# Visual to text coding LESSON 9: **Loops and arrays combined**

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

This is the ninth 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 explores creating powerful programs for managing and analysing data, by combining the previous skills of using loops and working with arrays.

# 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) . |
| Year 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 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

Start by watching the short video *What is Big Data and how does it work?*

In previous lessons, we've made simple tools and games. This lesson is about combining the power of **arrays** and **loops** to write programs for one of the most important tasks computers have: processing data.

We'll see that arrays and loops work hand-in-hand. Arrays store hundreds, thousands or millions of values, and loops allow us to access and manipulate each stored value in the same way.

Without the simple techniques in this lesson, we'd never be able to work with big data.

# 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 program that combines arrays and loops to analyse student marks and produce summary information,
3. use the combined power of arrays and loops to process new data.

# Learning input

Begin by watching the video demonstrating the student scores program:

As a class, read over the structural pseudocode below then answer following questions one by one to help understand the program and what it does. (Click here for a Scratch adaptation of the program.)

BEGIN

**student\_marks** ← [83, 72, 92, 65, 54, 54, 78, 67, 52, 54, 48, 69,  
 87, 55, 51, 52, 44, 57, 79, 64, 66, 19, 82, 71,   
 66, 31, 87, 83, 64, 78]

Using a loop...

Display each student's number and mark from **student\_marks**

**high\_mark** ← determine the highest mark in the **student\_marks**

Using a loop...

**low\_mark** ← determine the lowest mark in the **student\_marks**

Using a loop

**grade\_a** ← assemble an array of student numbers with marks >=85

**grade\_b** ← assemble an array of student numbers with >=65 and <85

**grade\_c** ← assemble an array of student numbers with >=45 and <65

**grade\_d** ← assemble an array of student numbers with >=25 and <45

**grade\_e** ← assemble an array of student numbers with <25

Display **high\_mark**

Display **low\_mark**

Display **grade\_a**

Display **grade\_b**

Display **grade\_c**

Display **grade\_d**

Display **grade\_e**

END

**QUESTIONS:**

1. What is the main input data for the program?

2. How many student marks are in the array?

3. After listing off the marks, the program displays two useful pieces of information about the student marks that were determined earlier. What are these?

4. Finally, the program displays five arrays assembled earlier. What do these store?

5. IMPORTANT: Why are loops necessary for this program?

6. Don't Python and JavaScript have built-in functions to work with arrays?

**ANSWERS**

1. An array of whole numbers for student marks.

2. There are 30 marks in the array.

3. The highest mark achieved and the lowest mark achieved.

4. The array **grade\_a** stores the student numbers for all marks from the input array that are greater than or equal to 85. **grade\_b** works on marks greater than or equal to 65 and less than 85, etc.

(Student numbers are just the positions in the input array, ie. the first mark in the input array is student number 0, the second mark is student number 1, and so on.)

5. Loops and arrays work hand-in-hand. The **student\_marks** array contains 30 numbers, so it would be very tedious (and not very adaptable) to access each number one-by-one over 30 lines of code. As we'll see in this lesson, loops allow us to *iterate* through the array, accessing each number with the short code inside the loop.

6. It is true that Python and JavaScript have convenience functions for working with arrays. For example, Python's print(student\_marks) will display the array contents. Another function max(student\_marks) gets the highest value.

But these functions are limited in *how* they do what they do. While print(student\_marks) will display the array contents, it does not allow control over *how* it is displayed. The function max(student\_marks) gives us the highest value, but what if wanted to ignore outliers, or perform some custom operation while working through the array?

SIDEBAR – Two types of loops

Remember, 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 can be used to pick each item's position as the loop works through an array.  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: Getting highest and lowest marks**

This video demonstrates coding the first part of the student marks program in Python. Try it yourself! Choose a link below to start with the data already in place, then check the completed code so far.

*Code skeleton with data in place on Python, JavaScript*

*Check the completed code so far on Python, JavaScript*

**Step 2: Tinker task**

First, add a few more scores to the array at the start: 78, 71, 14, 96 and 84. Test your program to make sure it still works as expected.

Next, see if you can determine the average (mean) score and display it. You'll need a loop to find the sum of all the scores first.

*Solution code on Python, JavaScript*

**Step 3: Completing the program**

This video demonstrates coding the remainder of the student marks program in Python. Try it yourself before checking the completed code (including JavaScript) below.

*Completed code on Python, JavaScript*

**Step 4: Tinker task**

At the end of the program, display the number of students who received an A grade, the number of students who received a B grade and so on.

*Solution code on Python, JavaScript*

# Challenge

These challenges use the skills covered so far.

1. A class of students has calculated the grams of sugar in their lunches, and you have collected the following data:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 17 g | 10 g | 14 g | 15 g | 12 g | 12 g | 14 g | 9 g |
| 12 g | 23 g | 6 g | 12 g | 24 g | 10 g | 4 g | 16 g |
| 17 g | 10 g | 21 g | 23 g | 3 g | 20 g | 8 g | 7 g |

Your task is to write a program to assign star ratings to the lunches based on the amount of sugar.

* Lunches with < 6g of sugar receive 5 stars.
* Lunches with >= 6g and < 10g receive 4 stars.
* Lunches with >=10g and < 15g receive 3 stars.
* Lunches with >=15g and < 20g receive 2 stars.
* Lunches with >=20g receive 1 star.

For this challenge, you don't need to create arrays for the star ratings, but you do need to loop through the main data array and display a rating for each lunch as follows:

Lunch no. 0 gets 2 stars.

Lunch no. 1 gets 3 stars.

Lunch no. 2 gets 3 stars.

...

* 1. Prepare pseudocode first.

|  |
| --- |
| **SAMPLE SOLUTION**  BEGIN  **lunches** ← [17, 10, 14, 15, 12, 12, 14, 9, 12, 23, 6, 12, 24, 10, 4, 16, 17, 10, 21, 23, 3, 20, 8, 7]  **noOfLunches** ← 24  For **i** from 0 to **noOfLunches** - 1  If **lunches**[**i**] < 6  **starRating** ← 5  Else If **lunches**[**i**] < 10  **starRating** ← 4  Else If **lunches**[**i**] < 15  **starRating** ← 3  Else If **lunches**[**i**] < 20  **starRating** ← 2  Else  **starRating** ← 1  End If  Display 'Lunch no.', **i**, 'gets', **starRating**, 'stars.'  End For  END |

* 1. Code the program in Python or JavaScript.

*Solution code: on Python, JavaScript*

1. A) A Selection Sort is one of the simplest algorithms for sorting values in an array, that is, putting the values in order. In this challenge, you'll write code for a selection sort that works by repeatedly moving the smallest value in an **unsortedList** into a **sortedList**.

First, read the pseudocode below. As a class or in pairs, use a trace table to test the algorithm.

Download a template version and a completed version of a trace table (A3 size).

BEGIN

**unsortedList** ← [11, 25, 12, 22, 64]

**sortedList** ← []

**noOfValues** ← length of **unsortedList**

Display "Here's the unsorted array: ", **unsortedList**

// Repeat the whole algorithm enough times to move every value.

For **i** from 0 to **noOfValues** - 1

// Identify the smallest value currently in the unsorted list.

**smallest** ← 100

For **j** from 0 to length of **unsortedList** - 1

If **unsortedList**[**j**]< **smallest**

**smallest** ← **unsortedList**[**j**]

End If

End For

// Move the smallest value across to the sorted list.

Remove **smallest** from **unsortedList**

Append **smallest** to **sortedList**

// Display as we go.

Display "Here's the sorted array: ", **sortedList**

End For

END

B) Next, implement the code in Python or JavaScript, using these tips:

* In Python, the **remove** function will search for a value and remove it from an array.  
  eg.   
  // Search for and remove "Australia" from the array.  
  countries.remove("Australia")
* In Javascript, the **splice** function will do a similar job, but you have to specify the *position* of the value to remove, as well as the number of values to remove. This means the JavaScript code will have to remember the position of the smallest value, not just the smallest value itself.  
  eg.   
  // Remove one country at position 3 in the array.  
  countries.splice(3, 1)

*Complete solution code: on Python, JavaScript*

Finally, try adding more values (below 100) to the unsorted list. Does the algorithm still work?

1. (OPTIONAL) Your friend Zippy is developing a fairground game in which a player is asked to choose a number between 1 and 60 inclusive. To decide whether the player wins, the chosen number goes through the following test:

*Subtract 25 from the number. Multiply the result by 3. Finally, square that result to get the final value. If the final value is greater than 1000, the player wins.*

Zippy says the game is very generous because most numbers between 1 and 60 will result in a win. But your other friend Wanda says that people don't choose numbers evenly. She says it's more of a curve with extreme numbers like 2 and 59 being much rarer than numbers like 29 or 31.

To really investigate how fair the game is, you decide to code a simulator:

* First, use a loop to generate a simple array of 60 numbers called **simpleNumbers**. It will consist of all the whole numbers from 1 to 60.
* Next, use a loop to generate a more complex array called **curvedNumbers**. It will have a total of 1000 numbers. To represent Wanda's curve, each of these numbers will be generated by rolling three 20-sided dice and adding them up. eg. 12 + 9 + 14 = 35.
* Next, use a loop to run the winning or losing test on **simpleNumbers**. You need to be able to display the total number of winners and the total size of the array, eg.   
  "Out of 60 numbers, there were ??? winners. This is a ??? percent chance of winning."
* Finally, you can test the other array by reusing the test loop code and running it on **curvedNumbers**, eg. "Out of 1000 numbers, there were ??? winners. This is a ??? percent chance of winning."

*Sample solution code: on Python, 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
* 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.)