Cracking the Pseudocode:

Teaching Algorithms From a Student Perspective

Lianna Beeching and Amanda Fan

Preston High School

THE MATHEMATICAL ASSOCIATION OF VICTORIA

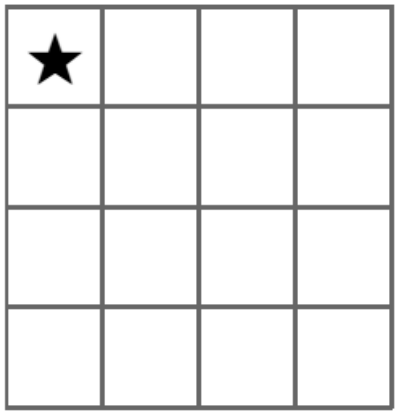
CURRICULUM, PEDAGOGY AND BEYOND

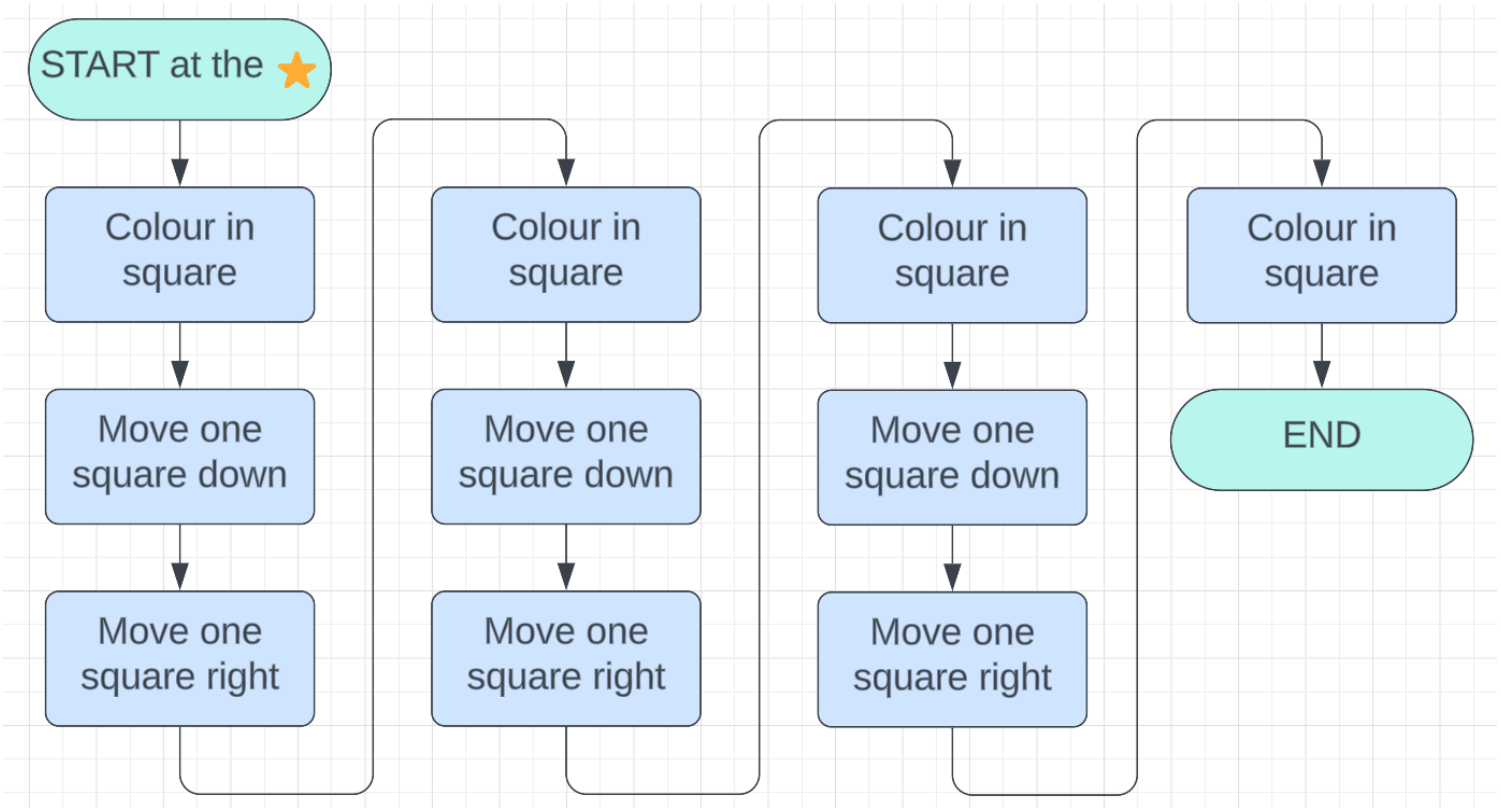
MAV24 CONFERENCE

We meet today on the lands of the Wurundjeri people of the Kulin Nation and pay our respects to Elders past, present and emerging.

Learn more at www.wurundjeri.com.au

Draw the picture using the algorithm!





What do you notice about this code?
What's inefficient about it?
How could we re-write it to be more efficient?

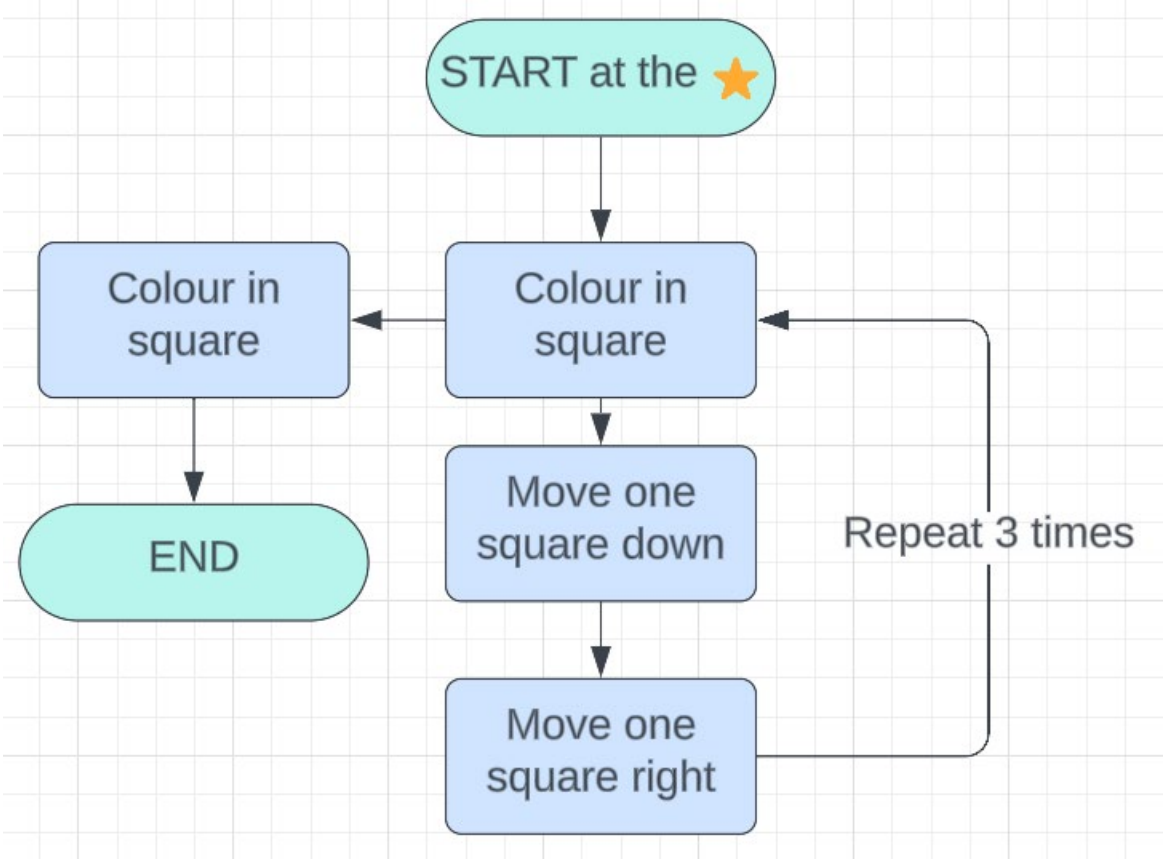
Session Structure

	Graph Paper Programming (Flow charts and unplugged learning)
15 min	Student Experience
5 min	Teacher Lens: Pedagogy and Curriculum
5 min	Next Steps: Ideas and Resources
	Pseudocode Intro and Desk Checking
15 min	Student Experience
5 min	Teacher Lens: Pedagogy and Curriculum
5 min	Next Steps: Ideas and Resources
5 min	Questions, Discussion and Survey

Graph Paper Programming

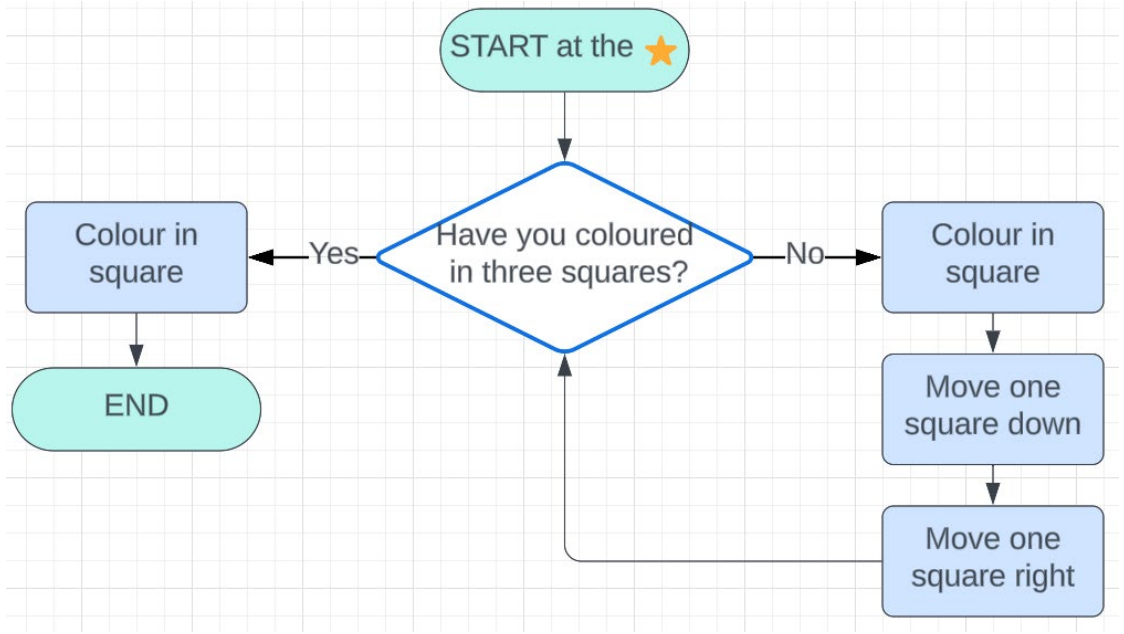
(Flow Charts and Unplugged Learning)

How can we improve our earlier algorithm? Loops!



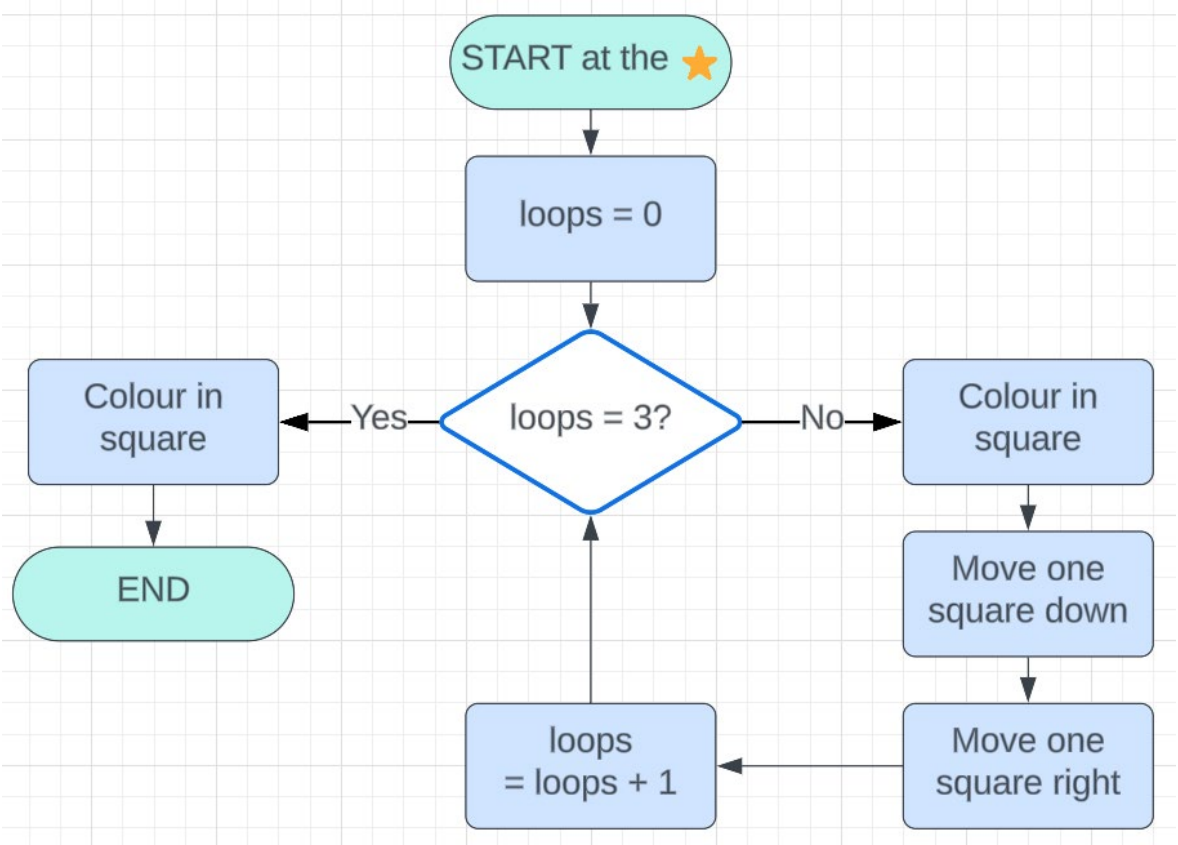

```
Repeat three times:
    Colour in square
    Move one square down
    Move one square right
Colour in square
```

Most loops check if they should keep running based on a **condition**:




```
While less than three squares coloured
    Colour in square
    Move one square down
    Move one square right
EndWhile
Colour in square
```

A computer needs to keep track of how many loops have happened so far - this is where a **counter variable** comes in!




```
loops = 0
While NOT loops = 3
    Colour in square
    Move one square down
    Move one square right
    loops = loops + 1
EndWhile
Colour in square
```

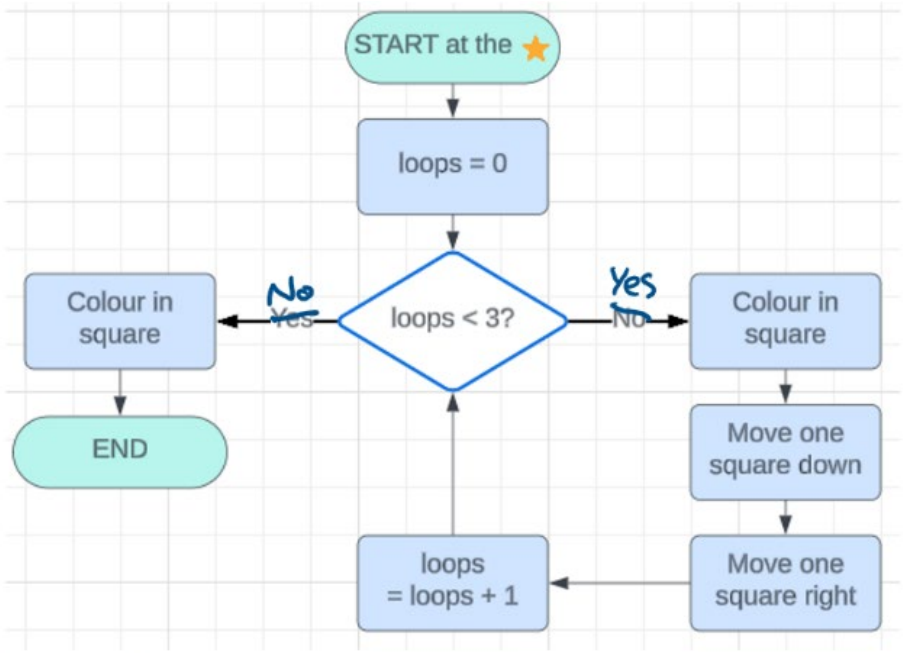
What would happen if we didn't have the step "loops = loops + 1"?

A **desk check** is where you run an algorithm with pen and paper!

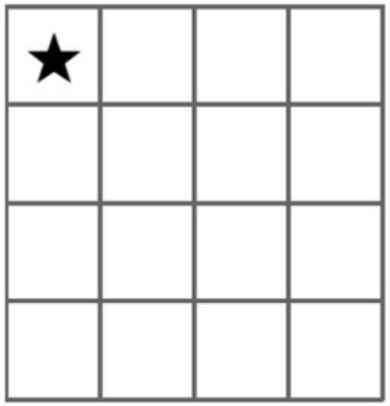
Desk checks are a great way to understand how an algorithm works and to check for bugs/ errors.

For this desk check, we will:

- Follow each step, one line at a time
- Write down the value of variables as they change



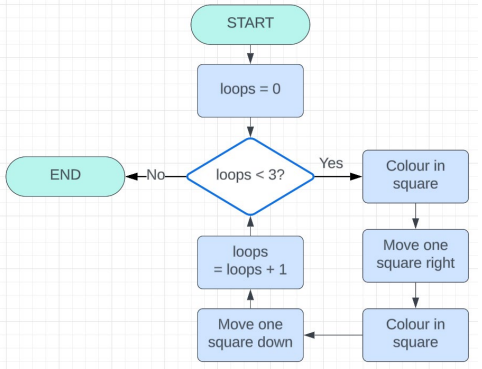
Desk check:



loops	

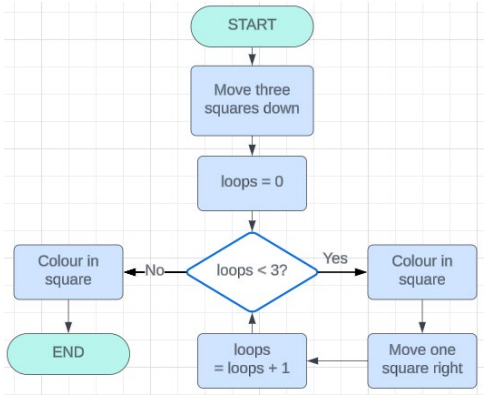
Your Turn: Solve these puzzles by desk checking!

Puzzle 2: mild




```
loops = 0
While loops < 3
    Colour in square
    Move one square right
    Colour in square
    Move one square down
    loops = loops + 1
EndWhile
```

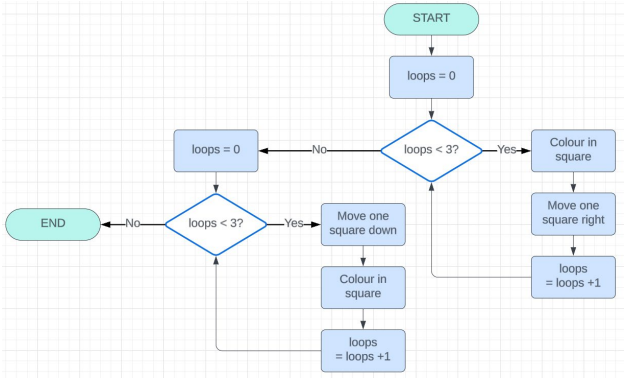
Puzzle 3: mild




```
Move three squares down
loops = 0
While loops < 3
    Colour in square
    Move one square right
    loops = loops + 1
End while
Colour in square
```

Puzzle 4: medium

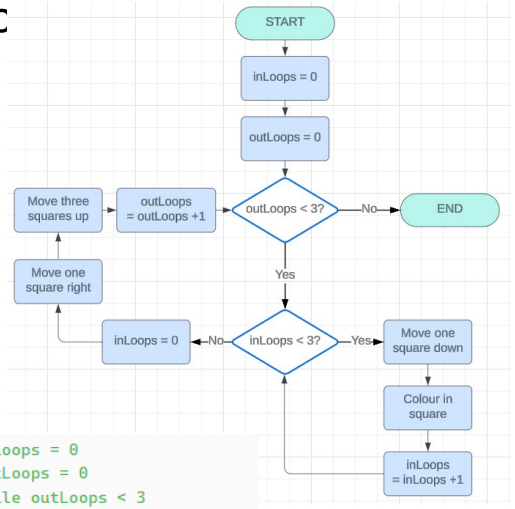
This is an example with two loops in a row!




```
loops = 0
While loops < 3
    Colour in square
    Move one square right
    loops = loops + 1
EndWhile
loops = 0
While loops < 3
    Move one square down
    Colour in square
    loops = loops + 1
EndWhile
```

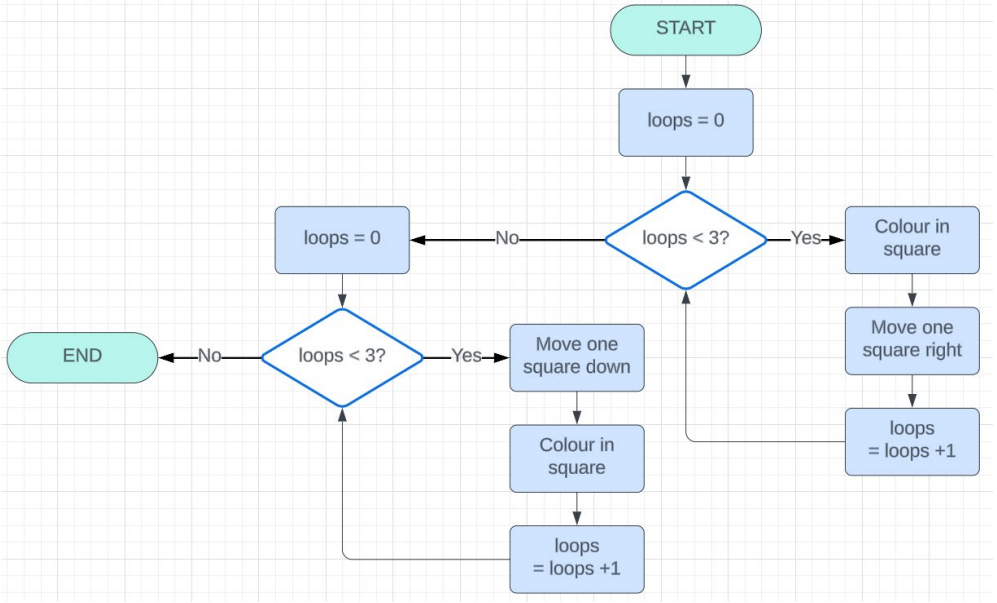
Puzzle 5: spicy

This is an example of a nested loop, a loop inside a lo

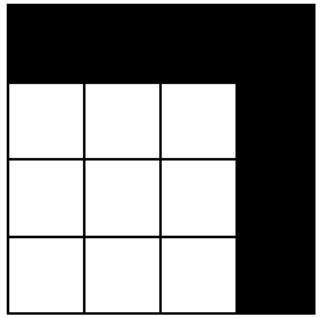



```
inLoops = 0
outLoops = 0
While outLoops < 3
    While inLoops < 3
        Move one square down
        Colour in square
        inLoops = inLoops + 1
    End while
    inLoops = 0
    Move one square right
    Move three squares up
    outLoops = outLoops + 1
End while
```

Modify a Puzzle!

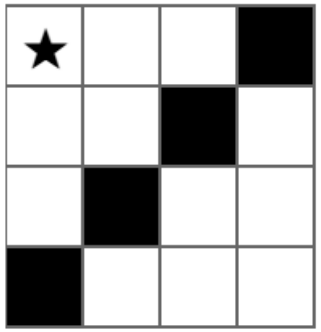


Add a step into puzzle 4 so that it creates this pattern instead:

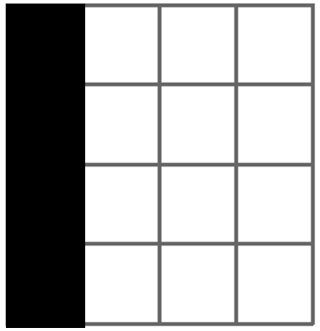


Your Turn: Write an Algorithm to Create Each Picture!

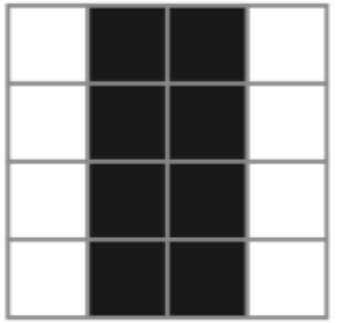
Puzzle 6 (fill in the blanks)



Puzzle 7



Puzzle 8



Linking to the Maths 2.0 Curriculum

Level 5 (number)	Level 6 (algebra)	Level 7 (space)	Level 8 (algebra, space)	Level 9 (space)
VC2M5N10 Follow a mathematical algorithm involving branching and repetition (iteration); create and use algorithms involving a sequence of steps and decisions and digital tools to experiment with factors, multiples and divisibility; identify, interpret and describe emerging patterns.	VCM6A03 Design and use algorithms involving a sequence of steps and decisions that use rules to generate sets of numbers; identify, interpret and explain emerging patterns.	VC2M7N10 Design algorithms involving a sequence of steps and decisions that will sort and classify sets of shapes according to their attributes and describe how the algorithms work.	VC2M8A04 Use algorithms and related testing procedures to identify and correct errors. VC2M8SP04 Design and test algorithms involving a sequence of steps and decisions that identify congruency or similarity of shapes and describe how the algorithm works.	VC2M9SP03 Design, test and refine algorithms involving a sequence of steps and decisions based on geometric constructions and theorems; discuss and evaluate refinements.

Linking to the Digi Tech 2.0 Curriculum

Level 5-6	Level 7-8	Level 9-10
VC2DTCD032 Design, modify and follow simple algorithms represented diagrammatically and in English, involving sequences of steps, branching, and iteration.	VCDTCD042 Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors.	VC2DTCD052 Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases.

Linking to the 2.0 Curriculum

Level 5 (number)	Level 6 (algebra)	Level 7 (space)	Level 8 (algebra, space)	Level 9 (space)
Follow, modify and create algorithms with sequences of steps, branching (IF) and iteration (Loops)		Trace algorithms to predict outputs and to spot and correct errors. Continue designing algorithms like in levels 5-6. Start describing how they work.		Similar to levels 7-8. Develop tests to check if an algorithm functions the way it should.
Content link: factors, multiples, divisibility	Content link: sequences/ patterns of numbers	Content link: classifying/ properties of shapes	Content link: similarity/ congruence tests	Content link: constructing angles and shapes
	Likely need to intro variables to do the content link		Start introducing “structured English” (pseudocode)	Good time to explore logical vs. semantic errors

Linking to Pedagogy

Desk Checks/ Tracing

Scaffolding and difficulty ramp – Moving from following to writing

There is research to support the usefulness of tracing code. In 2011 Lister describes that novices need to be able to trace code with more than 50% accuracy before they can begin to confidently write programs of their own (Lister, 2011). This follows on from years of research or this topic including a multi-institutional study published in 2004 showing that tracing code improves progaramming skills. Tracing is a practice that basically embodies a range of skills – a new programmer needs to be able to read, understand and explain how code works before being able to confidently write new code.

(Clear, 2011)

Linking to Pedagogy

Desk Checks/ Tracing

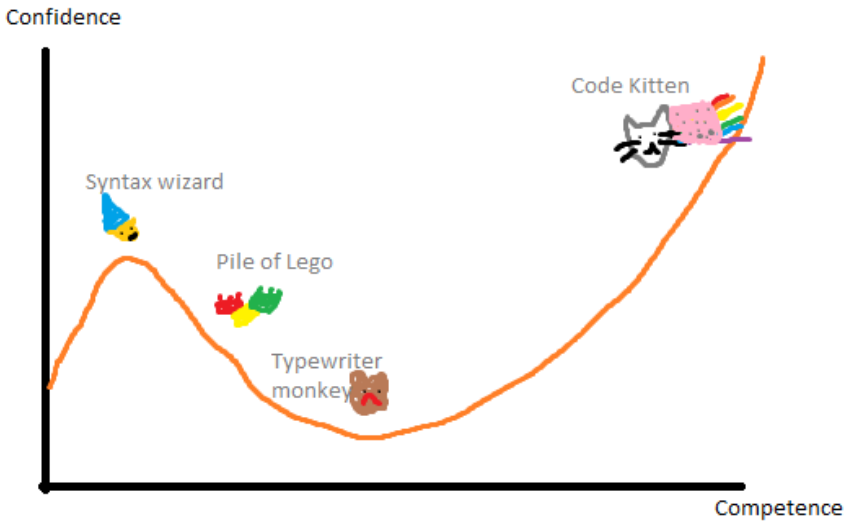
Scaffolding and difficulty ramp – Moving from following to writing

- After running this activity with three classes, this was the trickiest bit!
- Year 8s struggled more than Year 7s – perhaps confidence/ dispositions are a bigger barrier than the actual task
- Scaffolding with a fill the blanks helped
- Very slowly increasing the complexity of the algorithms involved when starting writing helped
- Adding the “modify” step – not sure yet if it helped!

Linking to Pedagogy

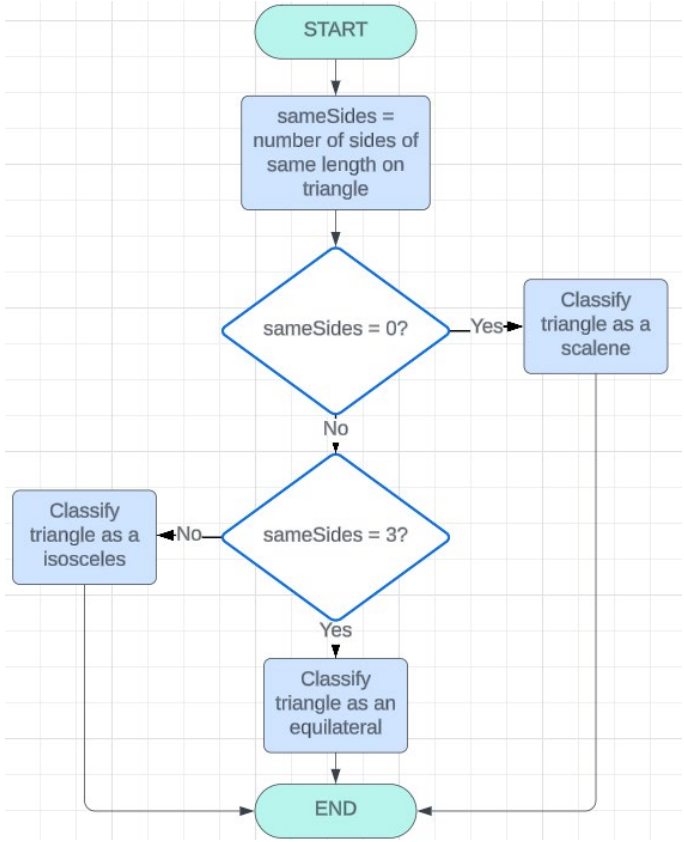
Desk Checks/ Tracing

Scaffolding and difficulty ramp – Moving from following to writing



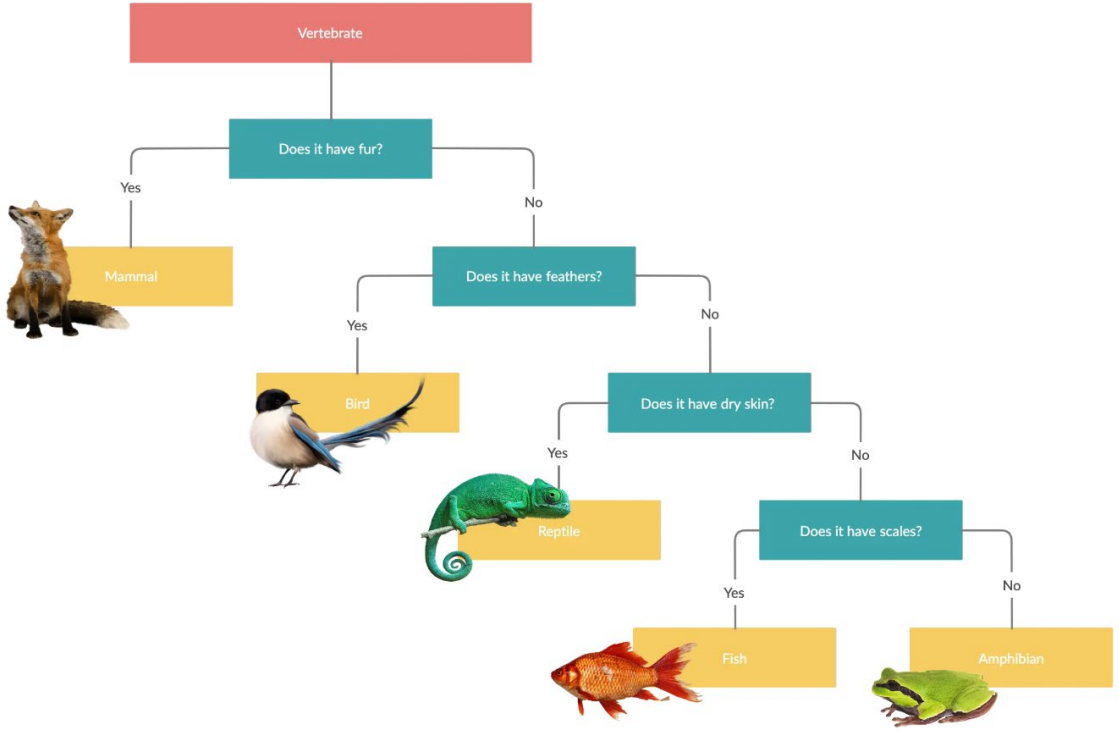
From the blog “Code? Boom”
https://codeboom.wordpress.com/2016/02/22/how-do-you-actually-teach-programming/

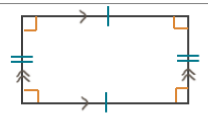
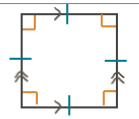
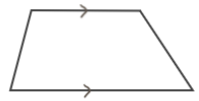
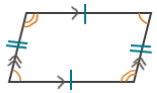
Next Steps – Other activity idea: Sorting Shapes



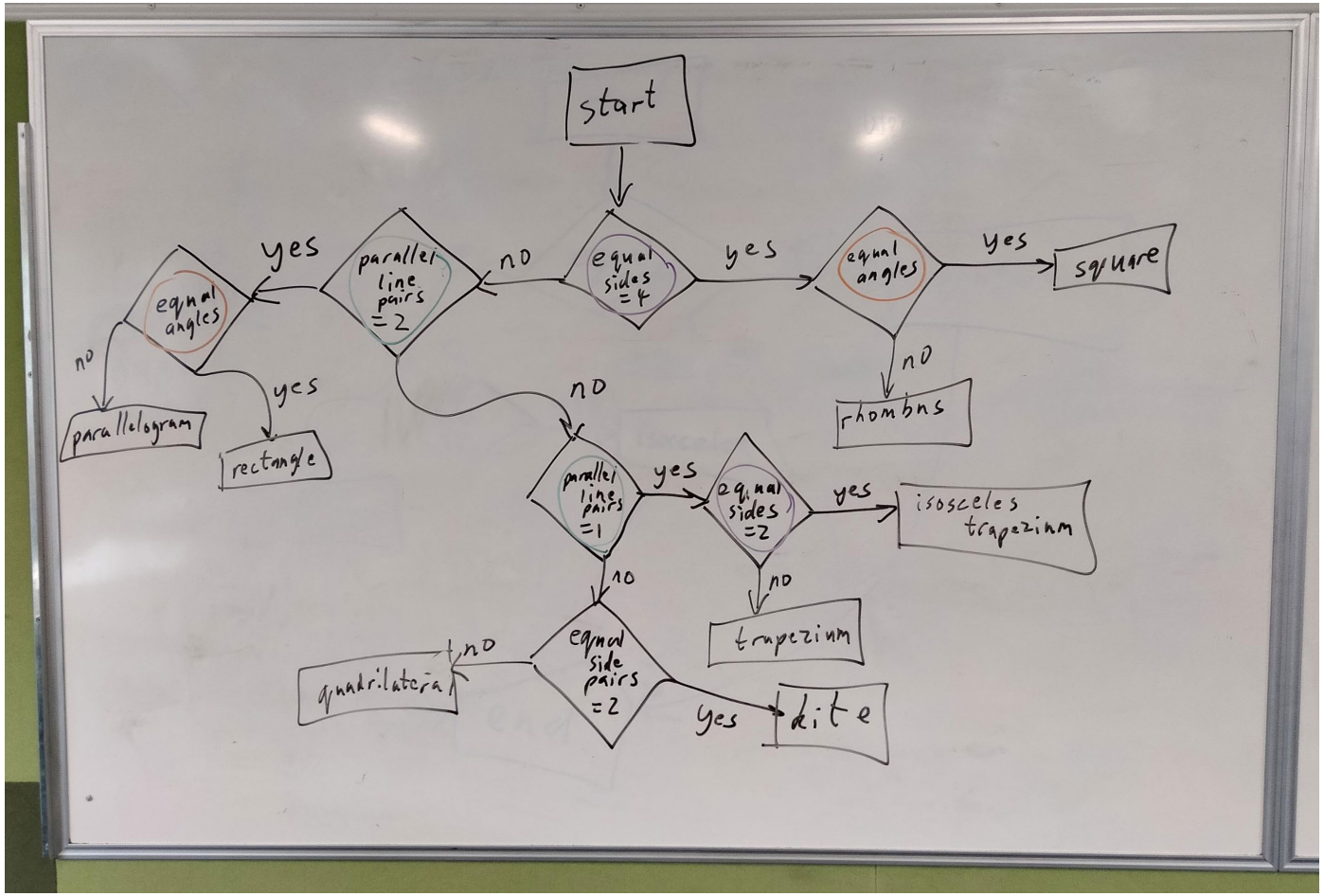

```
sameSides = number of sides of same length on triangle
If sameSides = 0 Then
    Classify triangle as a scalene
Else If sameSides = 3 Then
    Classify triangle as an equilateral
Else
    Classify triangle as an isosceles
End If
```

Next Steps – Other activity idea: Sorting Shapes



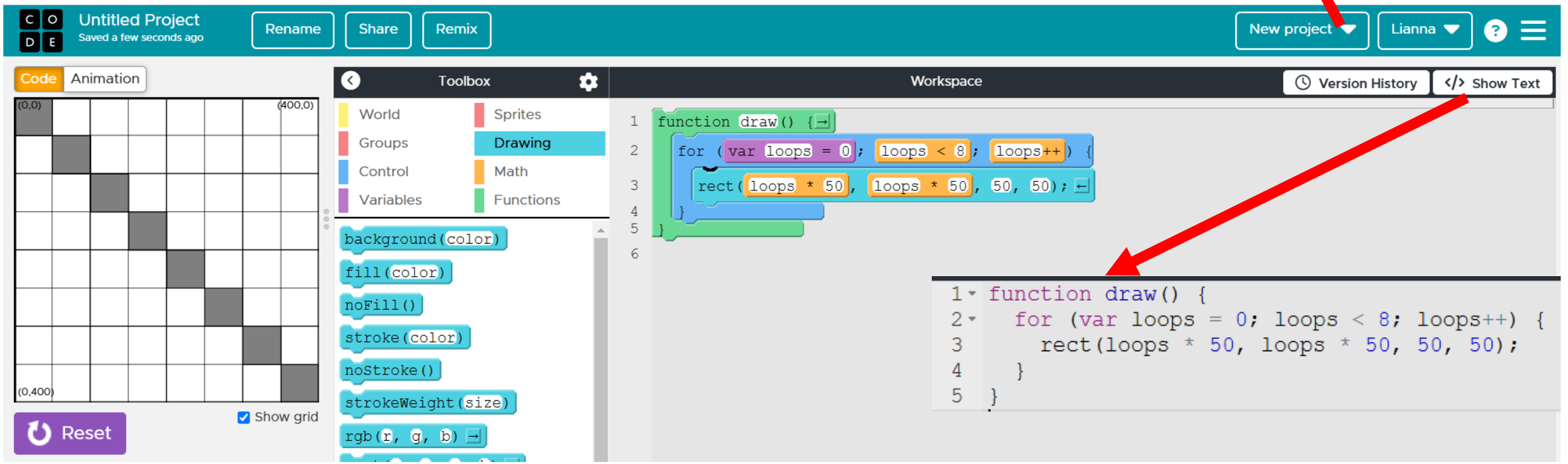
 **Rectangle** Opposite sides equal length All angles 90° Opposite sides parallel	 **Square** All sides equal length All angles 90° Opposite sides parallel	 **Trapezium** Sides any length Any angles Two sides parallel
 **Parallelogram** Opposite sides equal length Opposite angles equal Opposite sides parallel	 **Rhombus** All sides equal length Opposite angles equal Opposite sides parallel	 **Kite** Adjacent pairs of sides equal length Two equal angles (opposite) No parallel sides
	 **Quadrilateral** Any side lengths Any angles Sides can be not parallel	

Next Steps – Other activity idea: Sorting Shapes



Thanks to Chris Hill for trying this with his class and sending it through!

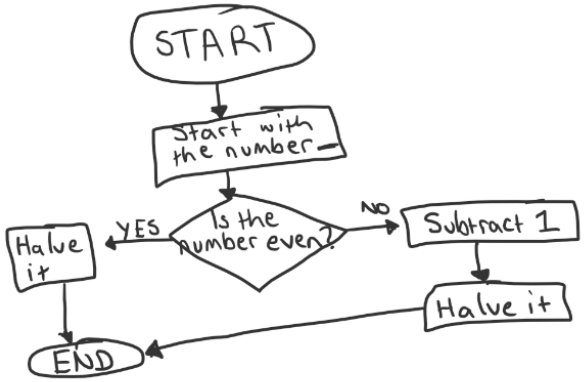
Next Steps –Taking this Activity Digital

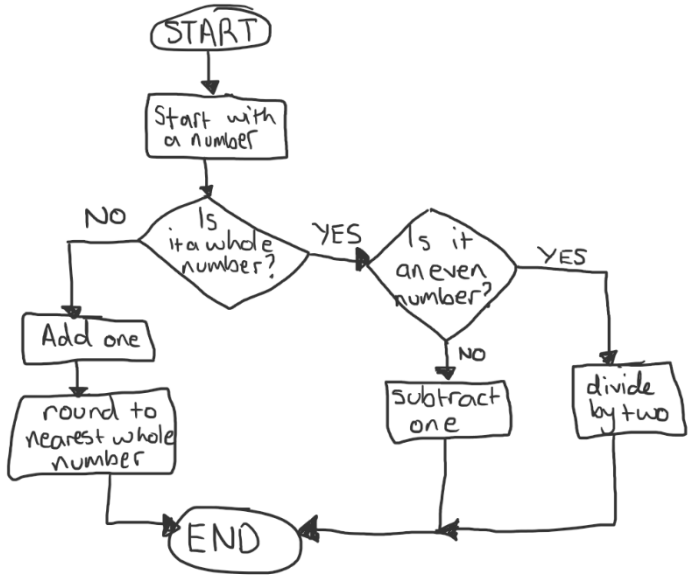


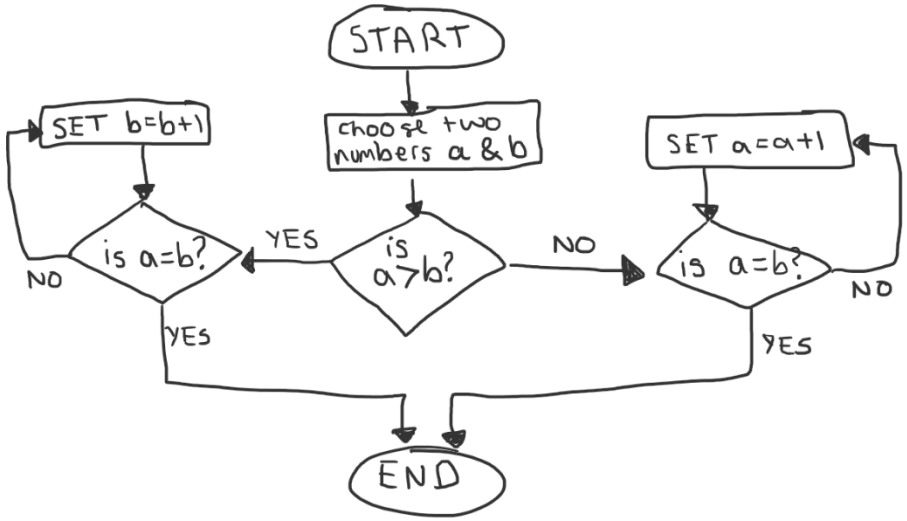
Thanks to Naomi Creelman for finding this online resource!

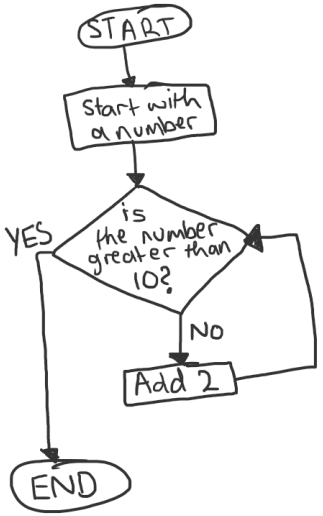
- A fantastic middle ground between block-based and structured code.
- Research suggests more structured coding blocks deepen understanding compared to unstructured blocks like in Scratch (Rose, 2016)
- Research also suggests that modifying code to achieve a specific goal is a great stepping-stone towards writing new code (Lee, et. al., 2011)

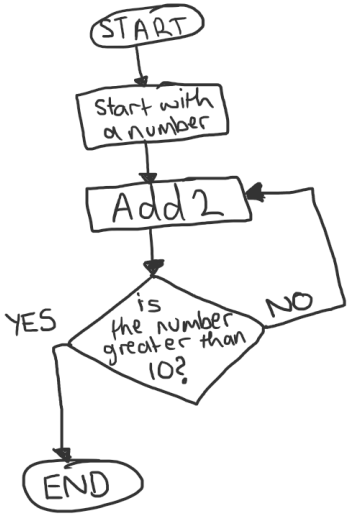
Next Steps – Bonus algorithms to steal!











Next Steps –Resources and Links

Resource	Description
Code.org game lab project	Structured block-based code editor (seen in previous slide)
Lucid Chart	Great for creating flowcharts (used in earlier slides) – to fiddly to be student-facing, but good for resource creation
Code.org, Grok Academy, codeclubau.org and CSS unplugged	Fantastic sources of unplugged lesson ideas and activities (the picture activity we did was adapted from code.org - https://studio.code.org/s/coursed-2022/lessons/2)

Pseudocode Intro and Desk Checking

Pseudocode Warm-Up: Assigning values to variables!

```
int r = 2;
int s = 4;
r = s;
```

In the code snippet to the right:

- What do you notice?
- What do you think each line does?
- What do you think the final values of r and s would be?

Problems taken from (Clear, 2011)

Pseudocode Warm-Up: Assigning values to variables!

Problems taken from (Clear, 2011)

Let’s try another!

Cheat sheet:

- “int x = 3” means “create an integer called $\boldsymbol{x}$ and assign it a value of 3”
- “x = y” means “assign $\boldsymbol{x}$ a value of $\boldsymbol{y}$ (the value of $\boldsymbol{y}$”

Pseudocode Warm-Up: Assigning values to variables!

```
int x = 5;
int y = 3;
int z = 7;
x = z;
y = x;
z = y;
```

Problems taken from (Clear, 2011)

Last one!

Cheat sheet:

- "int x = 3" means "create an integer called x and assign it a value of 3"
- "x = y" means "assign x a value of y (the value of y"

What is pseudocode?

Pseudocode is a human-friendly way of planning out code.

Computers can't read pseudocode, but we use it to jot down the ideas and structures behind a program/ algorithm, without needing to worry about specific languages and syntax.

Pseudocode doesn't have a specific way it needs to be written, but we will be trying to write it in a similar way to how VCAA (the people who write the year 12 exams) writes it.

Common Building Blocks in Pseudocode

Building block	Example	Description
Print	`print c`	Outputs/ displays a value

Assign a value	`i ← 1`
	`c ← c - 1`
	`x ← a + h`
		Sets the value of the variable on the left to the value on the right.

Return	`Return area`	Similar to print, but for outputting the final value of a function.
Inputs	See below	Defines the variables and functions involved at the start

```
Inputs: f(x), the function to integrate
        a, the lower terminal of integration
        b, the upper terminal of integration
        n, the number of trapeziums to use
```

Conditional Statements – IF, THEN, ELSE

Runs one chunk of code depending on a condition, then skips the other chunks of code.

```
input a, b
if a ≥ b then
    print a
else
    print b
end if
```

If a and b were inputted as 5 and 12 respectively, what's the output?

```
input mark
if mark ≥ 90 then
    print 'A'
else if mark ≥ 75 then
    print 'B'
else if mark ≥ 60 then
    print 'C'
else if mark ≥ 50 then
    print 'D'
else
    print 'E'
end if
```

If $mark$ was inputted as 68, what's the output?

Loops – FOR

Repeats the code inside the loop a certain number of times from a starting number to a finishing number.

Algorithm:

```
sum ← 0
for i from 1 to 4
    sum ← sum + i^2
end for
print sum
```

Desk check:

Algorithm:

```
input n
sum ← 0
for i from 1 to n
    sum ← sum + 1/i^3
end for
print sum
```

Desk check:

$(n = 3)$

Loops – WHILE

Repeats the code inside the loop while a condition is met.
Once the condition is not met, the loop stops.

Algorithm:

```
count ← 0
remainder ← 72
while remainder ≥ 14
    count ← count + 1
    remainder ← remainder – 14
end while
print count, remainder
```

Desk check:

```
A ← 1.3
while A^3 > 2.01 or A^3 < 1.99
    A ← 0.5 × (A + 2/A^2)
    print A, A^3
end while
```

Desk check:

Your turn: Have a go at desk checking some of the other algorithms!

```
define factorial(n):
    product ← 1
    for i from 1 to n
        product ← product × i
    end for
    return product
```

```
sum ← 0
for i from 1 to 10
    sum ← sum + 1 / factorial(i)
end for
print sum
```

Run for when a = 1 and b = 3

```
input a, b
while a + b < 20
    b ← b + 2a
    a ← a + 2
end while
print a, b
```

```
sum ← 0
for i from 1 to 5
    sum ← sum + i
end for
print sum
```

Linking to Pedagogy

Identifying/ Describing Code

More Desk Checks/ Tracing

- Intro simple, small snippet to explore and predict
- Then add a little explicit info, try again
- Then add complexity
- Reduce cognitive load of code comprehension – Syntax load (Donaldson and Cutts, 2018)

Linking to Pedagogy

Identifying/ Describing Code

More Desk Checks/ Tracing

- Starts to really shine when working with pseudocode and more variables.
- Gives a structured way to just take the code one step at a time – aiding with dispositions.
- Easy for you as the teacher to see their thinking as well!

Linking to Curriculum

Goal moving into Year 10 and VCE: Move into Pseudocode

Still using the same concepts as earlier years – just more granular now!

Level 10 (algebra)	Methods U1/2	Methods U3/4
VCM2M10A06 **Implement** algorithms that use data structures using pseudocode or a general-purpose programming language.	(snippet is from Specialist, since it's clearer, but intent is the same) The fundamental constructs needed to describe algorithms: sequence, decision (selection, choice, if … then … blocks) and repetition (iteration and loops) Construction and implementation of basic algorithms incorporating the fundamental constructs using pseudocode.	Same descriptions as Units 1/2. Apply ideas to more complex algorithms, often with the following content: • Newton's Method • Trapezium method for area under a curve • Bisection Method
Level 10A (algebra)		
VC2M10AA02 **Devise** and use algorithms and simulations to solve mathematical problems.		

Tripping points along the way

Sequencing of complexity of problems
– need to manage this carefully with fragile dispositions and confidence!

Loops – WHILE

Repeats the code inside the loop while a condition is met.
Once the condition is not met, the loop stops.

Algorithm:

```
count ← 0
remainder ← 72
while remainder ≥ 14
    count ← count + 1
    remainder ← remainder – 14
end while
print count, remainder
```

Desk check:

Desk check:

```
A ← 3
while A² > 11 + 10⁻⁴ or A² < 11 – 10⁻⁴
    A ← 0.5 × (A + 11/A)
    print A, A²
end while
```

Added this in way too soon! Cognitive load was already full with the idea of a while loop, then the scientific notation added

Tripping points along the way

Sequencing of complexity of problems
– need to manage this carefully with fragile dispositions and confidence!

This loop within a loop happened too early in the problem set – conceptually much trickier.

Prompted great convos about desk checking with tables vs. lists though!

Next Steps – Other Pedagogical Approaches

Code Highlighting and Description

(1) Underline the variable names and draw a box and label any input, process or output sections	(2) Describe what happens when particular sections of the program are executed

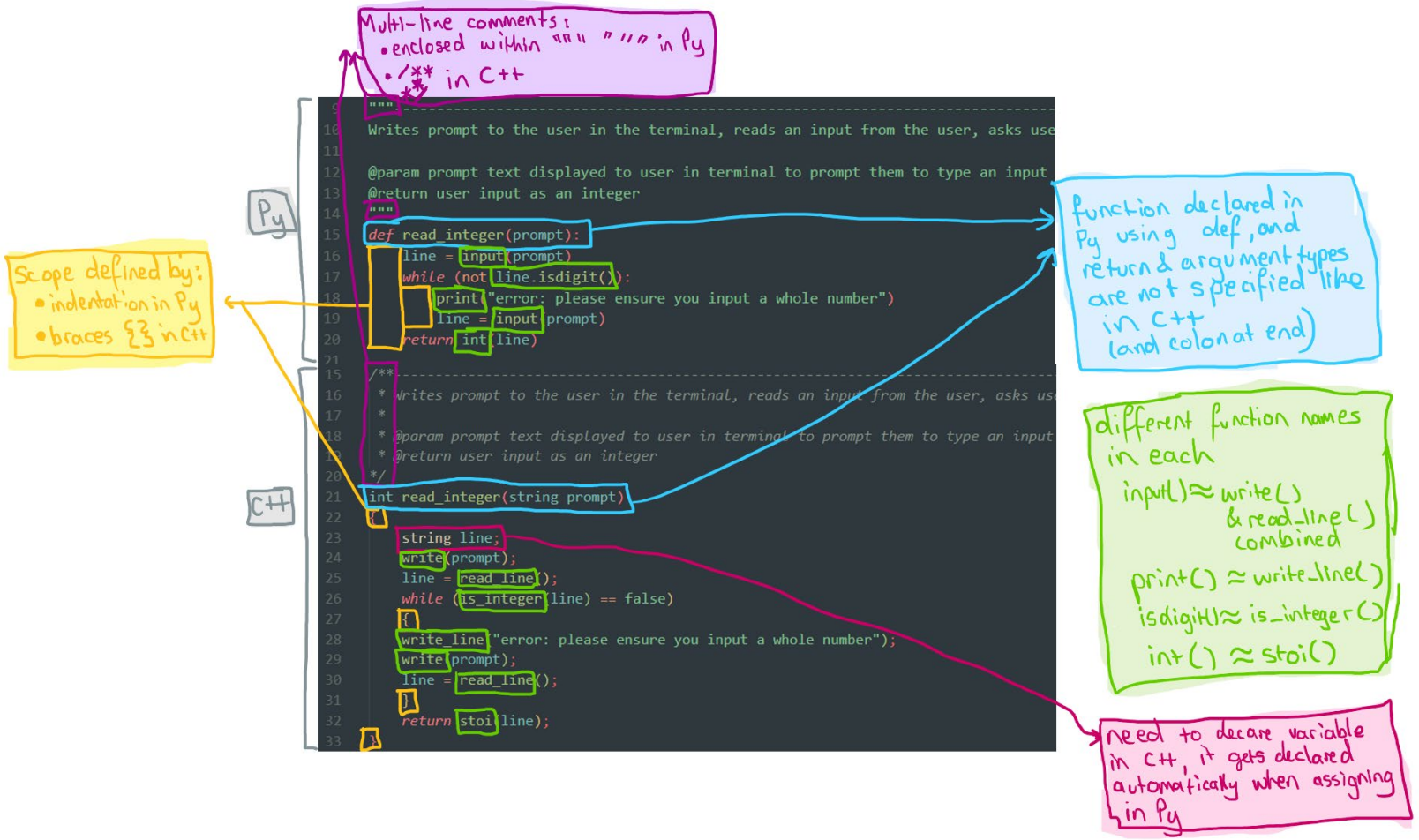
inputs
`height = input ( "What is your height in metres?" )`
`weight = input ( "What is your weight in kg?" )`
	A prompt is displayed to ask the users height and weight and the results are stored in two variables that are created to store the floating point and integer values.

process `bmi = weight / ( height ** 2 )`	The current value of height is fetched from the computers memory, squared and the result is used to divide the current value of weight. The result is stored in a new variable in the computers memory called bmi
output `print ( " Your BMI is " + str( bmi ) )`	The current value of bmi is fetched from the computers memory, converted into a string value and then joined with the string value to create a new string which is displayed on the screen to the user

(Donaldson and Cutts, 2018)

Next Steps – Other Pedagogical Approaches

Code Highlighting and Description



(from one of my assignments at Uni)

Next Steps – Other Pedagogical Approaches

PRIMM – an approach to scaffold towards writing code

- **Predict** – given a working program, what do you think it will do? (at a high level of abstraction)
- **Run** – run it and test your prediction
- **Investigate (Explain)** – get into the nitty gritty. What does each line of code mean? (low level of abstraction). I'm not sure that explain is quite the right term.
- **Modify** – edit the program to make it do different things (high and low levels of abstraction)
- **Make/Create/Design** – design a new program that uses the same nitty gritty but that solves a new problem. I had always called this Create, and it is certainly creative but reading Tedre & Denning's recent paper caused me to change this to Design, as a key computational thinking skill.

(Sentence, 2017)

Next Steps – Bonus Desk Check Examples

```
START
SET a = 2
SET sum = 0
WHILE a < 5
    sum = sum + a
    a = a + 1
END WHILE
PRINT sum
END
```

while loop repeats the two steps until a is 5

Hand Execution of algorithm

a: 2 3 4 5

sum: 0 2 5 9

Screen:

9

8 Consider this pseudocode which sorts a list of five numbers from smallest to largest:

1	START
2	SET list = [8, 5, 10, 2, 7]
3	SET $a = 0$
4	FOR n FROM 1 TO 4
5	a = list element at position n
6	FOR m FROM $(n + 1)$ TO 5
7	IF list element at position $m < a$
8	a = list element at position m
9	END IF
10	END FOR
11	SWAP a WITH list element at position n
12	PRINT list
13	END FOR
14	END

OUTPUT:
2, 5, 10, 8, 7
2, 5, 7, 8, 10
2, 5, 7, 8, 10
2, 5, 7, 8, 10

element at m < a?

list: [8,5,10,2,7] [2,5,10,8,7] [2,5,10,8,7]

a: 0 8 8 2 5

n: 1 2

m: 2 3 4 5 5 4 5

gave up after second round

Next Steps – Bonus Engage/ Warm-up

Engage

The Sieve of Eratosthenes - A famous algorithm!

Step 1 Create a list of all natural numbers from 1 to n.

Step 2 Cross out 1 and let p be 2 (the first prime).

Step 3 Circle p and then cross out all the other multiples of p.

Step 4 Let p be the smallest number in the list that has not been marked.

Step 5 If $p^2 \leq n$, then repeat from Step 3.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

☐ Follow the algorithm below one step at a time

? What does this algorithm do?

Next Steps – Bonus Example from 3/4 Methods (Newton's Method)

Engage

With a partner, discuss and annotate

```
count ← 0
remainder ← 72
while remainder ≥ 14
    count ← count + 1
    remainder ← remainder – 14
end while
print count, remainder
```

This algorithm is written in pseudocode.
It contains a **while loop**, which means the instructions between "while" and "end while" repeat until the condition is no longer met.

With your partner:

- [ ] Discuss what you think is happening
- [ ] Annotate each step with what you think it means
- [ ] Try to describe overall what the algorithm does
- [ ] Write down the output of this algorithm

As a class, share and discuss your answers.

Processing – Desk Check Newton's Method

The algorithm below shows how to use Newton's Method to solve $-x^3 + 5x^2 - 3x + 4 = 0$ near $x = 4$ (here, the starting "guess" pf $x = 3.8$ is used)

```
define f(x):
  return −x^3 + 5x^2 − 3x + 4

define Df(x):
  return −3x^2 + 10x − 3

x ← 3.8
while f(x) > 10^−6 or f(x) < −10^−6
    x ← x − f(x)/Df(x)
    print x, f(x)
end while
```

Use your CAS to help with the calculations here.
The first four lines basically just define the function and derivative function 😊
Desk check and fill in the table below using this algorithm:

	x	$f(x)$
Initial values		
Iteration 1		
Iteration 2		
Iteration 3		
Iteration 4		
Iteration 5		

Why did we stop after the fifth iteration?

Next Steps – Resources and Links

Resource	Description
VCAA Pseudocode Page https://www.vcaa.vic.edu.au/curriculum/vce/vce-study-designs/Pages/PseudoCode.aspx	Super clear resource on key pseudocode ideas, including VCAA style syntax examples and questions
Programmer's Field Guide https://programmers.guide/	Free online textbook/ resource published in a Uni collab. Fantastic resource if you want to deepen content knowledge. This was my bible for Uni!
Digital Technologies Hub https://www.digitaltechnologieshub.edu.au/plan-and-prepare/scope-and-sequence-f-10/years-7-8/general-purpose-programming/flowcharts-and-pseudocode/	Great bank of resources and ideas for teaching pseudocode, algorithms and coding.

Questions and Discussion

Question:

What would you like to know more about?

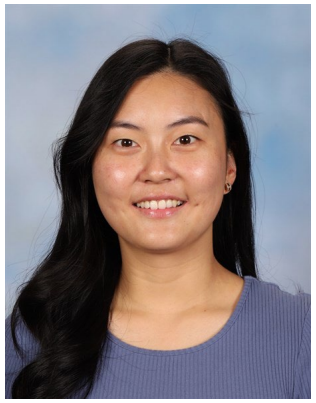
Discuss:

What excites you or engages your curiosity?

How could this work link to your context?

Get in Contact!

We are super happy to send and share any resources and/or collaborate!



Amanda Fan
Head of Cohort
Preston High School

amanda.fan@education.vic.gov.au

Lianna Beeching
Learning Specialist/ Timetabler
Preston High School

lianna.beeching@education.vic.gov.au

THE MATHEMATICAL ASSOCIATION OF VICTORIA

MAV24 CONFERENCE

Event App

App Download Instructions

Step 1: Download the App 'Arinex One' from the App Store or Google Play

App Store

Google Play

Step 2: Enter Event Code: **mav**

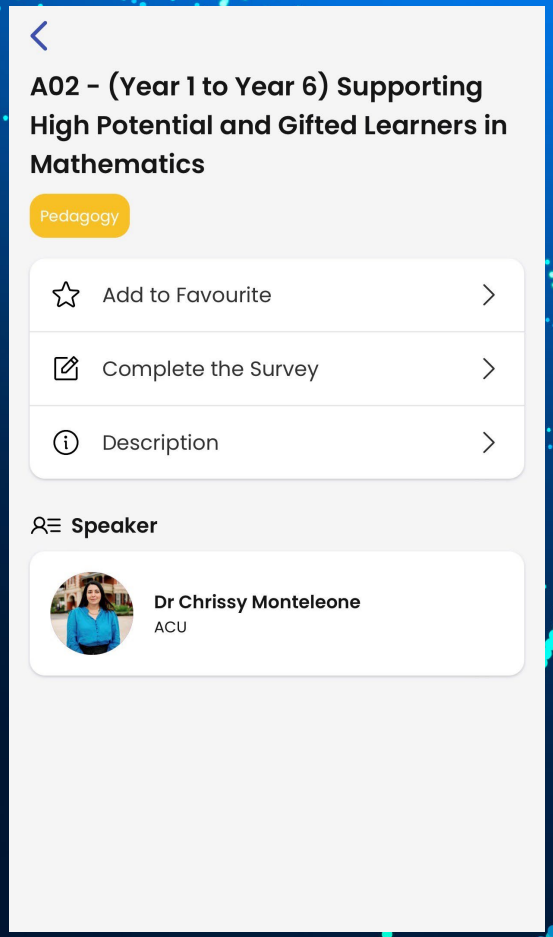
Step 3: Enter the email you registered with

Step 4: Enter the Passcode you receive via email and click 'Verify'. Please be sure to check your Junk Mail for the email, or see the Registration Desk if you require further assistance.

THE MATHEMATICAL ASSOCIATION OF VICTORIA

MAV24 CONFERENCE

Be in it to WIN!



References/ Further Research

Clear, Tony, Jacqueline Whalley, Phil Robbins, Anne Philpott, Anna Eckerdal, Laakso Mikko-Jussi, and Raymond Lister. “Report on the Final BRACElet Workshop:” Journal of Applied Computing and Information Technology 15, no. 1 (June 23, 2011).

Donaldson, Peter, and Quintin Cutts. “Flexible Low-Cost Activities to Develop Novice Code Comprehension Skills in Schools.” In Proceedings of the 13th Workshop in Primary and Secondary Computing Education, 1–4. Potsdam Germany: ACM, 2018. https://doi.org/10.1145/3265757.3265776.

Lee, Irene, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. “Computational Thinking for Youth in Practice.” ACM Inroads 2, no. 1 (February 25, 2011): 32–37. https://doi.org/10.1145/1929887.1929902.

Sentance, Sue. “Exploring Pedagogies for Teaching Programming in School.” Computing Education for All Young People, Wherever They Are in the World (blog), February 20, 2017. https://suesentance.net/2017/02/20/exploring-pedagogies-for-teaching-programming-in-school/.

Rose, Simon. “Bricolage Programming and Problem Solving Ability in Young Children: An Exploratory Study,” August 6, 2016.