# Visual to text coding LESSON 11: **Adaptable Functions**

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

This is the eleventh 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 builds on the coding concept of functions. With the addition of **parameters**, functions allow the programmer to adapt their reusable code’s behaviour, tapping into the Computational Thinking skills of generalisation, pattern recognition and abstraction.

# 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



Image from Wikipedia CC-0 (public domain)

Pig Latin is a language game where English words are altered to sound different, but using deceptively simple rules.

Here are some examples:

* **she eats too much duck** becomes **eshay eatsay ootay uchmay uckday**
* **have three old bananas** becomes **avehay eethray olday ananasbay**
* **his floor feels cold** comes **ishay oorflay eelsfay oldcay**

For simplicity, we'll say there are three main rules for converting an English word into a Pig Latin word. Each rule has the following format:

If the word starts with **[something]**, then **[make changes]**.

As a class, study the examples above and try to figure out the three main rules for converting an English word into a Pig Latin word.

Click to reveal: the first part of each rule:

1. if the word starts with a consonant, like "**p**ig" or "**d**uck"…
2. if the word starts with a consonant cluster, like "**thr**ee" or "**fl**oor"…
3. if the word starts with a vowel, like "**e**gg" or "**e**ats"…

Click to reveal: each complete rule:

1. if the word starts with a consonant, like "**p**ig" or "**d**uck", then move that consonant to the end, and add "ay" after that.
2. if the word starts with a consonant cluster, like "**thr**ee" or "**fl**oor", then move the cluster to the end, and add "ay" after that.
3. if the word starts with a vowel, like "**e**gg" or "**e**ats", then just add "ay" to the end.

Using these three rules, you could create a **function** to make a Pig Latin word from any supplied English word, then display it. Complete the gaps in the function pseudocode below:

Function convertToPigLatin(**englishWord**)

**pigWord** ← ''

If **englishWord** starts with a consonant then

**pigWord** ← all except the first letter from **englishWord**

Append the first letter from **englishWord** to **pigWord**

End If

If **englishWord** starts with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ then

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

End If

If **englishWord** starts with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ then

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

End If

Append 'ay' to **pigWord**

Display 'The Pig Latin word is ', **pigWord**

End Function

See Wikipedia for more details on the rules for Pig Latin.

# 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. learn the vocabulary of arguments and parameters,
3. practice writing functions that include parameters,
4. observe how functions reduce repetition in code,
5. generate a size chart using a custom shape to visualise data.

# Learning input

In this video a simple function for drawing a square is improved by adding a **parameter**.

Let's do a vocabulary check:

* **Arguments** are the values that are supplied when calling a function. Eg. 50, 100, 25
* **Parameters** are variables in a function for holding the values that are supplied. Eg. *side* takes the values 50, 100 or 25, whatever is supplied when the call is made.

# Learning construction

**STEP 1: SETUP**

View the video on setting up functions.

For more on setting up, see Lesson 10. This lesson continues with a turtle graphics approach for each language.

**SIDEBAR – What’s with the turtle?**

Turtle graphics has been around since the original Logo language for teaching programming to students, developed in 1966-1967. It is being introduced at this point in this course because it is an effective context to learn about functions, but examples will also be provided for writing functions outside of the turtle graphics context.

Like today’s Bee-bots and other rovers used in classrooms, the 'turtle' can be thought of as a robot moving about on the floor, and capable of drawing lines as it moves. Programmers issue simple instructions like **forward 20** to move forward 20 units of distance, or **right 90** to rotate clockwise 90 degrees.

To activate the pen, the typical command is **pen down** (think of a pen being dragged along the floor by the robot). The command **pen up** lifts the pen off the floor, allowing the robot to move around without drawing.

In Python and JavaScript environments with turtle graphics, these commands can be typed and run within your programs just like other lines of code.

In Scratch, sprites are already moved around using the blue motion blocks. Any sprite can become a drawing turtle by using the Pen extension, which brings in the **pen down** and **pen up** commands.

**STEP 2: WRITING A FUNCTION WITH ONE PARAMETER**

This video demonstrates writing a simple function with one parameter. Try it yourself!

*Completed code: Scratch, Python, JavaScript*

**STEP 3: WRITING A FUNCTION WITH TWO PARAMETERS**

This video adds a second parameter to the function. Try it yourself!

*Completed code: Scratch, Python, JavaScript*

**STEP 4: FUNCTION EXERCISE WITH TURTLE GRAPHICS**

Carefully read the pseudocode below.

1 BEGIN

2 Function drawDiamond(**sizeFactor**)

3Pen down

4 Left 60

5 Forward 30 \* **sizeFactor**

6 Left 60

7 Forward 30 \* **sizeFactor**

8 Left 120

9 Forward 30 \* **sizeFactor**

10 Left 60

11 Forward 30 \* **sizeFactor**

12 Left 60

13 Pen up

14 EndFunction

15

16 drawDiamond(1)

17 Forward 50

18 drawDiamond(1.5)

19 Forward 50

20 drawDiamond(0.5)

21 Forward 50

22 END

On paper, try to predict what the program will draw. The turtle starts facing right.   
Share with another student to see if you predicted the same outcome.

**Click for answer:**

A close up of a logo

Description automatically generated

Finally, implement the code in Scratch, as well as Python *or* JavaScript.

*Solution code: Scratch, Python, JavaScript*

**STEP 5: FUNCTION EXERCISE WITHOUT TURTLE GRAPHICS**

Carefully read the pseudocode below. This program does not use turtle graphics.

1 BEGIN

2 Function displayWordRepeatedly(**word**, **multiple**)

3 **i** ← 1

4 While **i** <= **multiple**

5 Display **i** + ' ' + **word**

6 **i** ← **i** + 1

7 End While

8 EndFunction

9

10 displayWordRepeatedly('banana', 5)

11 displayWordRepeatedly('apple', 3)

12 END

Predict the output of the program.

Click for answer:

1 banana

2 banana

3 banana

4 banana

5 banana

1 apple

2 apple

3 apple

Now, implement the code in Python *or* JavaScript:

* If doing Python, use a regular Python project in repl.it as in Lessons 1 to 9.
* If doing JavaScript, use JSFiddle as in Lessons 1 to 9.

*Solution code: Python, JavaScript*

**STEP 6: REDUCING REPETITIVE CODE WITH FUNCTIONS**

This video demonstrates the power of functions to reduce repetitive code, and ends with a challenge. The video shows Python, but it can be applied the same in JavaScript.

*Challenge code to work from: Python, JavaScript*

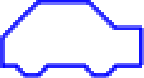
The next video concludes with a sample solution.

*Sample solution: Python, JavaScript*

# Challenge

These challenges use the skills covered so far.

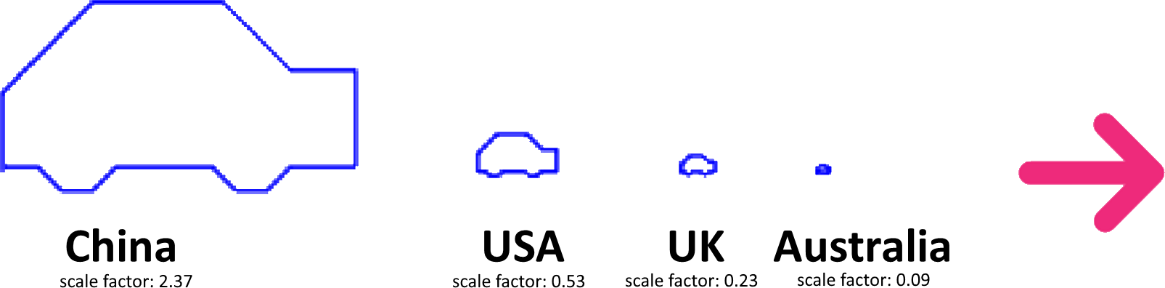
1. Write and test a function to create a car shape as below (must be all straight lines).



Now, modify your function to have a **sizeFactor** parameter, as with the diamond in step 4 of this lesson. Test it by calling it from the main program with arguments 1, 2 and 0.5.

Finally, call your finished function with specific values so as to generate a size chart of 2018 new car sales data for four countries - China, USA, UK and Australia. eg. China's sales were about 23.7 million, so use a size factor of 2.37. USA size factor would be 0.53, and so on.

Click for a sample visual of this solution (labels are not part of the program):

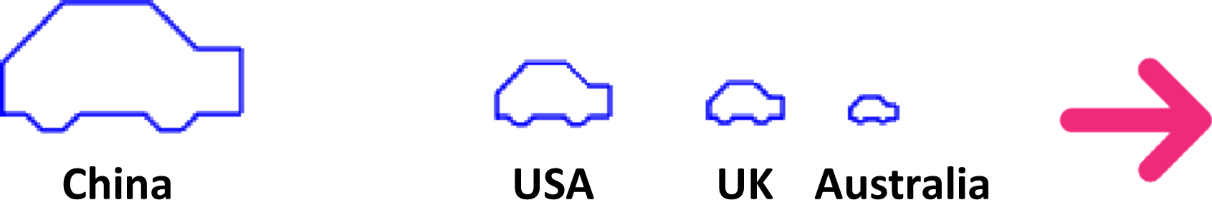


*Sample solution code: Scratch, Python, JavaScript*

**Optional discussion question:** Do you think your function allows your chart to give a fair representation of the difference in car sales between these four countries? How could you adjust it to more accurately reflect the differences?

**Sample answer:**

The **sizeFactor** parameter is simply multiplied to each forward movement, making it a linear scale factor. It would be better if the size of each car symbol was scaled according to its area. This could be done by replacing the size factor with its square root each time. The solution would now show as follows:



1. Write and test a function to generate and display the sequence of Hailstone numbers for any given starting number. The sequence should continue until you get to the number 1, or the sequence reaches a thousand numbers in length, whichever comes first.

Hailstone numbers are generated as follows:

Start with a given first number.

* If the first number is even, divide it by two to get the next number.
* Otherwise (odd), triple the first number and add 1 to get the next number.

Now repeat the above with the next number to get the following one, and so on.

eg. Starting with 12, we get the sequence: 12, 6, 3, 10, 5, 16, 8, 4, 2, 1. This sequence reaches 1 with only ten numbers. Begin by writing pseuocode for the function and a simple test. Your function will need a loop inside.

**Click for simplified/structural pseudocode:**

BEGIN

Function generateHailstones(**firstNumber**)

Display **firstNumber** and use as **number**

Loop while **number** isn't 1 AND the sequence isn't too long

Change **number** according to the rules

Display **number**

End Loop

End Function

generateHailstones(12)

END

**Click for complete pseudocode solution:**

BEGIN

Function generateHailstones(**firstNumber**)

Display 'Here is the sequence:'

Display **firstNumber**

**number** ← **firstNumber**

**sequenceLength** ← 1

While **number** Is Not 1 And sequenceLength < 100

If **number** Is Even

**number** ← number ÷ 2

Else

**number** ← number × 3 + 1

End If

Display **number**

sequenceLength ← sequenceLength + 1

End While

End Function

generateHailstones(12)

END

Now write the code in Python or JavaScript.

*Sample solution code: Python, JavaScript*

1. *(More challenging)* Write a function to convert an English word to Pig Latin and display it, using the rules in the Learning Hook section of this lesson.

* If doing Python, use a regular Python project in repl.it as in Lessons 1 to 9.
* If doing JavaScript, use JSFiddle as in Lessons 1 to 9.

Whether you code in Python or JavaScript, you will need to investigate commands for string manipulation, such as identifying characters that appear at the beginning of text. Note: Scratch does not have the required string manipulation capabilities. (This challenge can be done using spreadsheet functions in Microsoft Excel or Google Sheets.)

*Sample solution code: Python, JavaScript, spreadsheet (Google Sheets)*

# 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.