Subject: Computing **Year** 8 **Ability** Mixed

	Unit 4	Unit 5	
Topic	Algorithms	Python Programming - Sequence	Project - Programming
Topic overview **Pupils will learn…**	Understand the fundamentals of computational thinking (abstraction, decomposition, pattern recognition and algorithms) to solve a range of problems through flow diagrams and text-based programming	Understanding, through the use of text-based programming language, how to develop a range of simple programs, troubleshoot common issues and be able to identify appropriate data types and techniques to use within a given scenario	Combine the understanding of algorithms and knowledge of programming to design a solution to a problem, focused on supporting students
Components	• Understand the concepts of abstraction and decomposition in order to be able to focus on the most important aspects of a problem and break down large problems into manageable chunks • Design algorithms to solve a range of computational problems • Analyse the effectiveness of different approaches to solving problems in order to identify and apply the most efficient techniques when creating algorithms • Combine the principles of abstraction, decomposition and algorithm design with pattern recognition to solve a range of problems in order to replicate the user of algorithms to solve real world problems • Understand the benefits of a modular approach to programming to be able to develop efficient solutions for specific aspects of a larger problem • Recognise the standard symbols used in flow diagrams be able to read flow diagrams in order to visualise a solution to a problem and be able to design algorithms in the form of flow diagrams	• Understand a range of basic programming constructs in Python, including: know how to print to the screen, perform calculations, take inputs and store them in suitably named variables in order to navigate the programming language and develop basic programs that solve specific problems in the correct sequence • Develop working programs in Python to solve specific problems in order to show confidence in using its interface and understanding the correct use of syntax (rules) when writing program code • Analyse problems in computational terms, analyse the requirements of a program, identify the processes needed to solve a problem and plan creative solutions to problems in order to show understanding of computational thinking (from the previous unit) and apply it in a practical setting • Be able to read samples of Python code in order to spot errors and fix them, strengthening a fluency in use of the language in a range of unseen scenarios.	• Recall and reuse a range of basic programming constructs in Python, including: know how to print to the screen, perform calculations, take inputs and store them in suitably named variables in order to navigate the programming language and develop basic programs that solve specific problems in the correct sequence • Develop working programs in Python to solve specific problems in order to show confidence in using its interface and understanding the correct use of syntax (rules) when writing program code • Identify a real-world application for a Python program (purpose and audience) • Identify and utilise Programming knowledge from previous units that can be used to achieve specific aims • Use the design cycle to plan, develop, test and evaluate a digital product
What pupils should already know (Prior learning components)	Within the KS3 Computing, students should have been taught to – • create, reuse, revise and repurpose digital artefacts for a given audience, with attention to trustworthiness, design and usability • understand basic key algorithms that reflect computational thinking and use logical reasoning to compare alternative algorithms for the same problem • used a block-based programming language to solve a variety of computational problems • examine a set of basic flow diagrams in order to determine the sequence of an algorithm and predict outcomes (As well as recognise the basic purpose of each shape)	Within the KS3 Computing, students should have been taught to – • create, reuse, revise and repurpose digital artefacts for a given audience, with attention to trustworthiness, design and usability • understand basic key algorithms that reflect computational thinking and use logical reasoning to compare alternative algorithms for the same problem • used a block-based programming language to solve a variety of computational problems and identify the roles played by specific aspects of code • examine snippets of code and translate what is happening in simple steps • Understand the concept of sequencing within programs / algorithms and be able to place a set of instructions into the most appropriate order to solve a specific problem	Within the KS3 Computing, students should have been taught to – • create, reuse, revise and repurpose digital artefacts for a given audience, with attention to trustworthiness, design and usability • understand basic key algorithms that reflect computational thinking and use logical reasoning to compare alternative algorithms for the same problem • used a text-based programming language (Python) to solve a variety of computational problems and identify the roles played by specific aspects of code • examine snippets of code and translate what is happening in simple steps • Understand the concept of sequencing within programs / algorithms and be able to place a set of instructions into the most appropriate order to solve a specific problem
Transferrable knowledge (skills)	• Being able to access computer systems, navigate to specific files and organise work in a logical structure.	• Understand how to use selection with a programming language to make a decision and be able to write a program in Python that makes a decision based on the result of a comparison	• Being able to access computer systems, navigate to specific files and organise work in a logical structure.

	• Being able to use multiple pieces of software (such as a web browser, presentation software, word-processing software and a cloud computing system) in quick succession to create and refine design projects • Use of inference and articulation to obtain key knowledge from a topic and apply understanding when presenting findings • Being able to break down a large problem into smaller aspects and remove unnecessary details to identify key aspects	• understand how to use a calculation to make a decision and be able to write a program in Python that makes a decision based on the outcome of a calculation • Develop their understanding of the structure of the IF and ELIF statements, be able to write a program in Python that uses an if else statement and be able to use multiple IF statements to consider a number of conditions when making a decision. • understand how to use if else statements to compare strings • Be able to introduce random number generation into Python programs • and be able to use remainder after whole number division with an if else statement to decide if a number is odd or even.	• Being able to use multiple pieces of software (such as a web browser, presentation software, word-processing software and a cloud computing system) in quick succession to create and refine design projects • Use of inference and articulation to obtain key knowledge from a topic and apply understanding when presenting findings • Being able to break down a large problem into smaller aspects and remove unnecessary details to identify key aspects • Be able to use logical reasoning to identify the correct sequence of instructions when solving a computational problem • Being able to identify a specific aim for a project and the needs of the target audience.
Key vocabulary pupil will know and learn	computational thinking, decomposition, abstraction, algorithm design, algorithm, pattern recognition, input, output, flow, process, decision	decision, selection, if statement, condition, logical test, if statement, elif statement, else statement, string	computational thinking, algorithm design, algorithm, input, output, flow, process, decision, variable, selection, sequence, syntax, iterative testing
Assessment activities	• Regular low stakes testing at the end of each lesson to check knowledge. • Practical lesson activities with digital activities assessed by teachers • Do Now tasks which test previous learning and build recall on key terms and applying them to specific contexts	• Regular low stakes testing at the end of each lesson to check knowledge. • Practical lesson activities which will self-mark students' work is correct, with cells turning green when students enter the correct answer. • Do Now tasks which test previous learning and build recall on key terms and applying them to specific contexts	• Regular low stakes testing at the end of each lesson to check knowledge. • Practical lesson activities with digital activities assessed by teachers • Do Now tasks which test previous learning and build recall on key terms and applying them to specific contexts
Resources available	KS3 NC information National Curriculum - Computing key stages 3 and 4 (publishing.service.gov.uk) BBC Bitesize reference to computational thinking Computational thinking - KS3 Computer Science - BBC Bitesize BBC Bitesize reference to flowcharts Flowcharts - Designing an algorithm - KS3 Computer Science Revision - BBC Bitesize	KS3 NC information National Curriculum - Computing key stages 3 and 4 (publishing.service.gov.uk) BBC Bitesize reference to flowcharts Flowcharts - Designing an algorithm - KS3 Computer Science Revision - BBC Bitesize BBC Bitesize reference material and tutorials for Python programming Programming - KS3 Computer Science - BBC Bitesize	KS3 NC information National Curriculum - Computing key stages 3 and 4 (publishing.service.gov.uk) BBC Bitesize reference material and tutorials for Python programming Programming - KS3 Computer Science - BBC Bitesize
Notes **Why this topic is important…**	The term 'algorithm' has come to refer to any set of rules that precisely define a sequence of operations, such as making a cup of tea or cleaning your teeth. In the world of computing, an algorithm is a set of instructions that can be implemented as code to program a computer. Computational thinking is a logical, strategic approach to problem solving involving four cornerstones: decomposition, abstraction, pattern recognition and algorithm design to formulate an efficient and effective algorithm. These principles are key to being able to explore a problem and design an effective algorithm to solve it. Following on from an introduction to algorithms in the Year 8 roundelay, this unit introduces students to the use of computational thinking to solve problems. They will learn about three of the four cornerstones of the computational-thinking approach to problem solving: decomposition, abstraction and algorithm design. They will then apply what they have learnt to create a number of different algorithms, which will support them to develop logical thinking skills and support them in understanding programming and algorithm design in **GCSE Computer Science**.	There is a computer program behind just about everything we use today. Without computer programs, many things, from washing machines to aeroplanes, would not have the technological capabilities we have come to rely on. unit introduces students to writing a computer program in Python and covers taking inputs from the user, storing them in variables, calculating values using basic arithmetic operators and producing formatted output. The major data types are introduced, along with the key arithmetic operators needed to perform simple calculations in Python. The unit also looks at the concept of a list to store and manipulate multiple data items in Python and the basic manipulation of strings. The key programming construct underpinning all the work in this module is sequencing, which forms one of the three constructs that students will need to understand within GCSE Computer Science, and to be successful at programming more broadly. Throughout this module, students will use Repl.IT, a responsive online environment, to write and test their own code to solve coding challenges and develop their programming skills. There are opportunities to share	The majority of tasks within employment (and more broadly, within employment where IT skills are needed) requires a logical mindset and the ability to identify logical sequences of instructions that must be followed to achieve a specific aim. The use of computational thinking enables students to think sequentially and logically, as well as succinctly by focusing specifically on the problem. This unit will establish a real-world problem that can be solved through the development of a logical sequence of computational instructions, the outcome being a text-based programming solution. This will utilise skills obtained throughout key stage 3 Computing, such as within the **Scratch Programming unit (Year 7)**, and the **Algorithm / Python Programming units of Year 8**. It will further develop logical thinking and ensure students have a very clear idea of the areas that GCSE Computer Science will focus upon, should they decide to choose it as an option in key stage 4.

	The unit also builds upon the use flow diagrams, first introduced in the **Year 7 unit 5**, and students will learn how to use the functions of some standard flow diagram symbols and how to link them together to represent an algorithm. They will apply this learning to practical activities that will enable them to practise flow-diagram design and consolidate their comprehension of flow diagrams and how they relate to algorithms. Programming forms one of two major components within GCSE Computer Science and as such is a core element of KS3 Computing. This unit will enable students to program in a text-based language, and build upon smaller programming activities from using a block-based language within **Year 7 Computing**.	code, working in groups to collaborate and trouble shoot example programs. These skills are fundamental to the programming industry within Computing, as well as supportive of the expectations that students can develop solutions in Python within **GCSE Computer Science**. Linking back to previous content, it builds upon **algorithms unit 4 in Year 8**, and ensures that students experience programming in two different languages (including **scratch in Year 7**.)	