Visual to text coding LESSON 4: **Scissors, Paper, Rock**

**Go to Lesson Series** |  **Go to next lesson**

This is the fourth 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 a beginner 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 the combining of logical operators **and** and **or** for more complex decisions.

# Curriculum links

Links with Digital Technologies Curriculum Area

| **Strand** | **Year** | **Content Description** |
| --- | --- | --- |
| Processes and Production Skills | Year 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). |
| Year 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 can 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 brings in important skills for General Purpose Programming.

* Download a sample rubric in Word or PDF format.

# Learning hook



geschenkhamster.de/Wikimedia Commons, CC BY-SA 3.0

*For this group activity, students should work in small teams. Each team needs a copy of this handout, which contains logical operators, conditions and outcomes to be cut out with scissors.*

Your class is having a present giving celebration. The rules are quite complicated.

Read these carefully as a class to make sure you understand how the game works:

 A certain number of presents is placed together on a table, and everyone’s name is placed in a hat. Some names are written on green cards, and other names on red cards.

 If your name is picked from the hat, your action will depend on what colour card your name is written on.

If it’s a green card, you must take and unwrap a present from the table. However, if no presents remain on the table, you must take the other action as if you have a red card.

 If it’s a red card, you must choose a present to steal from the person who went before you. If that present has already been stolen twice, it is out of bounds and you get nothing! Otherwise, you successfully steal the present. (The victim must then obtain a replacement present by acting according to their own card colour.)

In your team construct three long, logical sentences to determine your actions. Each sentence leads to one of the three outcomes.

## Solutions

*(Solutions may vary, but they must be logically correct).*







SIDEBAR – Order for logical operators

Like a Maths sentence, logical operators have ‘BODMAS’ rules of their own.

* **AND** comes *before* **OR** (just like **×** comes *before* **+**)
* **NOT** comes before them both

As in Maths, brackets are often necessary to communicate the correct order.

eg.

**IF**  **(**I’m walking **OR**  I’m taking the train**)** **AND**  it is raining **THEN** bring umbrella.

Correct. I bring an umbrella if it is raining and if one of the other two conditions is true.

eg.

**IF** I’m walking **OR**  I’m taking the train **AND** it is raining **THEN** bring umbrella.

Without the brackets, the AND happens first. This means I bring an umbrella if it’s raining and I’m taking the train, but I also bring an umbrella every time I’m walking!

# 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. explore the design of a game the brings together user input, random numbers and complex winning conditions,
3. plan and code the Scissors, Paper, Rock game, where the user plays against the computer.

# Learning input

Begin by watching the overview video on [Scissors, Paper, Rock](https://youtu.be/_Ajv0D0LzUs).

As a class, write or type out the incomplete pseudocode below, filling in the gaps.

*(Note this program is slightly simplified.)*

**1** BEGIN

**2 randomNumber** ← random choice of 0, 1 or 2

**3** If **randomNumber** = 0 Then

**4 computerPick** ← ‘scissors’

**5** Else If **randomNumber** = 1 Then

**6 computerPick** ← \_\_\_\_\_\_

**7** Else If **randomNumber** = \_ Then

**8 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**9** End If

**10**

**11** Display ‘Scissors, paper or rock?’

**12 userGuess** ← input from user

**13**

**14** If **\_\_\_\_\_\_\_\_\_\_** = **\_\_\_\_\_\_\_\_\_\_** Then

**15** Display ‘Tie.’

**16** Else

**17** If (**computerPick** = ‘scissors’ And **userGuess** = ‘paper’) Or

**18** (**computerPick** = ‘paper’ And **userGuess** = \_\_\_\_\_\_\_\_) Or

**19** (**computerPick** = \_\_\_\_\_\_\_\_ And **userGuess** = ‘scissors’) Then

**20** Display ‘You lose.’

**21** Else

**22** Display \_\_\_\_\_\_\_\_\_\_\_

**23** End If

**24** End If

**25** END

This program has three sections (Lines 2-9, Lines 11-12 and Lines 14-24). Discuss the following questions to develop an understanding of the structure.

Q. What is the purpose of Lines 2 to 9?

A. This is where the computer’s pick (scissors, paper or rock) is chosen using a random number.

Q. What is the purpose of Lines 11 and 12?

A. This is where the user makes their pick.

Q. What is the purpose of Lines 14 to 24?

A. This is where the program determines who was the winner, or if there was a tie.

# 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: CODING THE GAME**

[This video](https://youtu.be/2tzGAYVHyrQ) demonstrates coding the solution in Scratch, Python and JavaScript. Try it yourself before checking the solution code below.

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/304079513/)*,* [*Python*](https://repl.it/%40JasonVearing/Scissors-paper-or-rock)*,* [*JavaScript*](https://jsfiddle.net/jasonvee/0kweua2d/)

**STEP 3: UNDERSTANDING VALIDATION**

The game rejects invalid user input. If the user types anything other than ‘rock’, ‘paper’ or ‘scissors’, they must try again.

This is called **electronic validation**, and is an important part of the user experience. Ideally, a solution should reject invalid data, and provide useful feedback or error messages to the user.

SIDEBAR – More on electronic validation

Like automatic spell checking, we are so used to electronic validation today that we often don’t realise how often it helps us.

Every time you fill out an online form, there are automatic checks to reduce the likelihood that you mistype something. Many online forms even go as far as to check your entered street address against a database of existing streets and suburbs.

Electronic validation is distinct from **manual validation**, where a human checks data for errors or missing items.

In coding, electronic validation can quickly become a burden. It is important that students focus on the main functional requirement of their solution first, before implementing a large amount of validation in their code.

Outside of coding, you can practice electronic validation of data with spreadsheet software like Microsoft *Excel*. Use Data Validation to circle cells with invalid data, according to rules that you choose. The same rules can be used to reject invalid data before it is entered in a cell, and even to force the user to choose from a drop-down of valid options.

CS Unplugged includes a [unit plan](https://csunplugged.org/en/topics/error-detection-and-correction/unit-plan/) with unplugged activities and programming challenges to explore how barcodes and credit card numbers incorporate systems for error detection.

Use the incomplete table below to discuss how different programs and user interfaces do electronic validation. You may be able to think of some more situations.

|  |  |  |
| --- | --- | --- |
| App or program | Situation | How might the app validate my input? |
| Scissors, Paper, Rock game | I might enter the wrong word. | IF word is not ‘rock’, ‘paper’ or ‘scissors’ THEN make me type again. |
| Farming game on my phone | Buying a new animal, but I might not have enough gold. | IF not enough gold THEN ‘Buy’ button is greyed out. |
| Online address form | I might forget a digit in my phone number. |  |
| I might misspell my state. |  |
| Mining and forging simulator | I might try to combine two items that can’t be combined. |  |
| Text adventure | I might type GO NORTH, but there is no path North from here.  |  |

**STEP 4: TINKER TASK**



Jim Champion/Wikimedia Commons, [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/deed.en)

Some people think that ‘rock’ should beat everything! Make a new version of the program where ‘rock’ always beats paper and scissors. (Keep your original game code somewhere else!)

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/304068665/)*,* [*Python*](https://repl.it/%40digitechhub/Scissors-paper-or-rock-Tinker-Task)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/qnaryp9k/)

# 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. Edit the original game to work “best out of three” – that is, the winner is the one who won the most games after 3 rounds.

You’ll need to add an additional variable to keep track of the number of times the user has won, then check it at the end of three rounds. You might also need a variable to keep track of the number of the times the computer has won.

*Note: You do* ***not*** *need to use a loop, since loops will be officially covered later in this course. You may instead retype or copy and paste your existing code.*

Solution code: [Scratch](https://scratch.mit.edu/projects/304079176/), [Python](https://repl.it/%40digitechhub/Lesson-4-Challenge-1), [JavaScript](https://jsfiddle.net/nathanesa/sup0qtxy/)

1. (OPTIONAL) More advanced versions of Rock-Paper-Scissors have been invented, including [RPS-5](http://www.samkass.com/theories/RPSSL.html) (Rock-Paper-Scissors-Lizard-Spock), [RPS-7](http://www.umop.com/rps7.htm), [RPS-15](http://www.umop.com/rps15.htm) and even [RPS-101](http://www.umop.com/rps101.htm)!

Watch this video ['Battle on the Breakwall'](https://vimeo.com/237852160).

Can you code a program for RPS-5 (Rock-Paper-Scissors-Lizard-Spock)?



Image credit: Rock Paper Scissors Lizard Spock by Sam Kass and Karen Bryla (CC-SA)

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/304082271/)*,* [*Python*](https://repl.it/%40digitechhub/Rock-paper-scissors-lizard-spock)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/7wLe5hsp/)

1. (OPTIONAL – **Python only**) With the addition of a little extra code, you can now make a program that uses live, real-world data. Have a go at the program below.

# 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**
* CheatSheets 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.)