# Create a language learning program

Please refer to the online lesson plan on the DT Hub to access all website links and additional resources.

**Year Level:** Years 3–4: Programming a digital solution with choices

Create a computer program to learn a traditional Aboriginal or Torres Strait Islander language.

# Suggested steps

1. Discuss and list ways you might learn a new language.
2. View the video Learn some Warrgamay words.
3. Discuss this approach to learning some words and phrases from a traditional Aboriginal language. From viewing the video, what words do the students now know? What does the video tell us about the animals and lands of the Warrgamay people?
4. Compare the video with this quiz, created in Scratch: Warrgamay animals.
5. Discuss how the quiz works and what programming blocks your students expect would have been used.
6. To check their predictions, select ‘See inside’.

**Note:** This program uses the blocks ‘broadcast message’ and ‘when I received a message’ to control the interaction. An alternative method could use ‘If/then’ blocks to control interaction.

After you have modelled the above, students are to **attempt** the task either individually or in small groups. They will need to choose a traditional Aboriginal or Torres Strait Islander language that can be learnt by using a computer program created using Scratch (or a similar programming language, eg Snap, Tynker or Hopscotch).

**Discuss** the approach to learning the language, eg common phrases and greetings, animal names, traditional Aboriginal objects and tools.

**Suggest** students create their quiz program, focusing on one question first to ensure it is working as expected (debugging). Once that program is tested and refined, then this can be applied to their remaining quiz questions. When working in groups of three, employ a strategy that allows all students to get equal opportunity to program.

1. **Considerations before setting the task:**

This task requires an abstract level of thinking where students need to do two types of coding.

* Language coding - the student needs to understand how words represent objects and that different words can represent the same object. For example, Eel and Wagal are two words that represent a long and thin, snake-like fish. Eel is the English word for this fish and Wagal is the Warrgamay word for this fish. You can extend this by considering that Wagal refers to the particular Eel fish seen in Warrgamay Country, whereas the English word Eel can refer to a variety of Eels.
* Digital Coding - where physical, real-world objects are represented as pictures, symbols, and binary code in a digital space. Students will need to be comfortable in using digital tools to represent real-world decision-making. For example, coding a language learning game in Scratch is a digital representation of the questions, actions, and prompting that would happen in a real-life quiz game. You may want to first illustrate the decision making as a whole-class and/or have students write down the steps to a learning game on paper before they attempt in a digital environment (see Level 1 below).

Students will also need to consider design elements of clear and effective presentations. You will need to ensure that your students are comfortable with this abstract thinking. Further, you should also regularly assess students' understanding of these deeper concepts as they may be able to complete the tasks without understanding the underlying thinking processes.

1. **Instructions (with Differentiation)**

The Digital Technologies curriculum differs from the old ICT curriculum in that there is an emphasis on students' thinking processes. Therefore this task has been divided into three levels where, first, students need to demonstrate they understand the logic and decision-making used to make a quiz game. Then they modify the example quiz to demonstrate understanding of Scratch. The actual creation of a new quiz is saved for last in Level Three. This allows you to differentiate the task depending on students' understanding of both computational thinking and Scratch (or similar programs). Proficient students could start on Level Two or even Level Three whereas students that find these tasks too challenging could start at Level One.

### Level One:

* Have students first create a flowchart of how a quiz game or multiple choice question gets asked and answered. If students are unclear what a flowchart is, then have them simply write down a fictious conversation between a quiz show host and a player. Alternatively, students could simply replicate and copy down the format of the first question and answer in the sample quiz shown above. They could also format it as a Comic Book panel if they prefer.
* Students should see that if the player guesses correctly, that the game does something different than if the player guesses wrong.

Visual ImpairmentVisual Impairment: If a student is vision impaired, then the student can explain the steps and then roll-play a game show host and/or player. The student could assign sounds to two parts of the game: 1. two different sounds to occur whether the player chose the correct or the incorrect answer, and 2. a different sound for each choice of a multiple choice or true/false question.

### Level Two:

* Have students then play the sample Scratch game and compare it to their flowchart (if created in Level 1). Students should be able to describe similarities (and differences, if appropriate).
* Then have students click "See Inside" and step through the Scratch code in the example. There are several features of Scratch that students must know in order to understand the sample:
  + The program starts at the Green Flag.
  + Backdrop is another term for Background image.
  + Say produces a speech bubble and displays text for the player.
  + Broadcast Message is like 'GOTO' in that it tells other code (with the same name) to start. In this example, you follow the code:
    - broadcast message1
    - When I receive message1
    - broadcast message2
    - ...and so on
  + Costume is a picture/image on top of the background and one or more Costumes make up a sprite.
  + Each sprite (picture/image/object) may have code attached to it -- click on a sprite to see the code.
  + Then, have the student modify parts of this code to make it their own. They should start with modifying text and testing it so they can see how their changes effect the quiz program.



* + Now, have the student add a new animal quiz question to the Scratch code.

They can do this by clicking on the Bream sprite, clicking the Costumes tab, and then drawing or uploading a picture.

* + - After they have done this, they will need to create two new messages (one for the new question and one to trigger on the correct answer).
    - Then they will need to add three new sprites for three new buttons.
    - Have students use and copy the existing code to make their fourth question.

Too challengingToo challenging: If the student is finding Scratch is too challenging, they can alternatively try their making their own quiz game in PowerPoint. It may be that some students require an example to follow and change the text as a starting point. Downloadable 'quiz game' examples are available from the Internet.

### Level Three:

* Students make their own language quiz. They need to choose a traditional Aboriginal or Torres Strait Islander language and which coding software they are to use (if you are offering them other choices besides Scratch). It is recommended for all but expert coders to create their game in Scratch so that they can refer back to the example Scratch game for help.

Too challengingToo challenging: Again, if a student is finding Scratch too challenging or digital coding, they could use PowerPoint or create a game using index cards.

1. Extension Activities

* **Sounds** - If you have access to a speaker of a traditional Aboriginal or Torres-Strait Islander language or you are able to find audio clips on the Internet, students could add recorded audio bites to enhance the program, learning and ensure correct pronunciation showing respect for the language. Students could also add sound effects for when the correct and incorrect answer is chosen.
* **Additional coding software** - If students finish early, suggest that they try making the same or a similar game in a different software program like Snap, Tynker or Hopscotch.
* **Drawing** - encourage students to draw (either by hand or using digital software such as MS Paint or Photoshop) their own backgrounds, sprites and costumes.
* **Game Timer** - students could add difficulty levels to their game by programming in a timer requiring an answer to be chosen in a set number of seconds.
* **Help Screen** - a help page and a "?" Help Icon could be added to give instructions to the player.

# Discussion

* What instructions did you provide? How easy are these to understand?
* Decisions: At what stages does the user have control and make choices? What programming did you use to enable this?
* Compare your algorithm to your final programming.

# Why is this relevant?

Decisions are important within computational thinking. They allow actions to be changed, based on the input of data. This input could be:

* user-input, for example selecting an onscreen value or button, typing in an answer
* sensed from the immediate environment; for example, collected via a sensor on a robotic device that senses an obstacle and is programmed to avoid it.

Algorithms are the step-by-step procedures required for solving a problem. Algorithms may be described either diagrammatically or in structured English. Flowcharts are often a good way of visualising algorithms and can be an effective way to teach the concept of ‘branching’. Branching involves making a decision between one of two or more actions, depending on sets of conditions and the data being inputted.

This activity can be used to strengthen students' understanding of computer programming as a series of instructions that can change depending on different user inputs or conditions. The focus is on how computers follow instructional pathways, and these can be described using flowcharts or visual programming languages.

# Assessment

Take note of students’ responses to their self-assessment. Ask students to self-assess using the following questions:

* Explain reasons for any changes to the algorithm before the final program was completed.
* Did debugging and testing lead to any improvements? In what way/s?
* How did you work together as a group? Now imagine that you had to complete this task on your own. How might the outcome be different?

# Australian Curriculum alignment

Technologies / Digital Technologies / Years 3 and 4 / Digital Technologies Processes and Production Skills:

* Follow and describe algorithms involving sequencing, comparison operators (branching) and iteration (AC9TDI4P02)
* Implement simple algorithms as visual programs involving control structures and input (AC9TDI4P04)

Humanities and Social Sciences / F–6/7 HASS / Year 4 / Knowledge and Understanding / Geography:

* Sustainable use and management of renewable and non-renewable resources, including the custodial responsibility First Nations Australians have for Country/Place (AC9HS4K06 )

Cross-curriculum priorities / Aboriginal and Torres Strait Islander Histories and Cultures / Key ideas / Culture OI.4: Aboriginal and Torres Strait Islander societies have many Language Groups.