Curriculum overview: Computing

Key Stage 2

Students explore:

- Website design
- Game programming using “Scratch” and “Kodu”
- Using spreadsheets
- Databases
- Physical animation using stop frame
- Photo Editing
- Movie maker
- Pod casting

Key skills/content requirements at GCSE

The Computer Science GCSE is a well-balanced course, covering topics including how a computer works, cyber security, networking, the internet, law, ethics and algorithmic thinking. Algorithmic thinking skills developed at key stage 3 are developed to enable students to solve problems that are more complex and implement their solutions with code. Students will learn the Python programming language throughout the course to build upon their problem solving skills. The course is kept up to date with up-to-date content and links to current affairs where appropriate; engaging learners with current big issues within the field of Computer Science. This course shall age well, as the fundamentals of Computer Science heavily underpin the delivery of this course, preparing students to go onto further study in the field of Computer Science with a sound understanding of the principles.

Year 7	Year 8	Year 9	Year 10	Year 11
E-safety ▪ Recognise threats to personal safety when using a computer or the internet. ▪ Identify and apply steps taken to reduce risks online. **ICT** ▪ Send emails using the school system and Outlook, utilising good email etiquette. ▪ Understand folder structure and the layout of shared information via a network. ▪ Recognise what makes good presentation design. ▪ Apply animation skills to enhance presentations. ▪ Incorporate multimedia features into a presentation. ▪ Understand how to use cell references within a spreadsheet. ▪ Apply formatting skills within a spreadsheet. ▪ Create basic formulae to automate the manipulation	**E-safety** ▪ Recognise threats to personal safety when using a computer or the internet. ▪ Identify and apply steps taken to reduce risks online. **ICT** ▪ Discuss what makes an effective website and justify reasoning. ▪ Design and create a website for a given scenario using web-authoring software. ▪ Evaluate effectiveness of a website based on given criteria and target audience. **Theory** ▪ Describe different types of network and devices used to allow systems to communicate. ▪ Evaluate how the physical topology of a network affects its performance. ▪ Model networks and evaluate their performance, applying these skills using packet-tracing software. ▪ Recognise threats to computer	**Theory** ▪ Explain how data is represented within a computer system. ▪ Convert between binary, denary and hexadecimal representation of data. ▪ Explain why hexadecimal representation is used to represent data for humans. ▪ Understand how the fetch-decode-execute occurs within a computer. ▪ Understand the difference between primary memory and secondary storage. ▪ Identify different types of storage and identify which type would be most appropriate in a certain situation. ▪ Use binary logic gates to show how binary is used within a computer to make decisions. ▪ Produce truth tables to display the results of a binary logic circuit.	**Theory** ▪ Explain how data is transferred over a network. ▪ Understand the roles of networking hardware. ▪ Identify different types of network and network topology. ▪ Suggest which network topology would be most appropriate for a given situation, justifying reasons for this. ▪ Identify different forms of malware that can affect a computer network. ▪ Suggest methods that could prevent viruses or other threats. ▪ Discuss how certain laws affect the way computer systems are used, including the Computer Misuse Act, Copyrights Designs and Patents Act, and Data Protection Act. **Problem Solving** ▪ Apply abstraction, decomposition and algorithmic thinking to solve a given scenario. ▪ Model searching and sorting	**Theory** ▪ Understand the role of computer hardware, and explain specific involvements with the fetch-decode-execute cycle. ▪ Explain the difference between primary memory and secondary storage. ▪ Identify the measures that could be taken to improve computer performance. ▪ Discern between different types of memory, and where they might be most appropriate. ▪ Describe different types of network and how the internet secures communication between devices. ▪ Describe modern threats to network security and evaluate suitable prevention and recovery mechanisms. ▪ Explain the various types of software and give examples of where they may be used. ▪ Discuss and evaluate the impact of computer systems on the wider society. ▪ Identify how certain laws affect how users interact with computer

of data. ▪ Design and evaluate the effectiveness of an application. ▪ Create an application from a given scenario. **Theory** ▪ Explain the role of hardware and peripheral devices within a computer system. ▪ Identify the most appropriate computer system for an end-user. ▪ Classify peripherals as input, output or storage devices. ▪ Explain how assistive technology makes computer systems more accessible. ▪ Convert between binary and denary representation of numbers. ▪ Perform binary addition. ▪ Explain what an algorithm is, and the importance of sequencing within an algorithm. ▪ Represent algorithms visually as a flowchart. ▪ Create sub-routines to decompose problems. ▪ Explain the role of hardware on the BBC micro:bit. **Programming** ▪ Display information and enter data on the micro:bit, making use of a visual programming language. ▪ Understand the structure of programs and ensure that code is ordered correctly. ▪ Create sub-routines to decompose problems. ▪ Specify sections of code to be run based on conditions.	systems and identify suitable preventative measures. ▪ Explain what encryption is, and apply this to protect information. ▪ Convert numbers between binary and denary representation. ▪ Understand how images and sounds are represented in a computer, and how these files are compressed. ▪ Explain character representation within computer systems, evaluating the effectiveness of ASCII and Unicode. **Programming** ▪ Select appropriate opportunities to create and manipulate variables within a textual programming language. ▪ Perform basic arithmetic within a textual programming language. ▪ Describe the difference between inputs and outputs within a computer program, and apply these within a textual programming language. ▪ Manipulate string and integer inputs as appropriate within a textual programming language. ▪ Apply selection techniques using Boolean expressions. ▪ Apply conditional loops to make a program repeat within a textual programming language. ▪ Modify a webpage using HTML scripting.	**Problem solving** ▪ Apply abstraction, decomposition and algorithmic thinking to solve a given scenario. ▪ Model searching and sorting algorithms and show the breakdown of how they work. ▪ Develop planning techniques for showing how algorithms work to solve a problem. ▪ Identify potential issues within a given algorithm and suggest a potential solution, correcting errors (syntax, logic and runtime) as appropriate. ▪ Explain how to prevent errors through use of data validation techniques. ▪ Design a test plan to test for expected, unexpected and boundary data within a program. **Programming** ▪ Decompose and use abstraction to break a problem down into smaller sections. ▪ Produce both flowcharts and pseudo-code as planning documentation to show visual representations of algorithms. ▪ Apply more advanced sequencing skills to produce programs that are more complex. ▪ Enhance application of selection through incorporation of Boolean logic. ▪ Apply both conditional and count-controlled iteration based in appropriate situations. ▪ Understand how algorithms can be used to sort and search through data, using commonly recognised sorting	algorithms and show the breakdown of how they work. ▪ Develop planning techniques for showing how algorithms work to solve a problem. ▪ Identify potential issues within a given algorithm and suggest a potential solution, correcting errors (syntax, logic and runtime) as appropriate. ▪ Explain how to prevent errors through use of data validation techniques. ▪ Design a test plan to test for expected, unexpected and boundary data within a program. **Programming** ▪ Understand and use different data structures – either 1 or 2 dimensional arrays – to solve problems. ▪ Develop file-handling skills to read, write and append data to a file. ▪ Ensure that solutions to problems meet an outlined success criteria. ▪ Evaluate the effectiveness of a solution based on success criteria. ▪ Plan solutions to a given problem using planning documentation. ▪ Apply programming skills to produce a solution for a given scenario.	systems. ▪ Interpret and solve truth tables, drawing corresponding diagrams to represent the results visually. ▪ Explain how a computer translates high-level code into machines using assemblers, compilers and interpreters. ▪ Explain how features within integrated development environments are used to detect and debug errors within a program. ▪ Explain how numbers are stored in binary representation. ▪ Convert between denary, binary and hexadecimal representation of values. ▪ Describe the units of measurement used within computer memory. ▪ Perform arithmetic with binary numbers. ▪ Explain character representation within computer systems, evaluating the effectiveness of ASCII and Unicode. ▪ Demonstrate how images and sounds are stored within a computer, explaining how differing factors can affect the size and quality of files. ▪ Select the appropriate file type for a specific purpose. ▪ Describe how compression techniques are used, identifying key differences with lossy and lossless compression. **Problem solving** ▪ Apply abstraction, decomposition and algorithmic thinking to solve a given scenario. ▪ Model searching and sorting algorithms and show the breakdown of how they work. ▪ Develop planning techniques for showing how algorithms work to solve a problem. ▪ Identify potential issues within a given algorithm and suggest a potential solution, correcting errors

▪ Create loops to iterate through code.		and searching algorithms. ▪ Incorporate functions and sub-routines into solutions to make programs more efficient. ▪ Structure data more effectively within programs using arrays. ▪ Test programs to prove effectiveness of solutions against criteria.		(syntax, logic and runtime) as appropriate. ▪ Explain how to prevent errors through use of data validation techniques. ▪ Design a test plan to test for expected, unexpected and boundary data within a program. **<u>Programming (NEA)</u>** ▪ Apply the three main programming constructs (sequence, selection and iteration) within a given programming language. ▪ Create and manipulate data within arrays within a given programming language. ▪ Decompose an algorithm into distinct sub-routines and implement as functions/procedures in a given programming language. ▪ Implement file handling techniques, including reading, writing, and appending a file as appropriate within a given programming language. ▪ Apply Boolean operators within a given programming language. ▪ Use a test table to ensure a program is reliable and robust - solving errors as appropriate, explaining solutions.

GCSE external assessment:

- Unit 1 – Computers and Hardware
  This is a formal examination lasting 1 ½ hours and covers computer systems and programming (40% of qualification).
- Unit 2 – Practical Investigation
  You will find out about a topic set by the exam board and write about what you have found (30% of the qualification).
- Unit 3 – Programming Project
  20 hours of controlled time to write a program using a programming language (30% of the qualification).