# Visual to text coding LESSON 8: **Guess the Number**

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

This is the eighth 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 teacher and/or students as guides on how to code each of the simple programs step-by-step in all three of these general-purpose programming languages: Scratch, Python and JavaScript.

This lesson may take two to three 45-minute periods. It brings together skills from the previous lessons to design and develop a Higher Lower game, where the player tries to guess a secret number.

# Curriculum links (button)

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). |
| 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).  |

# Assessment (button)

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

The Rubik’s Cube was invented in 1974, and is still a popular puzzle today.

There are a number of solutions to the Rubik’s Cube, each involving a sequence of steps. The Lego robot in the video above solves the cube using one of these sequences, with a colour sensor to monitor the colours while working. However, the Rubik’s Cube can even be solved blindfolded! By repeating a sequence of steps enough times, the solution can be reached even without looking at the colours.

A sequence of steps to solve a problem is referred to as an **algorithm**.

Invite students to play this game in pairs:

1. Think of a number between 1 and 20. Don’t tell your partner.
2. Now, ask your partner to guess the number.
3. If they get it wrong, tell them to go ‘lower’ or ‘higher’ and try again. If they get it right, the game is over.

Ask:

* What’s the quickest way to win this game as the guesser? What number would you start with?
* What would be your next guess if you were told to go ‘higher’ or ‘lower’?

Students can write this game out as a simple algorithm (flowchart or pseudocode is not necessary).

**SAMPLE SOLUTION**

Guess 10 (halfway between 1 and 20).

If told higher, ignore the numbers 1 to 10, and guess halfway between 11 and 20.

If told lower, ignore the numbers 11 to 20, and guess halfway between 1 and 9.

Keep repeating this process, throwing out the irrelevant numbers and guessing halfway between the numbers that are left.

The solution is an algorithm called a **binary search**. For an example of a binary search view The famous phonebook video clip.

# 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. code a higher/lower game, where the player must guess a secret number between 1 and 20
3. tinker with the game to make it more challenging.

# Learning input

Begin by watching the video demonstrating the higher/lower game, which uses loops and variables.

Now, examine the Pseudocode below carefully. Then answer the questions below it:

BEGIN

 **answer** ← 0

 Repeat While **answer** != 12

 **answer** ← Input ‘How many months are in a year?’

 End Repeat

 Display ‘You got it!’

END

**QUESTIONS:**

1. What question does the program ask the user?

2. How many times will the question be asked?

3. What happens once the loop ends?

**ANSWERS**

1. ‘How many months are in a year?’

2. The question keeps getting asked until the user enters the right answer. It could go forever!

3. Once the loop ends, the program displays ‘You got it!’

Here is a more complex version of the program. Read it carefully, then answer the questions.

BEGIN

 **answer** ← 0

 **tries** ← 0

 Repeat While **answer** != 12 And **tries** < 3

 **answer** ← Input ‘How many months are in a year?’

 **tries** ← **tries** + 1

 End Repeat

 If **answer** = 12 Then

 Display ‘You got it!’

 Else

 Display ‘You ran out of tries.’

 End If

END

**QUESTIONS:**

1. What is the purpose of the new variable ‘tries’?

2. In this version, the loop may end even if the user doesn’t get the right answer. Why?

3. Why is the If-Then-Else structure needed at the end of the program?

**ANSWERS**

1. The variable ‘tries’ keeps the number of attempts the user has made to enter the right answer.

2. A second condition has been added to the ‘While’ loop. The loop will now stop after three attempts. It also still stops if the user enters the right answer.

3. Once the loop ends, we need to check once more if the user got the answer right. It’s possible the loop ended after three unsuccessful attempts.

SIDEBAR – TWO TYPES OF LOOPS

 There are two main types of loops: **while** and **for**.

|  |  |  |
| --- | --- | --- |
| A **while** loop is like an **if-then** structure, except the code inside repeats as long as the condition is met.Examples:* repeat while the user has not entered the correct password
* repeat while the enemy is still alive
* repeat while we have time left to win
* repeat while a number hasn’t reached a target (this one is normally done with a **for** loop)

The closest Scratch equivalent is the **repeat until** block, but its logic is opposite. It repeats *until* a condition is met, not *while* a condition is met. |  | A **for** loop is specialised for counting. The code inside repeats a certain number of times.The counter (also called the ‘index variable’) changes each time the loop runs. It may be displayed inside the loop, or it may never be displayed.Examples:* repeat a message 10 times
* produce a series of numbers from 1 to 10
* produce a countdown from 10 to 1
* access each item in an array

The closest Scratch equivalent is the **repeat** block, but it is less flexible and does not provide access to the counter. |
|  |  |  |

# Learning construction

For more on setting up and choosing a language, see Setting Up.

**Step 1: Solution development**

The video ‘Guess the number’ demonstrates coding the solution in Scratch, Python and JavaScript. Try it yourself before checking the solution code.

Solution code: Scratch, Python and JavaScript

SIDEBAR – RULES VS. CONVENTIONS

By now, you have probably noticed that some coding ‘rules’ seem to matter more than others. Here are some examples of rules versus conventions:

|  |  |
| --- | --- |
| **Rule**(Do this or you will likely cause a syntax error or a bug in your program!) | **Convention**(Do this for code readability, but some programmers may disagree.) |
| Use round brackets with display commands. For example:**Python:** print(‘Hello’) **JavaScript:** document.write(‘Hello’); | Use single quotes inside display commands. Double quotes also work. For example: **Python:** print(“Hello”)**JavaScript:** document.write(“Hello”); |
| In **Python**, indent code inside structures. For exampleif (10 > 5): print('Of course!') | In **JavaScript**, indenting is strongly recommended.Indented:if (10 > 5) { document.write(‘Of course!’);}Not indented:if (10 > 5) {document.write(‘Of course!’);} |
| In **JavaScript**, use var (or let) to declare variables. (There is a subtle difference between var and let, important once you make complex programs. You can learn more [here](https://youtu.be/cm9U7k2oLts).) | Use ‘camel case’ for variable names. For example: myName, orderTotal, noOfStarsSome Python coders prefer underscores:my\_name, order\_total, no\_of\_stars |
| Use == for comparison of two numbers, never a single = sign. For example:**Python:** if (rating == 5): print('5 stars!')**JavaScript:** if (rating == 5) { document.write(‘5 stars!’);} | Include spaces in Maths statements. Often, they’ll work without spaces. For example:**Python:** if (rating==5): print('5 stars!')**JavaScript:** if (rating==5) { document.write(‘5 stars!’);} |

As with spoken languages, some rules used to be universally accepted as strict, but now are sometimes ignored, such as using semicolons (;) at the end of code lines in JavaScript. Whether this will cause a syntax error may depend on the coding environment being used.

Finally, there are some conventions that are just habits. For example, programmers often use the name **i** for the index variable in loops, because ‘i’ is short for ‘index’. But you could use another letter like **n**, or a word like **index** for the variable name.

**Step 2: Tinker task**

First, change the range for the secret number from 1–20 to a higher number; for example 1–50.

Next, modify the game so that the player only gets a maximum of guesses; for example 5 – otherwise they lose. The higher/lower game already records the number of guesses the player makes.

Solution code: Scratch, Python and 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. Design and code a program to provide a cumulative total as prices are entered, until the user chooses to stop. See the sample output below:

Welcome! Enter your first price: **17**

The total so far is $17.

Enter another price, or enter STOP to finish: **5**

The total so far is $22.

Enter another price, or enter STOP to finish: **12**

The total so far is $34.

Enter another price, or enter STOP to finish: **STOP**

Thank you for using this program.

* 1. Prepare pseudocode first.

|  |
| --- |
| **SAMPLE SOLUTION**BEGIN **total** ← 0 Display ‘Welcome! Enter your first price: ’ **price** ← input from user Repeat While **price** is not equal to ‘STOP’ **total** ← **total** + **price** Display ‘The total so far is $’, **total**, ‘.’ Display ‘Enter another price, or enter STOP to finish: ’ **price** ← input from user End Repeat Display ‘Thank you for using this program.’END |

* 1. Code the program in Python or JavaScript.

Solution code: Scratch, Python and JavaScript

1. Code a vowel replacer program by following the video.

Solution code: Scratch, Python and JavaScript

1. (OPTIONAL) Ask your students to write a higher/lower game where the roles are reversed. The human player thinks of a secret number, and the computer must guess the answer.

Here are some prompt questions to help students develop their pseudocode:

* What input and output will be required? How will the user tell the computer to go higher or lower? (This is a useful program to design as a class, especially for a device like the BBC micro:bit with limited inputs available to the user.)
* How will the computer reach the correct number? See our discussion of **binary search** at the beginning of this lesson. Note, after your calculation you will need a way to round down to the nearest whole number. To do this, you can use the int() command in Python and the parseInt() command in JavaScript.
* What things will the computer need to remember as it works toward the correct number? These will need to become variables in the program.

Sample solution code: Scratch, Python and JavaScript

# 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**
* CheatSheets 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.