Visual to text coding LESSON 7: **Times Table**

# (Top summary)

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

This is the seventh 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 introduces iteration(also called **loops**).

# Curriculum links

Links with Digital Technologies Curriculum Area

| **Strand** | **Year** | **Content Description** |
| --- | --- | --- |
| Processes and Production Skills | Years 5-6 | Design, modify and follow simple algorithms involving sequences of steps, branching, and iteration (repetition) [ACTDIP019](https://www.australiancurriculum.edu.au/Search/?q=ACTDIP019). |
| Years 7-8 | Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors [ACTDIP029](https://www.australiancurriculum.edu.au/Search/?q=ACTDIP029)  Implement and modify programs with user interfaces involving branching, iteration and functions in general-purpose programming language. [ACTDIP030](https://www.australiancurriculum.edu.au/Search/?q=ACTDIP030). |

# 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](https://dth-stg-cms.esa.edu.au/docs/default-source/Lesson-Ideas/visual-to-text-coding/self-assessment---visual-to-text-coding---lesson-7.docx) or [PDF](https://dth-stg-cms.esa.edu.au/docs/default-source/Lesson-Ideas/visual-to-text-coding/self-assessment---visual-to-text-coding---lesson-784924a9809f96792a599ff0000f327dd.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](https://dth-stg-cms.esa.edu.au/docs/default-source/Lesson-Ideas/visual-to-text-coding/rubric-example---software-design-and-development-with-general-purpose-language6baf4a9809f96792a599ff0000f327dd.docx) or [PDF](https://dth-stg-cms.esa.edu.au/docs/default-source/Lesson-Ideas/visual-to-text-coding/rubric-example---software-design-and-development-with-general-purpose-language.pdf) format.

# Learning hook



Long Zheng/flickr, [CC BY-SA 2.0](https://creativecommons.org/licenses/by-sa/2.0/)

Let’s talk about strategies to win a game like noughts-and-crosses (tic-tac-toe), or to solve a puzzle like Sudoku. What is your strategy?

* Do you take risks and make guesses?
* Do you write down possibilities, trying to predict the future before making your move? If so, how many moves ahead can you predict?

In 1996, a supercomputer named [Deep Blue](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)) won a chess game against Garry Kasparov, the reigning chess champion at the time.

As a class, discuss the question: *Was Deep Blue smarter than Kasparov?*

Some prompts to help in the discussion:

* What sort of strategies might Deep Blue have used?
* Thinking *fast* may not be the same as thinking *smart*.
* With Noughts and Crosses, there’s a small number of ways the game can play out, enough for a human to know for certain the best move at any time. What makes chess different to noughts-and-crosses?
* Computers, robots and other machines are great at repetitive tasks that bore or exhaust humans, known as **iteration**.

[](https://www.youtube.com/watch?v=Km024eldY1A)Deep Blue used a **brute force** method to win chess. It used its powerful processor to test out millions of possible futures in the game, over and over, then to select the move most likely to result in a win. Many times, computer programs seem ‘smart’, but they just have access to more data than a human and can repeat a behaviour more quickly.

**THOUGHT EXTENSION 1:** Could a powerful enough computer predict every single possible outcome of a chess game, guaranteeing that it would know how to win at every stage?

**THOUGHT EXTENSION 2:** What [other approaches](https://www.popsci.com/artificial-intelligence-takes-chess-beyond-brute-force) are computer scientists taking to make computers smarter at chess?

# 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. practise writing loops (iteration) in two forms – a **while** loop and a **for** loop
3. code a times table generator that produces a times table for any number that is entered.

# Learning input

Begin by watching the video introducing [arrays](https://youtu.be/AmMZzzpODcc?list=PL0dMM9mg1u6NKQNT1odqe-5dI-lEovDxm).

Examine the pseudocode below carefully, then write the output of the programs; that is, exactly what they will display on screen:

|  |  |  |
| --- | --- | --- |
| BEGIN  For **n** from 1 to 5  Display ‘DEFCON ’, **n**  End For  END  **SOLUTION:**  DEFCON 1  DEFCON 2  DEFCON 3  DEFCON 4  DEFCON 5 |  | BEGIN  Repeat 4 times  Display ‘May it rain,’  End Repeat  Display ‘Whisper words of water,  may it rain.’  END  **SOLUTION:**  May it rain,  May it rain,  May it rain,  May it rain,  Whisper words of water, let it rain. |

Now implement the two programs in Python or JavaScript, using the **for** loop techniques learned in the video.

*Solution code:* [*Python*](https://repl.it/@digitechhub/Lesson-7-Learning-Input)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/ognxzh8t/)

SIDEBAR – Two types of loops

The video on loops introduces two 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 n** 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](https://www.digitaltechnologieshub.edu.au/teachers/lesson-ideas/visual-to-text-coding-setting-up/).

**STEP 1: SOLUTION DEVELOPMENT**

This video on [Times tables](https://youtu.be/HAA4DQ6fdzw?list=PL0dMM9mg1u6NKQNT1odqe-5dI-lEovDxm) demonstrates coding the solution in Python and JavaScript. Try it yourself before checking the solution code.

*Solution code:* [*Python*](https://repl.it/@JasonVearing/Times-tables)*,* [*JavaScript*](https://jsfiddle.net/jasonvee/ovdaxg7f/2/)

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. eg.  **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 example  if (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://hackernoon.com/heres-the-difference-between-let-and-var-in-javascript-83d8864b74c0).) | Use ‘camel case’ for variable names. For example  myName, orderTotal, noOfStars  Some 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, some conventions 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**

Modify your times table program to go all the way up to 20.

Next, add a second prompt to ask the user how high the times table should go. For example, if they enter 25, the times tables goes all the way up to 25.

Lastly, add a second table that shows the powers for the number. Here’s an example of the output:

5 to the power of 0 = 1

5 to the power of 1 = 5

5 to the power of 2 = 25

5 to the power of 3 = 125

...

You may need to research how to perform this operation in Python or JavaScript. Refer to the Resources for cheat sheets.

*Solution code:* [*Python*](https://repl.it/@digitechhub/Lesson-7-Tinker-Task)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/3cwd0stL/)

# 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. Carefully examine the password check system in the flow chart below.

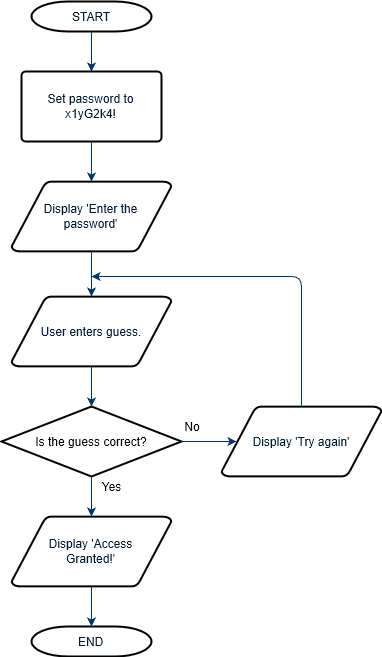
[](https://tinyurl.com/yxa7vafx)

Image: Flow chart for password check system

* 1. Convert the algorithm to pseudocode (structured English) first. You will need to use a **while** loop to handle the repetition, so it won’t be the exact same sequence as the flowchart.

|  |
| --- |
| **SOLUTION**  BEGIN  **password** ← “x1yG2k4!”  Display “Enter the password”  **guess** ← input from user  While **guess** is not equal to **password** repeat  Display “Try again”  **guess** ← input from user  End While  Display “Access Granted!”  END |

* 1. Code the program in Python or JavaScript.

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/314886111/)*,* [*Python*](https://repl.it/@digitechhub/Lesson-7-Challenge-1)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/ncr0pob9/)

1. Improve the password check program in Challenge 1 so that there is a limited number of guesses before the program ends. HINT: You’ll need to add a variable to keep track of the number of tries.

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/314886424/)*,* [*Python*](https://repl.it/@digitechhub/Lesson-7-Challenge-2)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/vf1xcors/)

1. (Optional) Create a countdown, rather than counting up. The output of the program should be:

Beginning countdown

10

9

8

7

6

5

4

3

2

1

Blast off!

There are at least two ways you can do this, as shown in these solution codes. One way does not require any new knowledge about loops.

*Solution code:* [*Scratch*](https://scratch.mit.edu/projects/314886952/)*,* [*Python*](https://repl.it/@digitechhub/Lesson-7-Challenge-3)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/5Lfj379m/)

1. (Optional) Choose from the links below to find pre-made code with some arrays declared. The code also includes a simple program to display all the items in the **colours** array using a loop.

*Solution code:* [*Python*](https://repl.it/@digitechhub/Lesson-7-Challenge-4-Starter)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/woLx4j0m/)

Add some code to display all the items in the **months** array using a loop.

Finally, add some code to find the sum of all the items in the **numbers** array.

*Solution code:* [*Python*](https://repl.it/@digitechhub/Lesson-7-Challenge-4-Solution)*,* [*JavaScript*](https://jsfiddle.net/nathanesa/r31f4j7v/)

# 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.)