Visual to text coding LESSON 2:

**Calculator**

**Go to lesson series** |  **Go to next lesson**

This is the second in a series of lessons to transition from visual coding to text-based coding with a general-purpose programming language.

Included videos can be used by the teacher and/or students to see how to code each of the simple programs step-by-step in all three languages: Scratch, Python and JavaScript.

This lesson may take two to three 45-minute periods. It introduces how to make decisions (**branching**) and identify **data types**.

## Cross-curricular approach

This lesson can be used by Mathematics teachers keen to include programming and logical thinking as part of their course. However, it is suggested that students complete the previous lesson to familiarise themselves with relevant skills on which this lesson builds.

# Curriculum links

Links with Digital Technologies Curriculum Area

| **Strand** | **Year** | **Content Description** |
| --- | --- | --- |
| Processes and Production Skills | 5–6 | Design algorithms involving multiple alternatives (branching) and iteration [(AC9TDI6P02)](https://v9.australiancurriculum.edu.au/f-10-curriculum.html/learning-areas/digital-technologies/year-5_year-6/content-description?subject-identifier=TECTDIY56&content-description-code=AC9TDI6P02&detailed-content-descriptions=0&hide-ccp=0&hide-gc=0&side-by-side=1&strands-start-index=0&subjects-start-index=0&view=quick). |
| 7–8 | Design algorithms involving nested control structures and represent them using flowcharts and pseudocode [(AC9TDI8P05)](https://v9.australiancurriculum.edu.au/f-10-curriculum.html/learning-areas/digital-technologies/year-7_year-8/content-description?subject-identifier=TECTDIY78&content-description-code=AC9TDI8P05&detailed-content-descriptions=0&hide-ccp=0&hide-gc=0&side-by-side=1&strands-start-index=0&subjects-start-index=0&view=quick).Trace algorithms to predict output for a given input and to identify errors [(AC9TDI8P06)](https://v9.australiancurriculum.edu.au/f-10-curriculum.html/learning-areas/digital-technologies/year-7_year-8/content-description?subject-identifier=TECTDIY78&content-description-code=AC9TDI8P06&detailed-content-descriptions=0&hide-ccp=0&hide-gc=0&side-by-side=1&strands-start-index=0&subjects-start-index=0&view=quick). |

# Assessment

Students undertake a self-reflection of the programming task. The teacher can use the completed self-assessments to assist in summative assessment.

* Download the self-assessment sheet in Word or PDF format.

In assessing code in languages like Python or JavaScript, consider a rubric that includes important skills for general-purpose programming.

* Download a sample rubric in Word or PDF format.

# Learning hook



 sugree/flickr, [Creative Commons BY-SA 2.0](https://creativecommons.org/licenses/by-sa/2.0/)

Use the context of ordering food for lunch to explain (in a practical way) the use of branching.

1. You and your friends are having lunch at a local food court.
* Small plate: $8.95
* Large plate: $10.95

You know you have enough money for at least the small plate. How do you decide what to order? Write down some **conditions**. These are questions that result in a **yes** or **no** answer, for example:

* Am I very hungry?
* Do I have enough money for the plate I want?
1. After ordering your food, you discover that drinks cost $2.95. Write down some **conditions** for this second decision.
* Am I thirsty?
* Do I have enough money for the plate and a drink?
1. You’ve just made two branching decisions. Here’s pseudocode for an example:

BEGIN

If **I am very hungry** and **myMoney >= 10.95** then

 Say “I’ll have the large plate.”

 If **I am thirsty** and **myMoney >= (10.95 + 2.95)** then

 Say “And a drink too, please.”

 End If

Else

 Say “I’ll have the small plate.”

 If **I am thirsty** and **myMoney >= (8.95 + 2.95)** then

 Say “And a drink too, please.”

 End If

End If

END

# Learning map and outcomes

In this lesson, students will:

1. access an online programming environment for visual code (**Scratch**) and for general-purpose programming (**Python** or **JavaScript**)
2. learn how to code decisions with an example about movie ratings
3. plan and code a calculator program where the user enters an operation (add, subtract, multiply or divide) as well as two numbers to operate on, then the computer calculates and outputs the result.

# Learning input

1. As a class, or in teams, design the calculator program as a flowchart. Effective designs may vary, for example, the program might ask for the operation first, in the middle, or last.

Here is one solution:

Image: Flowchart – Calculator - Simplified.xml



1. In the following revised flowchart, we break the big decision into four **yes** or **no** decisions. This is closer to how we will code the program.

Image: Flowchart – Calculator.xml Simpl



1. Once the flowchart is complete, write the program in pseudocode (structured English). Remember, each of your conditions must come to a **yes** (true)or a **no** (false).

BEGIN

Display “Choose your operation - a for add, s for subtract, m for multiply, d for divide:”

**operation** ← input from user

Display “Enter the first number:”

**firstNumber** ← input from user

Display “Enter the second number:”

**secondNumber** ← input from user

If **operation** = “a” then

 Display “The result is “, **firstNumber** + **secondNumber**

Else If **operation** = “s” then

 Display “The result is “, **firstNumber -** **secondNumber**

Else If **operation** = “m” then

 Display “The result is “, **firstNumber** \* **secondNumber**

Else If **operation** = “d” then

 Display “The result is “, **firstNumber** / **secondNumber**

End if

END

SIDEBAR – Starting pseudocode

Program designers use pseudocode (also called structured English) before coding an algorithm in a real, specific programming language.

The purpose of pseudocode is to clearly understand and communicate an algorithm, regardless of the final language used. Because it translates more readily into real code, it is often more popular than a flowchart.

Pseudocode has few strict rules, but here are some helpful hints to get started:

* Begin your algorithm with BEGIN. End it with END.
* Use a left-pointing arrow (←) to indicate assigning a value to a variable.

eg: **income** ← 5 × 12 The variable **income** will now contain the value 60.

At Years 7 and 8, the Australian Digital Technologies curriculum specifies: ‘Design algorithms represented **diagrammatically** and **in English** …’ ([ACTDIP029](https://www.australiancurriculum.edu.au/Search/?q=ACTDIP029)). This is further specified as ‘**structured English**’ in Years 9 and 10 ([ACTDIP040](https://www.australiancurriculum.edu.au/Search/?q=ACTDIP040)).

# Learning construction

**Step 1: Setup**

For more on setting up and choosing a language, see Lesson 1.

View the videos on [Setting Up](https://youtu.be/LSRs_T6Nga8) and [Gotchas](https://youtu.be/3pQ6USS6M_s).

**Step 2: Introducing branching**

Decision making is called **branching** or conditionals.

In general-purpose programming, we do this with IF, THEN and ELSE code structures.

View this video on [Control flow](https://youtu.be/Vs-wWooWjEI).

Have students practise branching. Interpret the following pseudocode and code it in Scratch **and** either Python or Javascript. Students should predict the output of the program (what it will display) before doing their code.

BEGIN

**a** ← 20

Display **a**

**b** ← 14

Display **b**

**c** ← **a** + **b**

If c > 34 then

 Display “You win $1 000 000!”

Else

 Display “Better luck next time.”

End If

If (a + c) = 54 then

 Display “YOLO”

End If

END

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/304045767)*,* [*Python*](https://repl.it/%40digitechhub/Introducing-Branching-solution)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/grp6qs9k)

SIDEBAR – Strategies for coding and debugging

Students may work at vastly different paces when doing general-purpose programming, and some may seem quicker at spotting problems than others.

Try these strategies with your students:

* **Cheat sheets** bring together the most basic Python commands or JavaScript commands. See Resources at the bottom of this page.
* **Pedantic computer:** Think of the computer as a **really** pedantic person. It usually won’t cooperate if important punctuation (syntax) is missing, if something is spelled differently in two places, or even with the wrong upper or lower case.
* [**Pair programming**](https://www.youtube.com/watch?v=vgkahOzFH2Q) **or ‘Ask three before me’:** Often it just takes a fresh pair of eyes to spot errors or bugs in a program. Before asking the teacher the student checks with other students.
* **Trace errors:** The computer always runs the program in order, line-by-line, according to the code. Study each line one-by-one and ask, ‘What does this line do?’ Consider writing down the values of variables as you go.
* **Tinker:** Encourage students to tinker with the code. Ask, ‘What would happen if I changed this?’ Alter a part of the code or add a new feature, predict what might happen, and run it to see. You can always put it back again if something goes wrong.

This [International Space Station video](https://youtu.be/JLmTBoGZlbM) is a real-world example of how branching is used in a more advanced program. If there is connection to the internet ( if (response.status\_code == 220) then proceed, otherwise print(' no connection') and the program stops.

 **Step 3: Calculator**

It’s time to code our calculator.

The video on [Calculators](https://youtu.be/uTdB7vbFZZU) covers the entire process of building the code.

*Final code:* [*Scratch*](https://scratch.mit.edu/projects/283928268/fullscreen/)*,* [*Python*](https://repl.it/%40JasonVearing/Calculator)*,* [*JavaScript*](https://jsfiddle.net/3jybt9f7/1/)

 **Step 4: Tinker task**

Modify your calculator program to allow a fifth operation: power. If the user enters ‘p’ for the operation, the first number is raised to the power of the second number.

eg: if first number is 5 and second number is 3, result is 53 = 125

You’ll need to research how to perform this operation in Python or JavaScript. See the Resources at the bottom for cheat sheets.

*Solution code:* [*Python*](https://repl.it/%40digitechhub/Calculator-Tinker-task-solution)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/60rtLxs5)

Note: there is no power block (xy) in Scratch.

# Challenge

These challenges use the skills covered so far. By writing or modifying their own programs, students have an opportunity to demonstrate Application and Creation.

1. Code a new program in Python or JavaScript to ask for the user’s name, then give a custom reply depending on the name entered. For example:

 Hi, what’s your name? **Dani**

 You’re the best, Dani!

 Hi, what’s your name? **Wei**

 Welcome, Wei!

* 1. Write out the algorithm in pseudocode first.
	2. If it helps, code the program in Scratch before going on to Python or JavaScript.

Challenge early finishers to add a second question about the number of Wi-Fi devices in their home, then give a smart comment based on the value.

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/304059812/)*,* [*Python*](https://repl.it/%40digitechhub/Lesson-2-Challenge-1)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/9b036jgt/)

1. Design (flowchart or pseudocode) then code a program to sell you an ice cream.
* First, ask which type of cone (plain = $1, cup = $1.50, waffle = $2).
* Next, ask which flavour (always $2.50, no matter what flavour).
* Next, ask which topping (nuts = 50c, sprinkles = 50c, chocolate sauce = $1).
* Finally, display the total cost.

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/304060522/)*,* [*Python*](https://repl.it/%40digitechhub/Lesson-2-Challenge-2)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/5tu30wgo/)

1. Code a maths quiz, for example:

 What is 2 + 5? **7**

 That’s right!

 What is 10 – 5? **5**

 That’s right!

 What is 3 \* 5? **25**

 Incorrect.

 Your total score is 2 out of 3.

Here’s a way to break down this challenge into smaller chunks:

* 1. Ask one question, then check if the user got it right. **Test your program.**
	2. Add two more questions to follow the first one. **Test your program*.***
	3. Include a **score** variable to keep track of how many questions the user got right.
	4. (Extension) Can you make your questions unique every time? Check out how to generate random numbers in Lesson 3.

# Resources

* Online environments for coding in each language:
	+ [**Scratch**](https://scratch.mit.edu/)
	+ [repl.it](https://repl.it/), an online environment suited to **Python**
	+ [JSFiddle](https://jsfiddle.net/), an online environment suited to **JavaScript**
* Cheat sheets listing basic commands for coding:
	+ Python [Cheatsheet](https://groklearning-cdn.com/resources/cheatsheet-python-1.pdf) (from Grok Learning)
	+ JavaScript [CheatSheet](https://htmlcheatsheet.com/js/) (Tip: Press the little blue tabs to move **Variables**, **Basics**, **Strings** and **Data Types** to the top.)