types (integer, real, Boolean, character, string), and when should each be used?	How can we use casting (e.g., int(), str(), float()) to convert between data types, and why is this useful?	Can you create a form-style program that takes user input and validates or casts it to the right type? Application: Students complete input/output tasks, choosing the correct data type and applying casting with validation.
Week 4	What is string manipulation and how can we slice, concatenate, and format strings in a program?	What is an array and how can we store and retrieve data using one-dimensional arrays?	Can you build a program that stores a list of students' names and allows users to search, add, or update them? Application: Tasks involve string slicing, appending to arrays, and looping through array contents.
Week 5	How do we open, read, write, and close files in a program, and why is file handling important?	What are records and two-dimensional arrays, and how can we use them to store structured data like a database table?	Can you create a program that reads from a file into a 2D array and allows basic lookup or edits? Application: Students complete file-read and 2D array tasks (e.g., CSV-based table of students and marks).
Week 6	How can we use SQL commands (SELECT, FROM, WHERE) to search data stored in a table or 2D array?	What are subroutines (functions and procedures), and how do they help us write organised code?	Can you write a program that includes a function to calculate and return a result (e.g., average grade), and also uses an SQL-style search feature? Application: Exam questions include writing subroutines and SELECT queries using pseudocode.
Week 7	How do we generate and use random numbers in a program, and what practical uses do they have?	How can we write robust programs that handle errors, incorrect input, and edge cases?	Can you combine variables, arrays, loops, subroutines, and randomness to create a mini-project like a number game or user login system? Application: Final mock programming challenge – students apply all concepts from 2.2.1–2.2.3 in one integrated program.
Half Term 3			
Date	2.3.1: Defensive Design and 2.3.2: Testing Using Python		
	Lesson 1	Lesson 2	Lesson 3
Week 1	How can a program anticipate incorrect use and prevent misuse through design?	What is authentication and how can it be implemented in a login system?	Can you build a simple login system using username/password authentication and prevent basic misuse? Application: Students complete an exam-style task requiring them to validate a login, reject unauthorised users, and comment on misuse.
Week 2	How does input validation protect a program from crashing or behaving unpredictably?	What are the key features of maintainable code (e.g. naming, indentation, comments, subroutines)?	Can you write a small interactive program that includes input validation and clear commenting, indentation, and naming conventions? Application: Students write a small input-checking script and apply maintainability principles (commenting, structure).
Week 3	Can you design a program that balances validation, authentication, and maintainability effectively?	Why do we test programs, and what is the difference between iterative and final testing?	What are syntax and logic errors, and how do they affect a program differently? Application: Can you identify and fix syntax and logic errors in a buggy code example? Students are given broken code samples and asked to trace, debug, and correct them.
Week 4	What is the difference between normal, boundary, invalid, and erroneous test data?	How can we design a test plan that clearly shows how a program will be tested?	How can you refine an algorithm after testing to improve accuracy and reliability?
Week 5	Application: Can you create a complete test plan for a simple program, including a table of inputs, expected outputs, and test data categories? Students use a scenario (e.g. age checker, login form) to fill in a proper test plan chart.	Application Students take an algorithm they've used and improve it based on feedback and test outcomes.	Application: Discussion-driven recap task where students review and improve example code for all elements of defensive design.
Week 6	Revision for content covered so far	Revision for content covered so far	Application: Quizziz and Exam Questions.
Half Term 4			

Date	2.4.1: Boolean Logic 2.5.1: Languages and 2.5.2: The IDE (Integrated Development Environment) Using Python		
	Lesson 1	Lesson 2	Lesson 3
Week 1	What are the symbols and truth tables for the AND, OR and NOT logic gates?	How do we use truth tables to test different logic circuits and inputs?	Can you complete truth tables and draw logic diagrams for combined gates (AND/OR/NOT)? Application: Students complete OCR-style logic diagram and truth table exam questions.
Week 2	How do we combine multiple logic gates in a circuit (e.g., AND-OR-NOT together)?	How can logic gates be used to solve real-world digital decision problems (e.g. security system logic)?	Can you design a logic diagram for a given scenario and justify how it works using a truth table? Application: Students complete design tasks where they create their own logic circuits to meet requirements.
Week 3	What are the characteristics and uses of high-level vs low-level languages?	Why do computers need translators to run our code?	Can you compare high- and low-level languages, and explain when you'd use each in a real scenario? Application: Exam-style long answer tasks comparing translator types and code examples.
Week 4	What is the difference between a compiler and an interpreter, and how does each work?	What are the advantages and disadvantages of using a compiler vs an interpreter?	Can you decide whether a compiler or interpreter is best for a given situation (e.g., game dev vs education app)? Application: Students complete a comparison table and exam-style questions based on translator choice.
Week 5	What is an Integrated Development Environment (IDE), and why is it useful for programmers?	How do editors, error diagnostics and debugging tools help improve a program?	Can you identify key features in an IDE (e.g. Thonny, IDLE, Visual Studio Code) and explain how you've used them? Application: Students explore an IDE (guided task or demo screenshots) and complete tool-matching exam questions.
Half Term 5			
Date	2.5.2 continued 1.1.1: Architecture of the CPU 1.1.2: CPU Performance and 1.1.3: Embedded Systems		
	Lesson 1	Lesson 2	Lesson 3
Week 1	What is a run-time environment, and how does it help us test and run our code safely?	How do translators and code suggestion tools work in a real IDE to support the programmer?	Can you write and debug a short program using an IDE, identifying how the tools supported you? Application: Students complete a coding task (e.g. file reader or calculator) and explain how the IDE helped.
Week 2	How can we combine logic, translators and IDE tools to build and test more complex code?	What common errors do students make in questions on logic gates, translators and IDEs and how can we avoid them?	Can you complete a mock exam paper section on logic, language types and IDE tools with full written justifications? Application: Students complete OCR-style past paper questions with a mark scheme review afterwards.
Week 3	Mock Test revision	Mock Test revision	Mock Test revision
Week 4	What is the purpose of the CPU and how does the fetch-execute cycle work step by step?	What are the functions of the ALU, CU, cache, and registers — and how do they interact during processing?	How do the components of the CPU interact throughout the fetch-execute cycle in a real program? Application: OCR-style application task where they label a CPU diagram, describe each stage of the cycle using technical terms, and explain how the MAR, MDR, PC, and Accumulator are used during the process. They then answer two past paper questions focused on CPU component roles and sequencing in the cycle.
Week 5	What is Von Neumann architecture and how do the MAR, MDR, PC, and Accumulator operate during processing?	How do clock speed, cache size, and the number of cores affect CPU performance individually and when combined?	How can we compare two CPUs based on their characteristics and explain which performs better in specific scenarios? Application: OCR-style questions that involve interpreting processor specifications (e.g., 3.2GHz quad-core vs 2.4GHz dual-core), explaining the advantages of higher clock speed or additional cache, and reasoning which CPU is best for multitasking. They finish with a structured 6-mark question comparing CPU upgrades.
Week 6	What are embedded systems and how do they differ from general-purpose computers?	What are the key characteristics of embedded systems, and why are they suitable for tasks in devices like washing machines or traffic lights?	How can we evaluate the design and use of embedded systems in a real scenario?

			Application: compare a general-purpose computer (e.g. tablet) to an embedded system (e.g. smart thermostat), describe how each is optimised for its task, and answer an OCR-style question on the advantages of embedded systems. They then design their own embedded system for a bike lock, explaining what components and features it would include.
Half Term 6			
Date	**1.2.1 – Primary Storage 1.2.2 – Secondary Storage 1.2.3 – Units 1.2.4 – Data Storage (Binary, Hex, Characters, Images, Sound) and 1.2.5 – Compression**		
	Lesson 1	Lesson 2	Lesson 3
Week 1	What is primary storage, and why does a computer system need RAM and ROM?	How do virtual memory and cache improve memory management when RAM is full or performance needs boosting?	How can we compare RAM, ROM, cache, and virtual memory in an exam scenario? Application: Identify functions of memory types, Justify when virtual memory is used, Analyse cache's role in CPU performance (J277/01-style Q2 or Q3)
Week 2	Why do computers need secondary storage, and how do magnetic, optical, and solid-state types differ?	How do capacity, speed, durability, portability, and cost affect the suitability of a storage device for a task?	Can you evaluate and justify which storage type is most suitable for different scenarios? Application: paper questions (J277/01) where they are given real-world situations (e.g., choosing storage for CCTV footage or game saves), and must select and justify the correct storage solution using specification criteria.
Week 3	What are the standard data units (bit, nibble, byte, KB, MB, GB, TB, PB), and how do we convert between them?	How do we calculate the storage requirements for sound, image, and text files using their respective formulas?	How do we apply unit conversions and file size calculations to exam-style problems? Application: OCR past paper questions where they, convert between KB, MB, and GB, Calculate the size of a .txt, .jpg, and .wav file, Justify storage decisions based on calculated file sizes
Week 4	How do we convert positive whole numbers between denary (0–255), binary (up to 8 bits), and hexadecimal (00–FF)?	How do we add binary numbers and perform binary shifts, and what is an overflow error?	How do we use binary and hexadecimal skills in OCR-style exam questions? Application: mini-assessment tasks including: Binary addition with overflow Binary ↔ Hexadecimal conversions Binary shift logic and explanations Each question is from OCR past paper or exam board exemplar content.
Week 5	How are characters stored in binary using ASCII and Unicode, and what is a character set?	How are images stored as binary using pixels, metadata, resolution, and colour depth?	How can we analyse and apply binary representation in characters and images in exam scenarios? Application: OCR-style questions including: ASCII code tables and binary character representation Calculating image file sizes Comparing two images based on resolution and depth
Week 6	How is sound captured and stored in binary, and what affects the quality and size of sound files?	What are lossy and lossless compression, and how are they used in digital storage?	Can you solve real-world storage and compression problems using OCR-style assessment questions? Application: mini-assessment including: Sound file calculations Explain differences between compression types Identify when lossy/lossless would be used (e.g. audio streaming vs text file backup)
Week 7	How do we apply knowledge of memory, storage devices, and units to solve integrated questions?	How do binary, hexadecimal, characters, images, and sound concepts link together in computer systems?	Can you tackle an OCR-style mini mock paper covering memory, storage, data representation, and compression? Application: Structured 30-minute mock covering: RAM/ROM, cache, secondary storage Unit conversions and capacity calculations Binary, hexadecimal, file sizes, and compression types Peer or teacher marks using an OCR-aligned mark scheme.

Week 8	What are the most common misconceptions in memory, storage, and binary questions and how can we fix them?	How do we approach 6-mark “explain” and “compare” questions on storage and compression topics?	Can you complete a mixed-topic final practice task demonstrating your full understanding of all 1.2 content? Final Application: Students work through 4–5 questions combining storage devices, unit conversions, file types, binary/hex, image/sound data, and compression to demonstrate full mastery. This is peer-reviewed with model answers.