# Visual to text coding Lesson 3: **Heads or tails**

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

This is the third 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 how to generate and use **random** numbers.

# 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) . |
| 7–8 | Design algorithms involving nested control structures and represent them using flowcharts and pseudocode (AC9TDI8P05) .Trace algorithms to predict output for a given input and to identify errors (AC9TDI8P06) . |

# 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 brings in important skills for General Purpose Programming.

* Download a sample rubric in Word or PDF format.

# Learning hook



 Francois Philipp/flickr, Creative Commons BY 2.0

In groups, think of some of your favourite tabletop or video games, and complete a table like the one below by answering these questions:

1. Does the game have any **random** element? Something that is left to chance?
2. What mechanism is used to generate the randomness?

The first few rows are completed as examples.

|  |  |  |
| --- | --- | --- |
| Game | Random element(s) | Mechanism to generate randomness |
| Monopoly | Which square the player lands on | Dice roll |
| Chance cards | Shuffled cards |
| Chess | No random element (except for starting player) |
| Matching puzzle phone game | Order of arrival of pieces to match | Game code chooses piece types |
| First person shooter game | Spawn positions of players | Game code chooses locations |
| Locations and types of upgrades | Game code chooses locations and types |
| AI of enemies | Game code chooses behaviours with a mix of strategy and randomness |
|  |  |  |
|  |  |  |
|  |  |  |

You may want to take this one step further in a class discussion:

**Q.** A computer often makes thousands of ‘random’ choices when you play an electronic game. How does it choose a random puzzle piece, for example?

**A.** When a program seems to choose a random puzzle piece, it’s because each piece type has been assigned a number. This is like rolling a die to select which piece type will appear next.

**Q.** But there’s no dice inside a computer. Can it really choose random numbers?

**A.** For a computer to come up with a truly random number, it must gather complex data from the world outside, such as fan noise. Computers can also make ‘pseudorandom’ numbers, which can appear to be random but are generated in sequence by an algorithm and a seed value. How random numbers are generated is quite a controversial topic, because it applies to encryption for security.

# 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 simple game combining user input and random number generation
3. plan and code a Heads or tails game, where the user tries to guess what the computer will throw.

# Learning input

Begin by watching the overview video on Heads or Tails.

As a class, or in teams, discuss what the Heads or tails program will do, then design the program as a flowchart. Effective designs may vary, for example, the program might toss the coin first, then ask the user for their guess.

Here is one solution:

Image: Flowchart for Heads or tails game



Once the flowchart is complete, write the program in pseudocode (structured English).

BEGIN

Display “Pick heads or tails:”

**guess** ← input from user

**computerPick** ← random choice between ‘heads’ or ‘tails’

If **guess** = **computerPick** then

 Display “You guessed it.”

Else

 Display “Better luck next time.”

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.

# Learning construction

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

**Step 1: Coding the game**

The video demonstrates coding the solution in Scratch, Python and JavaScript. Try it yourself before checking the solution code below.

*Solution code: Scratch, Python, JavaScript*

**Step 2: Dice roll**

Now that you can toss a coin, this video below challenges you to code a 6-sided dice roll.

Try it yourself before finishing the video or checking the solution code below. You may want to design your program with a flowchart or pseudocode first.

*Solution code: Scratch, Python, JavaScript*

SIDEBAR – Strategies for coding and debugging

Students may work at vastly different paces when doing general-purpose programming, and some may seem to be 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 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.

**Step 3: Tinker task**

Modify your dice roll program to simulate a single 20-sided die.

Next, edit the program so that it starts by asking the user how many sides the die should have, then rolls that die.

*Solution code: Scratch, Python, JavaScript*

# 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. How about a program to help you pick who gets to go first when playing a board game? The program should randomly pick one name out of five names.

Note: You don’t need to ask the user to enter all the names. They can be built into the program (or ‘hard-coded’).

* 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 improve the program. Now it will ask the user to enter the five names at the start, store each one in a variable, then randomly choose one of them to display.

*Solution code: Scratch, Python, JavaScript*

1. Mad Libs is a word game where you are asked for some words and then the computer, or another person, creates a silly sentence or story by using your words to fill in blanks. For example:

 Enter your name: **Bill**

Enter your favourite place: **Paris**

 Enter your favourite sport: **soccer**

 Enter an exclamation: **Ouch!**

 Enter an adjective: **hairy**

 Bill was playing soccer in Paris when a hairy bicycle crashed into him. ‘Ouch!’ said Bill.

By asking for more than one place, or more than one sport, you can make the story more random by choosing just one response.

*Solution code: Scratch, Python, JavaScript*

1. (Optional) Armed with the skills to use **branching** (from Lesson 2), you can write a choose your own adventure story. By adding **variables**, you’re already making a smarter story than is possible using slide presentation software. Maybe you can even add a **random** element to your game.

# Resources

* Setting up online environments
* Online environments for coding in each language
	+ Scratch
	+ repl.it : an online environment suited to Python
	+ JSFiddle : an online environment suited to JavaScript
* Cheat sheets listing basic commands for coding:
	+ Python Cheatsheet (from Grok Learning)
	+ JavaScript CheatSheet (Tip: Press the little blue tabs to move Variables, Basics, Strings and Data Types to the top.)