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	Investigate how data is transmitted and secured in wired, wireless and mobile networks, and how the specifications affect performance (ACTDIK023)	Investigate how digital systems represent text, image and audio data in binary (ACTDIK024)	Acquire data from a range of sources and evaluate authenticity, accuracy and timeliness (ACTDIP025)	Analyse and visualise data using a range of software to create information, and use structured data to model objects or events (ACTDIP026)	Define and decompose real-world problems taking into account functional requirements and economic, environmental, social, technical and usability constraints (ACTDIP027)	Design the user experience of a digital system, generating, evaluating and communicating alternative designs (ACTDIP028)	Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors (ACTDIP029)	Implement and modify programs with user interfaces involving branching, iteration and functions in a general-purpose programming language (ACTDIP030)	Evaluate how student solutions and existing information systems meet needs, are innovative, and take account of future risks and sustainability (ACTDIP031)	Plan and manage projects that create and communicate ideas and information collaboratively online, taking safety and social contexts into account (ACTDIP032)

Sequence of Lessons / Unit	Approx. time rq'd	Year A or B	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 #	CD	Achievement standard #
Create an app or a game	16	7	☐		☐		☐		☐		☑	4	☑	5	☑	5	☑	5	☑	6	☐	

Years 5 and 6 Achievement Standard	Years 7 and 8 Achievement Standard	Years 9 and 10 Achievement Standard
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. • They explain how digital systems use whole numbers as a basis for representing a variety of data types. • Students define problems in terms of data and functional requirements and design solutions by developing algorithms to address the problems. • They incorporate decision-making, repetition and user interface design into their designs and implement their digital solutions, including a visual program. • They explain how information systems and their solutions meet needs and consider sustainability. • Students manage the creation and communication of ideas and information in collaborative digital projects using validated data and agreed protocols.	By the end of Year 8 • Students distinguish between different types of networks and defined purposes. (1) • They explain how text, image and audio data can be represented, secured and presented in digital systems. (2) • Students plan and manage digital projects to create interactive information. (3) • They define and decompose problems in terms of functional requirements and constraints. (4) • Students design user experiences and algorithms incorporating branching and iterations, and test, modify and implement digital solutions. (5) • They evaluate information systems and their solutions in terms of meeting needs, innovation and sustainability. (6) • They analyse and evaluate data from a range of sources to model and create solutions. (7) • They use appropriate protocols when communicating and collaborating online. (8)	By the end of Year 10 • Students explain the control and management of networked digital systems and the security implications of the interaction between hardware, software and users. • They explain simple data compression, and why content data are separated from presentation. • Students plan and manage digital projects using an iterative approach. • They define and decompose complex problems in terms of functional and non-functional requirements. • Students design and evaluate user experiences and algorithms. • They design and implement modular programs, including an object-oriented program, using algorithms and data structures involving modular functions that reflect the relationships of real-world data and data entities. • They take account of privacy and security requirements when selecting and validating data. Students test and predict results and implement digital solutions. • They evaluate information systems and their solutions in terms of risk, sustainability and potential for innovation and enterprise. • They share and collaborate online, establishing protocols for the use, transmission and maintenance of data and projects.

Create an app or a game

Use the context of apps and digital games development to build students' capabilities and confidence in creating a digital solution that uses a general-purpose (text based/scripting) programming that allows for choices (branching) and repetition (iteration). There is a

wide range of interactive online tutorials that students could work through to learn and practise coding for a particular programming language, such as Python and Ruby. Once students have determined the purpose and requirements of the game, they describe how the solution will be created and consider design features appropriate to the audience. After the game has been developed, students evaluate its success.

Flow of activities				
Short text	*Examine existing digital solutions* *Consider the user when defining a problem and identifying the functional requirements.*	*Designing a solution* *Use their functional requirements as the basis for developing an algorithm and the user interface.*	*Programming a solution* *Use a relevant programming language to implement their digital solution.*	*Evaluate* *Evaluate their digital solution against design criteria and user needs.*
Questions to guide exploration	*Who's the game for and what's its purpose?*	*How can I represent my design of the solution?*	*How can I code my solution?*	*Have I met the user's needs?*
AC Alignment	Investigating and defining (ACTDIP027)	Generating and designing (ACTDIP028) / (ACTDIP029)	Producing and implementing (ACTDIP030)	Evaluating (ACTDIP031)
What's this about?	It is very important to clearly state the intention of the game, namely what it is required to do (functional requirements) as well as identify if there are any constraints or factors that should influence the nature of the game or how it is developed. An important aspect of this process is developing empathy for the user. What possible constraints (usability) might the user have? Are there any accessibility needs such as having both sound and images to indicate an action (social constraint)? Are there any technical constraints such as a game requiring a specific input device to issue instructions such as a 3D mouse or a Gamepad (technical constraint)? Also user preference is important in this process, for example the users might prefer touch screen over device inputs. At this level students are required to decompose a problem. For example for an adventure game it could mean identifying for each character their characteristics, actions, settings and sequences. Decomposition is about isolating the key elements and then teasing out features of each. This allows identification of patterns, relationships and anomalies. Once these aspects are understood, the process then moves on the identification and design of algorithms that represent a complete, logically structured set of instructions that are needed to solve the problem as well as factors contributing to user experience. Examining existing digital solutions enable students to identify features that may be transferable to new but similar digital solutions. Look for cross-curricula opportunities when designing an app. For	An algorithm is a logical step-by-step process for stating how to create a digital solution. Algorithms are generally written as a flowchart or in pseudocode. **Note:** There is not an expectation that both algorithm techniques are used when designing one solution. A flow chart is a common way to visually represent an algorithm. Another relevant approach particular for games and apps is to do a storyboard which often focuses on the onscreen actions. Pseudocode is a way of describing a set of instructions that does not have to use specific syntax. At the level students use structured English to express these instructions, for example using 'while' and 'endwhile' when describing a 'while loop'. When designing how the solution is created students need to refer back to any constraints identified when defining the problem, such as social and technical ones. The design of the user interface (drawing on design principles such as contrast, space and balance, and repetition) and consideration of these constraints is referred to as user experience.	It can be expected that students at this level may be transitioning from block-based visual programming languages to general-purpose (text-based) languages. **Note:** General-purposes languages allow students to solve more complex problems as they are not restricted by the functionality of visual programming languages. Some block-based visual programming languages such as the app Tynker, provide the equivalent programming instructions in a text-based language. Python and Java are programming languages commonly used in schools. At this level, students begin to test their solutions and make changes to the program if needed. It is a good idea for students to plan what tests they will conduct before they start coding (expected/actual results). For example, students might test if the allowed number of repeated actions in their game is the same as they planned or if a navigation path takes them to the destination stated in the design.	The process of evaluating involves judging if the digital game met its purpose. Evaluating involves using criteria to make that judgement, and at this level, students can determine the value of their game/app based on criteria related to one or more of the following: the stated requirements, innovativeness, sustainability and risks. For example, students might evaluate their games on: • how well they meet user needs • how innovative their solutions are compared to existing games • how sustainable their solution will be for different users, purposes, and technology improvements • if there were any social/ethical risks with their games such as exclusion, bullying, links to inappropriate websites.

	example, make links to History: What would a smartphone contain from someone in ancient times? [contacts, maps, notes, SMS, photos etc]			
The focus of the learning (in simple terms)	Students can begin this process by researching and reviewing a range of existing games or apps: • For game design: see what makes them fun to play and identify elements that are less engaging. • For Mobile Apps: Look for common element such as: Login-ins, Welcome page, Navigation, Graphic user interface. As a result of their investigation students identify features that may be of use in their digital solution. Model ways to record ideas such as sticky notes, a checklist or table to record details of their game or app research. Set the task of identifying what a digital game or mobile app for a particular audience needs to do (functional requirement). For example: • Digital game: a word game that rewards the selection of the correct word or a maths game that rewards selection of a particular shape. A more complex requirement is a game that allows multiple users to work with multiple options. • Mobile app*:* a school map that shows the location of buildings and rooms with a navigation path. Using an organisation chart template such as Smart Art or a mindmap students can break the problem down into sub elements for example in game design show the relationship between characters, movements, collisions and scoring. Students should identify at least one constraint on the solution and investigate how that constraint is handled with existing games or apps. Use different approaches to organise collaborative groups so students have to work with a diverse range of students with different skills, abilities and backgrounds. Implement strategies to help students work effectively. Support and guide students with the planning and project management.	Students use their functional requirements as the basis for developing an algorithm and the user interface. At this level, students generate two or three different design ideas (ideas as to how the game/app will operate and look). These are just broad ideas with not a lot of detail. This process draws on creative thinking skills (Critical and Creative Thinking general capability) and students use their functional requirements to judge what idea best meets these requirements. This might involve students undertaking interviews of the target audience to support the user-centred design process. They may offer screen options or initial design ideas to gauge what users like and dislike. This helps students select the best design idea for further development. A paper prototype can be used in the design process to map out ideas for example what's on screen, the logic behind transitioning between screens and how various elements may work together as a system. The paper prototype can inform algorithm development. Use the algorithm to identify parts of the program that involve branching (where decisions by the user are enabled), iteration (where loops and repeat functions have reduced the script length and detail) and other functions that might have been suggested for example the use of variables. Students describe their algorithmic steps to others, and have other students interpret their algorithms and give feedback to improve accuracy and clarity of their instructions. Raising questions about the	Students could explore similarities/differences and discuss the advantages/disadvantages of both categories of programming languages (visual/text). Students could undertake various online courses and modules to increase and further develop their programing skills and knowledge. For students transitioning between visual and text-based programming provide an opportunity to create a game using Scratch, Pyonkee, Tynker or similar programming language. Once familiar with a range of coding commands and ways to manipulate data, students could implement their game or mobile app design to meet their particular purpose and audience. Students should test some limited features of their solutions to make any changes, if needed. They might like to pair with a student and each complete one test of their partner's solution.	Students evaluate their solutions on the basis of how well they meet user needs, how innovative their solution is compared to existing solutions, and how sustainable their solution will be for different users, purposes, and technology improvements. It is often easier for students to frame these criteria as questions. This guides their evaluation and also focuses attention quite specifically on the functional requirements. For example: • Was it easy to log on to the app? • Could you change an action if you made a mistake? • Is an action shown as an image as well as a noise? For example, clapping when a score was shown onscreen. • Are answers/responses displayed on the screen quickly enough? • Are the button/icons big enough so you can easily select the correct target? • Was there something different about this game/app that you had never seen before?

		sequence helps students articulate their programing intentions.		
Supporting resources and tools and purpose/ context for use.	Invent a Game! This lesson is a transition lesson from coding using a visual programming language to a general-purpose programming language. It provides useful guidance around functions. This game uses the robotic device Sphero and a Sphero App. Game design This sequence of lessons integrates game design using scratch and a Makey Makey programming board. (Note this partially meets the programming achievement standard; to fully meet the standard would need to program using general-purpose programming language.)	Designing an algorithm This resource provides a simple explanation and example of Pseudocode. Flow chart software Gliffy Storyboard generator Use this free software to generate storyboards for game or app design. Lesson 2: The Need for Algorithms Students are presented with a "Human Machine Language" that includes 5-commands and then must figure out how to use these primitive commands to “program” the same algorithm. Functions A good resource to explain functions needed Lesson 8: Creating Functions with Parameters Students learn that writing functions with parameters can generalize solutions to problems even further. *Mockflow* Wireframe user interface design tool. Rapid Prototyping Studio A range of ideas for prototyping	***Learning to code:*** Learn to Code 3 Help students expand the coding skills to start thinking more like an app developer. The guide supports teachers to guide their students to code in Swift Playgrounds through unplugged activities and practice with the Swift Playgrounds app. Grok learning resources Grok provides online interactive programming courses for individuals or classrooms. Codecademy This site provides tutorials on web design tools. *Coding Ground* *Various text based programming languages with online development environments for students to explore and compare with block based languages.* Coding bat This site provides a range of exercises to practice coding and to build coding confidence in Java and Python. Swift Playgrounds Swift Playgrounds is a free iPad app from Apple that makes learning and experimenting with code interactive. ***Lesson ideas:*** DT Challenge 7/8 Python – Chatbot Write programs to solve problems with code and create word games.	*Evaluating the Product* Template for product evaluation.

			Tools to create mobile apps: *AppLab* *Students can create apps and switch between block (visual) and text based programming.* *MIT App Inventor* *Students can create apps and test these on mobile devices.*	
Assessment	**Suggested approaches may include:** One constraint on the solution. List of two or three functional requirements of the solution. **Achievement standard** **Define and decompose problems in terms of functional requirements and constraints.**	**Suggested approaches may include:** Two design ideas One example of each of branching and iteration in the algorithm (diagrammatic or structured English) **Achievement standard** **Design user experiences and algorithms incorporating branching and iterations**, and test, modify and implement digital solutions.	**Suggested approaches may include:** , A mini report (table/verbal/digital) for **two** tests outlining: • What is being tested in the solution • What are the expected results • What were the actual results • What changes were made, if needed **Achievement standard** Design user experiences and algorithms incorporating branching and iterations, and **test, modify and implement digital solutions**	Approach with a one-line prompt **Suggested approaches may include:** Demonstration of solution to a small group of students who rate key features on a scale. Demonstration of solution to a small group of students who each identify **one** feature that they thought was innovative or interesting. In pairs each student uses each other's solution and answers three evaluation questions. Each student explains how their solution met **one** functional requirement and **one** constraint. **Achievement standard** **Evaluate** information systems and their solutions in terms of meeting needs, innovation and sustainability.