Digital Technologies – 3 and 4_Digital solutions Digital solutions

Strand			Knowledge and understanding				Processes and production skills									
			Digital systems		Representation of data		Collecting, managing and analysing data		*Creating digital solutions by:*							
									Investigating and defining		Producing and implementing		Evaluating		Collaborating and managing	
Content Description			Identify and explore a range of digital systems with peripheral devices for different purposes, and transmit different types of data (ACTDIK007)		Recognise different types of data and explore how the same data can be represented in different ways (ACTDIK008)		Collect, access and present different types of data using simple software to create information and solve problems (ACTDIP009)		Define simple problems, and describe and follow a sequence of steps and decisions (algorithms) needed to solve them (ACTDIP010)		Implement simple digital solutions as visual programs with algorithms involving branching (decisions) and user input (ACTDIP011)		Explain how student solutions and existing information systems meet common personal, school or community needs (ACTDIP012)		Plan, create and communicate ideas and information independently and with others, applying agreed ethical and social protocols (ACTDIP013)	
Sequence of Lessons / Unit	**Approx. time rq'd**	**Year**	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #	CD	Achievement standard #
Intro to programming	8?	3	☐		☐		☐		☑	3	☑	3	☐		☐	

Years F-2 Achievement Standard	**Years 3 and 4 Achievement Standard**	**Years 5 and 6 Achievement Standard**
By the end of Year 2 • Students identify how common digital systems (hardware and software) are used to meet specific purposes. (1) • They use digital systems to represent simple patterns in data in different ways. (2) • Students design solutions to simple problems using a sequence of steps and decisions. (3) • They collect familiar data and display them to convey meaning. (4) • They create and organise ideas and information using information systems, and share information in safe online environments. (5)	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. (1) • They explain how the same data sets can be represented in different ways. (2) • Students define simple problems, design and implement digital solutions using algorithms that involve decision-making and user input. (3) • They explain how the solutions meet their purposes. (4) • They collect and manipulate different data when creating information and digital solutions. (5) • They safely use and manage information systems for identified needs using agreed protocols and describe how information systems are used. (6)	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)

Intro to programming

Year 3 TOPIC Digital solutions Time: 8 HOURS

Programming is one process of the larger problem-solving methodology of creating digital solutions. Using a programming language can create a solution to a problem. The starting point for the problem-solving methodology is finding out about (investigating) and working out (defining) the problem. Once the problem has been defined the next step is to represent the solution as a series of steps (an algorithm). The algorithm can highlight any decisions (branching) that need to be made and what pathways might result, as well as how a user might engage and provide input. Algorithms at this level might be described verbally, written as a series of steps, represented on card, drawn or created digitally. The algorithm may then be implemented using a programming solution where students use a visual programming language that involves dragging and dropping programming blocks into a sequence. The final process is to evaluate how well their solution solved the problem.

Flow of activities				
Short text	Define the problem Define a problem drawing on computational thinking and draw some conclusions about its features or needs.	Coming up with a solution Create a storyboard or flow chart to record relationships between the content and processes.	Implementing a solution Use a visual programming language as part of the digital solution.	Evaluation Evaluate how well the solution met the desired outcome.
Questions to guide exploration	*What is the problem?*	*How can I describe the solution?*	*What programming skills do I need to create a digital solution?*	*How do you know if my solution is OK?*
AC Alignment	Investigating and defining (ACTDIP010)	Investigating and defining (ACTDIP010)	Producing and implementing (ACTDIP011)	Evaluating (ACTDIP012)
What's this about?	When trying to solve a simple problem an important first step is to define the problem. This helps us work out a relevant solution. A simple problem is one that has a straightforward solution. At this level the students should attempt to create a solution that meets a common personal, school or community need; however, for this unit the focus is restricted to a personal or school need. At this level, defining a problem initially involves students summarising the facts or features of an existing problem or an opportunity (being proactive in creating a solution rather than responding to a problem) so they can draw some conclusions about it. Note: Typically, when defining problems you do not ask 'how' as this belongs to the process of designing a solution (how can a solution be created?) Once the problem has been defined, the solution (how it can be solved) can be represented as an algorithm.	Students need to determine how the solution will be created. This is done by stating or following an algorithm. At this level the algorithm can be verbal or drawn using a combination of images and text or perhaps by sequencing ready-made cards. The algorithm needs: • to be clear and explicit in its instruction (as the computer will only do what it is told) • to be sequenced in the correct order • to have minimal steps. Problems that require a digital solution often need content that is prepared or sourced prior to programming the solution. For example, for a simple guessing game where the user selects the correct coding instruction to create a geometrical shape, the content includes the correct coding sequences in a suitable language. Consideration is also given to the kinds of processes to be used; for example, checking the coding instruction using a Pro-Bot or a turtle program such as Pencil Code to draw the shape. Creating storyboards or flow charts helps to record relationships between the content and processes. Once the plan (algorithm) for the solution is completed, students follow the instructions using their visual programming language.	Visual programs are made by dragging and dropping blocks to create a computer program. Each block represents a different instruction. In most visual programming languages, the blocks are grouped by colour with each one signifying a particular function; for example: movement or adding sound. They could also signify traditional instructions to control decisions such as the repeat loop instruction. At this level, students need skills in using a programming language that creates options for users to make choices in solutions. For example, a user input and branching mechanism such as buttons in a slideshow or selecting a different sprite to follow an alternate pathway in a story created in Scratch. Programmable robotic devices that use a visual programming language can be used in engaging ways to develop computational thinking skills. There are many types of robotic devices available, each with its own mobile app to enable control and/or programming. Robotic devices can be used effectively across a range of learning areas.	After students implement their digital solution it is important that they evaluate it. One form of evaluation considers how well the solution met the desired outcome and in particular the problem defined at the beginning of the process. Did the content and processes used, contribute to a suitable solution? Did the solution meet a common personal or school need? At this level, students use the criteria to determine how well their solution met one or more of the following needs: • personal, such as entertainment • school, such as music learning • community, such as facilities at local parks. For this unit, students should restrict their criteria to personal or school needs.
The focus of the learning (in simple terms)	Defining a problem involves computational thinking. Support students to break down problems (or needs) into smaller parts, focus on the key elements and ignore detail that may not be required, look for any patterns and create an algorithm. Defining a problem can be difficult – it is very tempting for students to make comments such as 'I want a guessing game' or 'the robot needs to move around' or 'something doesn't work and it needs fixing'. Students need to be able to summarise the	Students follow an algorithm for simple or familiar tasks; for example, their morning routine, learning a sporting skill, playing a card game, making a paper plane or providing directions to a hidden item. Use a range of ways to represent the algorithm; for example, on cards, verbally, or in a combination of images and text. A fun way to demonstrate the need to be precise in your instructions is to model a simple task such as making a jam	Use an unplugged activity where students become familiar with the various visual programming blocks. Provide laminated colour printed versions of the main blocks such as motion, looks, sound and control. Students use non-permanent markers and change values on coding blocks (for example the angle of turn, the number of steps or text in a 'say' block). Provide some challenges such as sequencing blocks to create	The focus of learning in this unit is on students evaluating their own solutions – they will progress to evaluating existing information system solutions in the next unit. Provide opportunities for pairs of students to review each other's solution, identifying two features that they like and two features that could be changed (maybe to suit a different audience) or improved.

	facts or characteristics of a future need/solution so they can draw conclusions about it. This involves being able to ignore some less important information in order to grasp the key features. The process of defining a new **need/opportunity** can be guided by key questions such as: • Who would like this solution (who is the audience)? • Why does this opportunity exist (why should this solution be created)? • What need would be met by this solution (what should the solution be able to do)?	sandwich. By only doing exactly as the instructions command all sorts of humorous mistakes can be made. This task can be used to explain the need to review and rewrite a program. Design a sequence of instructions using words and/or symbols for others to follow. For example: *Arranging blocks* One student creates an image made up of several attribute blocks. The student then instructs their partner to recreate the image by selecting from a pool of blocks. This task uses technical language of correct shape name, colour and size as well as directional language: above, beside to the left or right. Discuss the algorithms: How easy was the algorithm to follow? Were the steps in order? Did the steps lead to the completion of the desired outcome? Asking procedural type questions (How do you build …? How do you make …?) may give rise to the need for instructions to solve the problem. These instructions are considered an algorithm, a series of step-by-step instructions to complete a desired task. Depending on the problem, a digital solution may be required; for example, as with these design-based questions: • How do I create a digital story with more than one ending? • How can I control a robot to move through a maze? • How can I help someone learn words in another language?	simple programs; for example, use a control block to make the sprite move, say something or perform an action. Provide an introductory programming tutorial to learn the basics of visual programming. Courses such as those provided by Code.org are a great place to start. Refer to some of the Disney-inspired coding challenges or those that provide guidance to create a simple game. Provide access to programmable robotic devices that do not require visual programming, such as Bee-Bot or Pro-Bot. Students develop their computational thinking skills. They complete challenges such as creating a maze game with a Bee-Bot; programming a Pro-Bot to draw geometric shapes and designs; or programming a Pro-Bot to fulfil a range of actions such as moving, making sounds and avoiding obstacles. Prior to programming more complex commands students create their own algorithms using a combination of arrows, symbols, words and images. Using a Pro-Bot, decisions can be incorporated using 'if' commands and sensors. For example, if the robot moves into darkness turn on the headlights. Integrate mathematics and geometry. Design a simple guessing game that presents several possible coding options for some geometric shapes. The selected coding instruction is input into a turtle drawing program such as Pencil code or into a Pro-Bot to check whether the correct option was selected. Prepare the content (shapes and coding instructions) and plan an algorithm using a flow chart.	The process of evaluation should be directly linked to the statement defining the problem, remembering that the context of the solution should be to meet either a common personal or school need.
Supporting resources and tools and purpose/ context for use.	Creating-digital-solutions General advice about the problem solving process related to creating a digital solution to address a problem.	Introducing algorithms Students design a sequence of steps for others to follow. Making a jam sandwich This video shows a collection of algorithm errors, and great debugging opportunities. Real-life algorithms: Paper planes Use an algorithm to make a paper airplane.	Pro-Bot lessons from Simon Haughton Pro-Bot robotics, Bee-Bot Background information about commands for Pro-Bot. Move my robot A wealth of ideas to program a Pro-Bot. Balloon pop Design a course challenge for another user which will result in the Bee-Bot, with a pin attached, reversing into a balloon to pop it. Pencil code An easy-to-use turtle drawing program freely available online (uses block-based programming). Code your own sports game Learn to code by following this tutorial about the basics of visual programming. Code with Anna and Elsa: An Hour of Code tutorial This is an introduction to coding and computer science in a safe, supportive environment.	Again, we can only refer to the evaluation resource mention in the levels 5 and 6 resources.

			Moana: Wayfinding with code This Disney Hour of Code tutorial uses a visual programming language where students drag and drop visual blocks to write code.	
Assessment	**Suggested approaches** For a new need or opportunity, identify: • who the audience is • what the main purpose/function of the solution is • why the audience would like the solution. **Note:** Students can present their problem or need definition as a digital, oral or written statement. **Achievement standard** **Define simple problems,** design and implement digital solutions using algorithms that involve decision-making and user input.	**Suggested approaches** Are students able to show one example of branching and one example of user input in the algorithm? Each student reads aloud part of another student's algorithm. **Achievement standard** Define simple problems**, design** and implement **digital solutions using algorithms that involve decision-making and user input.**	**Suggested approaches** Presentation or demonstration of one aspect of the solution that the student thinks helps solve the problem (this also addresses evaluation). **Achievement standard** Define simple problems, design and **implement digital solutions using algorithms that involve decision-making and user input.**	**Suggested approaches** • Presentation or demonstration • Complete the statement – I like my solution because … • Complete the statement – next time I would … • Select one feature of the solution and describe how it helped meet a personal or school-related need/opportunity. **Achievement standard** **They explain how their solutions meet their purposes.**