Digital Technologies – 5 and 6_ Creating a digital solution

	STRAND		Knowledge and understanding				Processes and production skills													
									Creating Digital Solutions by:											
			Digital Systems		Representation of data		Collecting, managing and analysing data		Investigating and defining		Generating and designing				Producing and implementing		Evaluating		Collaborating and managing	
	Content Description		Examine the main components of common digital systems and how they may connect together to form networks to transmit data (ACTDIK014)		Examine how whole numbers are used to represent all data in digital systems (ACTDIK015)		Acquire, store and validate different types of data, and use a range of software to interpret and visualise data to create information (ACTDIP016)		Define problems in terms of data and functional requirements drawing on previously solved problems (ACTDIP017)		Design a user interface for a digital system (ACTDIP018)		Design, modify and follow simple algorithms involving sequences of steps, branching, and iteration (repetition) (ACTDIP019)		Implement digital solutions as simple visual programs involving branching, iteration (repetition), and user input (ACTDIP020)		Explain how student solutions and existing information systems are sustainable and meet current and future local community needs (ACTDIP021)		Plan, create and communicate ideas and information, including collaboratively online, applying agreed ethical, social and technical protocols (ACTDIP022)	
Sequence of Lessons / Unit	**Approx. time rq'd (hrs)**	**Year 5or 6**	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #
Problem-solving processes	16 hrs	A	☐		☐		☐		☑	3	☐		☑	4	☑	4	☐	5	☐	

Levels 3 and 4 Achievement Standard	**Levels 5 and 6 Achievement Standard** The numbering of the Achievement Standards below is reflected in the grid above to show coverage across the 8 units.	**Levels 7 and 8 Achievement Standard**
By the end of Year 4 • Students describe how a range of digital systems (hardware and software) and their peripheral devices can be used for different purposes • They explain how the same data sets can be represented in different ways. • Students define simple problems, design and implement digital solutions using algorithms that involve decision-making and user input. • They explain how the solutions meet their purposes. • They collect and manipulate different data when creating information and digital solutions. • They safely use and manage information systems for identified needs using agreed protocols and describe how information systems are used.	By the end of Year 6: • Students explain the fundamentals of digital system components (hardware, software and networks) and how digital systems are connected to form networks. (1) • They explain how digital systems use whole numbers as a basis for representing a variety of data types. (2) • Students define problems in terms of data and functional requirements and design solutions by developing algorithms to address the problems. (3) • They incorporate decision-making, repetition and user interface design into their designs and implement their digital solutions, including a visual program. (4) • They explain how information systems and their solutions meet needs and consider sustainability. (5) • Students manage the creation and communication of ideas and information in collaborative digital projects using validated data and agreed protocols. (6)	By the end of Year 8 • Students distinguish between different types of networks and defined purposes. • They explain how text, image and audio data can be represented, secured and presented in digital systems. • Students plan and manage digital projects to create interactive information. • They define and decompose problems in terms of functional requirements and constraints. • Students design user experiences and algorithms incorporating branching and iterations, and test, modify and implement digital solutions. • They evaluate information systems and their solutions in terms of meeting needs, innovation and sustainability. • They analyse and evaluate data from a range of sources to model and create solutions. • They use appropriate protocols when communicating and collaborating online.

Problem-solving processes

Year Level 5 TOPIC Creating a digital solution Time: 16 HOURS

When students are set the task of solving a problem that requires a digital solution, they usually start by investigating and defining the problem. They draw on computational thinking, a problem-solving approach that involves activities such as organising data logically, breaking down problems into components, and designing and using algorithms and models to show how the solution will be developed and how it will appear. As part of designing their solution, students generate ideas and consider the user of their digital system. During the producing and implementing process students typically create their own solution using a visual programming language. Once a digital solution has been created it is important to evaluate it against relevant criteria, such as: Did it entertain the users (if a game)? Can updated data be added so the solution can be used in the future? (Future needs). Note: Sometimes when students are creating digital solutions they might return to a process they have already completed in order to make adjustments; however, typically at this level, students engage in each of these processes in the above-mentioned order.

Programming is the way we communicate algorithms to a digital system, such as a laptop or notebook, so that the system understands the instructions. Digital systems need precise instructions as they are unable to understand instructions that include superfluous details. We use programming languages to code the instructions. There are many different visual programming languages but all have common programming statements and use a common approach to creating a program and running it to see if it works as intended.

Flow of activities				
Short text	Define the task Analyse the problem and identify the functional requirements of the solution.	Design a digital solution Represent how the solution will be created and what the solution will look like	Visual programming solution Use a visual programming language to implement a digital solution.	Evaluation Evaluate the level to which the solution met the needs of the target audience or its intended purpose.
Question to guide exploration	What is the problem?	How should the digital solution work and what should it look like?	How do I transform my design into a working solution?	Did the solution solve the problem?
AC Alignment	*Investigating and defining (ACTDIP017)*	*Generating and designing (ACTDIP019)*	*Producing and Implementing (ACTDIP020)*	*Evaluating (ACTDIP021)*
What is this about?	Clearly defining a problem is a crucial step in developing a software solution to a problem. This is the process students undertake when they analyse the problem and identify the functional requirements of the solution. Students determine what the solution has to do to solve the problem (eg accurately count the number of guesses before the next question appears or calculate the distance travelled by a robot). Defining the problem involves identifying the 'pieces of the jigsaw': the main elements or components of the problem, and the data needed to better understand or solve the problem. Defining involves stating what would solve the problem, not how to solve the problem. When the question 'What is the problem?' is answered, the process moves to how the problem will be solved. The solution is found. Then algorithms are designed to represent a complete, logically structured set of instructions that are needed to solve the problem. Students examine existing digital solutions to identify features that may be transferable to new but similar digital solutions.	Designing involves representing how the solution will be created and what the solution will look like (user interface). It is the 'how' process. An algorithm is a step-by-step process or series of instructions to achieve a particular outcome. It is used to show how the solution will function – it is the rules, the sequence and decisions. Algorithms can be written as a series of steps or drawn as a flow chart. Creating an algorithm is an integral part of computational thinking and of creating a program to instruct a computer or robotic device. Computational thinking helps break down a complex task into smaller chunks. Looking for patterns in the algorithm helps us work out opportunities to use loops where code is repeated. A paper prototype can be used to map out design ideas; for example, what is on screen, the logic behind transitioning between screens and how various elements work together as a system. The paper prototype can inform algorithm development. Once the algorithms have been completed they are converted into a program, so that those instructions can be executed by the digital system. At this level, students use a visual programming language.	A visual programming language enables students to sequence commands (displayed as blocks) to create a program (or digital solution). This could be a simple task of animating a character (sprite) in a story; or it could be creating complex programs to model a real-world application. In programming languages, decisions (branching) are implemented using if/then or if/then, else statements. Repetition is implemented using loop statements. As students form their sequential blocks, they can introduce the repeat/loop block to avoid repetition in code as a more advanced aspect of sequences. For example, instead of putting the same single blocks one after the other, we can wrap an iteration (repeat/loop) block around blocks they would like to repeat, to tell the computer to execute the code a certain number of times. Repeat loop blocks allow us to set a value to control how many times the loop is executed. For example, when creating a quiz, the questions are repeated until the correct response is given. User input is a way the user interacts with the computer program. For example, a user might click on a sprite or avatar in a game or animation to make it react in some way, or they could enter their name or a quiz answer when prompted. When we think of input and output, we can characterise the images on the screen and sound as output. Input is anything that provides some information to our program – such as a click of the mouse or entered text, which in turn will activate or modify a process.	Evaluation takes places at two levels. The 'micro' level is where students judge if the solution they created met the functional requirements identified in the defining process. The 'macro' level, which takes a broader view, asks students to consider how their and existing solutions used in information systems, such as a library borrowing system, would be judged on the basis of being sustainable and able to meet the current and future needs of a community. Sustainability includes factors such as the energy levels required to operate the solution and other resources used such as paper for printed output. Future needs could include whether new data, such as new library books and DVDs, could be used in the solution. Evaluating draws on systems thinking where students need to consider how the outputs (solution) meet and affect the users.
The focus of the learning (in simple terms)	Model how to define a problem. First, take a familiar problem or one that students have an interest in. Next,	Provide opportunities for students to design, modify and follow simple algorithms. Share the algorithms and	Use Scratch, Tynker, Snap or other similar visual programming language to develop a quiz, interactive	Use sustainability criteria to explain how well students' solutions meet requirements; for example, the solution

	brainstorm as a class what is known. For example, 'You want to share digital photographs safely online with your friends'. Some things that may come up in brainstorming include: • that photos are taken and stored on a smart device • how the photos are shared (what type of platform or process is used) • how people receiving the photo know it is safe to download to their device • what file size is acceptable • whether the user can comment on the photo, and if so how • what user data is needed to send the image • how users are notified • whether a photo can be tagged • whether a photo can be edited before being sent • whether users need a profile; and if so how they can remain safe online. Brainstorm a list of ideas for problems that require students to create a digital solution. Use these problems as a focus of designing, implementing and evaluating the solution. Students could use mindmaps to identify the elements of each problem and then show the connections between each element – this will help them to group common factors.	discuss how the decisions (branching) and repeat instructions (iteration) are represented in flow charts or written as simple steps. For students less familiar with designing an algorithm perhaps start with an algorithm that has some missing steps, too many repeated steps or steps out of order. The task is to debug the algorithm to make it work. Share revised algorithms. Present the challenge of taking a familiar task, for example cooking a hard-boiled egg, and writing it as a series of steps with or without drawings. Use a think-pair-share strategy to combine ideas to make a group-designed algorithm. This task could be extended by adding decisions to allow for choice (such as the ways it is cooked: fried, poached or scrambled). What steps are common? A flow chart is a relevant way to present the algorithms. Discuss the list of problems developed as part of a class brainstorm or set a problem that suits your class context. Set the task of designing an algorithm to solve a particular problem of interest. Ask students to develop a paper prototype and share with others to gain feedback. For the user input aspect, students could compare the user interface of familiar apps or games and identify the features that contribute to 'good design', such as easy navigation, appropriate size of icons, attractive colours and interesting and appropriate graphics. At this level, students begin to generate different design ideas before selecting the one that best allows the problem to be solved. Once the preferred idea has been chosen it is fully developed as an algorithm and a layout diagram of the user interface is created.	story or to simulate a real world application. Ensure that the design phase includes algorithm development and consideration of user input. Students evaluate their implemented design. Provide students with an existing program created in a visual programming language that has the option for remix; for example, a Scratch project. Students can modify the program to meet a similar need. Use a turtle-based program such as Pencil Code to create and draw geometric shapes and designs. Integrate mathematics geometry with the programming of geometrical shapes. Concepts such as angles, and properties of 2D shapes, directions and Cartesian planes, can be investigated. Use the BBC Micro:bit emulator to program. Use a visual programming language or a relevant app to control a robotic device such as Sphero, Dash and Dot, Edison, mBot, Lego EV3 or a mini-drone. Set up challenges that require students to design and implement program solutions. Sphero can be used as the movement source for many open-ended challenges.	can only be viewed on screen to avoid printing (environmental). Students could write their criteria in the form of questions and focus on one factor contributing to sustainability. Factors could be economic, environmental or social. Economic factors include the cost of producing or running the solution. Environmental factors are the resources needed to use the solution. Social factors could include fairness of rules and accessibility to specific audiences. Students' solutions may incorporate icons or buttons or simple forms of navigation to improve user experience. Ask students to evaluate the use of these approaches where applicable.
Supporting resources and tools and how to use them	Learning environments This lesson sets up the problem of creating the most suitable conditions for learning. Check out the checkout This lesson sets up an approach to model how a checkout works.	Learning to loop Students create algorithms with a condition that tells the computer to repeat a sequence of instructions. Making maths quizzes 1: Plan and test our programs This lesson sequence provides guidance on how to design an algorithm. The context is a maths quiz that gets harder or easier depending on user performance. Eco-calculator This lesson sequence provides guidance on how to design an algorithm. The context is making a paper prototype of an eco-calculator to demonstrate human impact on the environment and to suggest changes in behaviour.	Course C This Code.org course is an introduction to creating programs with loops, events and conditionals. Scratch tutorials Follow these tutorials to get started with your project. Swift Playgrounds This is a free iPad app that students can use to learn programming. Design a flag with Pencil Code Create a step-by-step process (algorithm) to program your flag design after exploring a 'block-based' turtle drawing program such as Pencil Code. Making maths quizzes 2: Implementing a digital solution In this sequence of lessons students implement a digital solution for a maths quiz. They test and assess how well it works. BBC Micro:bit code maker	Software evaluation checklist Students could select some checklist points for their solutions and convert them into questions to assist with evaluation.

			This online editor makes it easy to program your Micro:bit in Blocks. You don't even need a Micro:bit. DT Challenge: 5/6 Blockly – Chatbot Set up a login for students to learn about user input. As they complete the modules students write brief programs that prompt the user for input, store it in a variable and then print it – often in combination with some other text, such as a greeting. Sphero: Catch me if you can This is a good introduction to Sphero with challenges and tasks for students to develop an understanding of ways to control Sphero through programming. There are apps to control Sphero; Sphero Edu is recommended.	
Assessment	**Assessment task examples** • Mindmap showing the key components of the problem • List of data needed to solve the problem • List of the functional requirements of the solution **Achievement standard** Define problems in terms of data and functional requirements and **design** solutions by **developing** algorithms to address the problems.	**Assessment task examples** • Use a flow chart to design an algorithm to process user input into a simple maze game. • Use a flow chart to design an algorithm that includes branching and iteration. • Use a repeat loop block in a visual programming language to demonstrate understanding of branching and iteration. (Write an algorithm for a game that uses repetition.) **Achievement standard** Define problems in terms of data and functional requirements and **design solutions by developing algorithms to address the problems.** Incorporate decision-making, repetition and user interface design into their designs and **implement** their digital solutions, including a visual program	**Assessment task** Modify an existing game or quiz: • to suit learners of a different age group • to make it more fun. Dr Scratch is a free online analytical tool that provides feedback on Scratch (MIT) project progress. **Achievement standard** Incorporate decision-making, repetition and user interface design into their designs and **implement their digital solutions, including a visual program.**	**Assessment task examples** Write three evaluation questions about an existing game, such as: • Are the rules easy to understand? • Are the rules fair? • Can you change an answer or action easily? For an information system used in the local community, such as a directory kiosk in a shopping centre or a supermarket check-out system, list three functions or pieces of information that it currently performs or shows. Then list three changes that would need to be made so that it can perform or show an element in one year's time; for example, a new price for a product or the removal of a store. **Achievement standard** **They explain how information systems and their solutions meet needs and consider sustainability**