Introducing variables to novice programmers

Before we look at ways to introduce aspects of variables, let us examine the range of information a programmer will need to fully understand variables in three simple examples. A simple score that might be used in a quiz or game, a user input word such as a name that could be used to personalise almost any project and a variable that changes within a loop.

Aspects a novice programmer needs to comprehend to understand variables

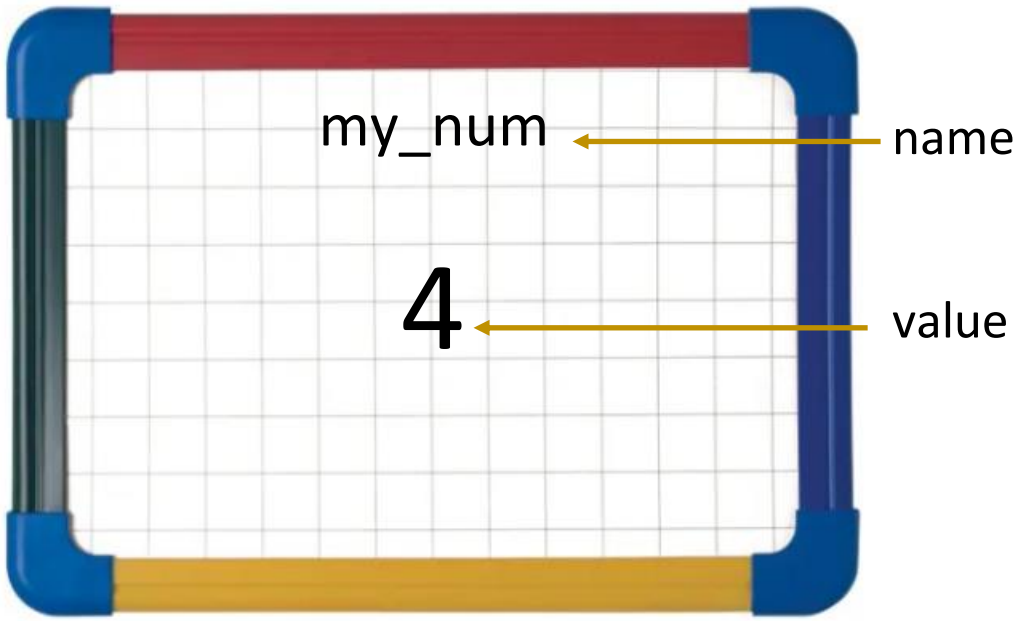
Variable to collect a score	**Variable to hold and use text input by the user**	**Variable iterating through a loop to count**
Variable consists of a name and a value	Variable consists of a name and a value	Variable consists of a name and a value
When a value is linked to a variable, we call it assigning	When a value is linked to a variable, we call it assigning	When a value is linked to a variable, we call it assigning
Variables are named after their role	Variables are named after their role	Variables are named after their role
In most textual languages there are naming conventions	In most textual languages there are naming conventions	In most textual languages there are naming conventions
In many textual languages there are reserved words a programmer can't use when naming variables	In many textual languages there are reserved words a programmer can't use when naming variables	In many textual languages there are reserved words a programmer can't use when naming variables
Values are assigned to variables at the beginning of an algorithm or program. This is called initialisation	Values are assigned to variables at the beginning of an algorithm or program. This is called initialisation	Values are assigned to variables at the beginning of an algorithm or program. This is called initialisation
In some text-based programming languages the type of variable value has to be decided and stuck to	In some text-based programming languages the type of variable value has to be decided and stuck to	In some text-based programming languages the type of variable value has to be decided and stuck to
In many programming languages the programmer has to decide if the variable is local or global	In many programming languages the programmer has to decide if the variable is local or global	In many programming languages the programmer has to decide if the variable is local or global
Number values can be assigned to variables *In most text-based programming languages further divided into floats or integers*	Text values can be assigned to variables. These are strings of characters which include text, digits and symbols	Number values can be assigned to variables *In most text-based programming languages further divided into floats or integers*
Read the name get the value. Variables can be used in place of numbers in any part of an algorithm or program	Read the name get the value. Variables can be used in place of string values in any part of an algorithm or program	Read the name get the value. Variables can be used in place of numbers in any part of an algorithm or program
Number values can be retrieved from variables, combined using mathematical operations and the results can be stored back in variables.	Variables can be assigned by the user through a keyboard input. *Some languages automatically store the most recent keyboard input in a variable called "answer".*	Number values can be retrieved from variables, combined using mathematical operations and the results can be stored back in variables.
		The value stored in a variable can be repeatedly changed each time through a loop

Red are issues related to most text-based programming languages

Purple is an issue with block-based programming languages

Variables consist of a name and a value

Variables have two parts (1) a name which we can use to refer to them in algorithms and programs and (2) a value. In the past I have often referred to this as like a box with a label on the side and objects inside. The concreteness of this analogy appealed to me because pupils can put real objects into a box to increase the value and take away physical objects to decrease the value. However, as soon as you try to move into negative numbers the analogy falls apart. You also inherit other box-like traits such as their ability to accept lots of different types of objects at the same time, where a variable will only accept one type of object at a single time. A better, more universal introductory analogy is the whiteboard.



Variables are like whiteboards

Whiteboards easily display a variety of different types of values such as numbers, strings (numbers, text, punctuation etc) and True and False. Unlike the values put into a box which often couldn't be seen, a whiteboards variable value is clearly visible[1].

When a value is linked to a variable, we call it assigning

When I used the box variable example, I often used lots of box language. Values were **inside** the variable/box, the variable **contained** a number or word. I now use the word 'assign' right from the start. 'Assign' has no unwanted connotations, it implies something linked to something else but that this might not always be the case. A value may be linked ('assigned') to a variable at a given point but at some point, the value might change.

Variables are named after their role in an algorithm or program

Many of my younger users have not really encountered a variable in Maths yet but if they have then their variables will have been letters rather than a description of what it represents. Encouraging pupils to name variables sensibly after the role they take is important. It makes their algorithms and programs much more comprehensible and aids the finding of errors (bugs). For the novice programmer, their language understanding is far in advance of any programming logic understanding so readability is important. So, if a variable is used to collect a user's score then

‘score’, ‘points’ or ‘total’ would be good names. If there are multiple scores, we need to keep track of then ‘player_score’ or ‘users_points’ are good names because they further refine what the variable is being used for. Good habits are also important in naming variables. Text-based programming languages won’t allow multiple words in the name so using examples that link the words helps to encourage good practice. My thanks to Jane Waite for the use of the underscore example. In our normal writing we rarely use an underscore for anything else, so it makes multiple word variables stand out in an algorithm.

Values are assigned to variables at the beginning of an algorithm or program. This is called initialisation

Initialisation is important for all objects in an algorithm or program. Where will the characters be at the start of the game? What value will the score have before the quiz starts? What number will we start with when we start to count? Encouraging novice programmers to think about this in their planning improves their projects and leads to less frustrations in the programming and execution stages. Initialising all variables is essential in many programming languages so teaching this early establishes good practice.

We can assign different types of data to variables

For the novice programmer it is enough for them to know that we can assign numbers or strings (digits/text/punctuation) to a variable. You will notice in my examples above that we only need to know about one of these to create each project. We can reduce the cognitive load by starting with just one type at a time. At some point in their learning journey we want them to understand that variable can have different types of data assigned but only ever one type at a time. Maybe we can cover this when they encounter an example that uses a different type from the introductory example.

Is it best to start with number examples or string example for novices?

In an online discussion on Twitter, Jens Mönig, lead programmer of Snap! (an advanced block-based programming environment) mentioned that they prefer using textual values for novices because of the fear of maths. I think there is value in this reason but have also noticed some younger and less literate pupils prefer number examples because there is less to write. If we choose to assign numbers for our introductory examples let us keep to low values for our demonstrations to keep any maths phobic students onside and to reduce cognitive load and if we choose textual values let us keep writing to a minimum for those who find writing a chore. I am with Jens on this one and think the text examples are slightly simpler, but you will know your class better than we do.

Read the name get the value

It is easy to forget that many novices won’t understand this unless it is explained and demonstrated. When we read an algorithm and we get to a variable we read the name but should act on the value. When a program gets to a variable it also reads the name and acts on the value.

Text example

Assign Florence to variable called best_friend
Say Hello best_friend

When we follow this algorithm we would say

Hello Florence

We read the name but act on the value

Number example

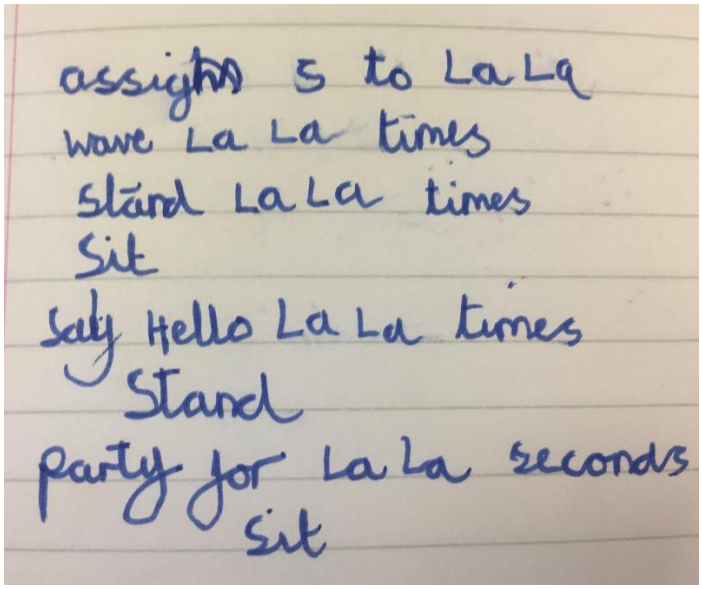
Assign 3 to variable called my_num
Stand up
Bow my_num times
Sit down
Wave my_num times
Say my_num

When we follow this algorithm we would

Stand up
Bow 3 times
Sit down
Wave 3 times
Say 3

We read the name but act on the value

We know that the harder we are forced to think about something then the more it will be likely to make it into our long-term memory, so after acting out the teacher's everyday algorithms with variables a great next step is to write their own. This also helps the teacher to spot any pupils who have faked understanding by copying their classmates' actions earlier.



Year 4 pupil creates an everyday algorithm with a variable called LaLa

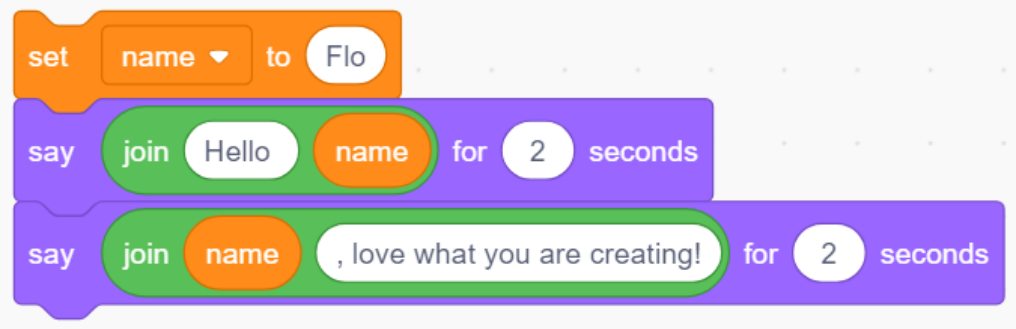
Well done if you spotted that the name doesn't follow the naming convention. Spotting something like this can be a great opportunity for a brief next-steps to improve their work conversation.

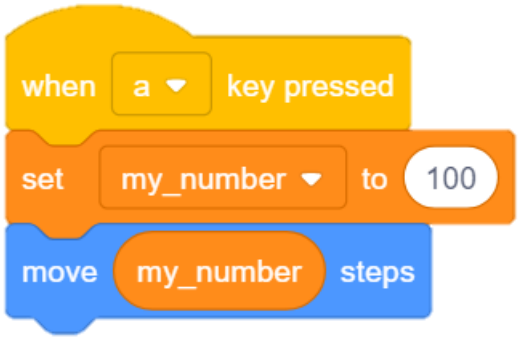
Pause to consolidate and explore

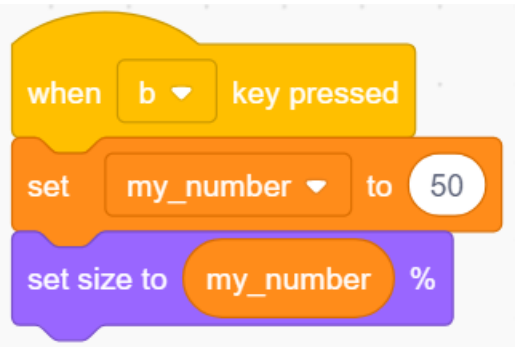
Having worked with many primary classes introducing variables in many contexts it can be tempting at this point to plough on and cover changing a variable using maths or a loop or user input into a

variable. In my opinion, exploring the basics inside a programming language like Scratch or Python is often more useful even if it is only for a short period of time. Pupils give the everyday action algorithms a real context and application.

An easy first step uses text assigned variables in say or think commands or exploring all the ways you can replace number values using a variable.







These are great opportunities for formative assessment questions. What is the value of your variable? What have you named your variable? Why did you name it that? When the program gets to that variable what will it read and what will it act on? Why have you chosen to initialise the variable with that value?

Some variables can be changed using mathematical operations at any stage in the algorithm or program

Fortunately, the very name ‘variable’ implies ‘changeable’ which gives you a chance to introduce how some variables can be constant (not changing) and others can be variable.

The simplest introduction, that I have discovered so far, involves roleplay using everyday algorithms whilst tracking the changes using the whiteboard variable.

Assign 5 to variable called my_num
Stand up
Bow my_num times
Subtract 2 from my_num
Say my_num
Sit down
Wave my_num times
Add 1 to my_num
Say my_num

my_num
5

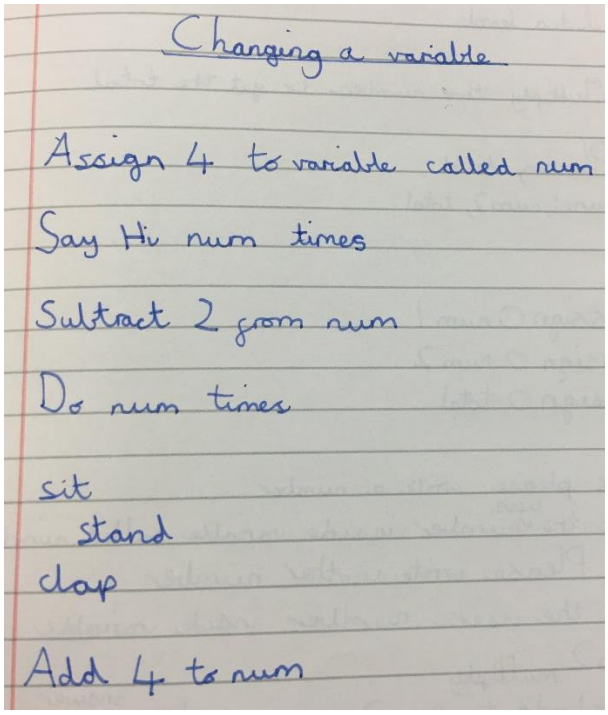
my_num
3

my_num
4

When we follow this algorithm we would

Assign 5 to variable called my_num
Stand up
Bow 5 times
Subtract 2 from my_num
Say 3
Sit down
Wave 3 times
Add 1 to my_num
Say 4

Pupils enjoy writing their own examples and challenging their neighbours to act them out.



This is great for introducing the basic concept on changing a variable using maths, but it doesn't encourage pupils to think about variables that change in real contexts. Shuchi Grover[2], a computing education research scientist from Stanford University has worked with collaborators (Nicholas Jackiw & Patrik Lundh of SRI) for some great research results exploring variables in a pictorial story context

with middle school pupils before coding in Scratch. Pictures on the left include a display of drinks which a customer is purchasing, T-shirts for different ages, weather patterns over a day and a basketball match. The questions prompt thought and discussion before planning how to use these in a program.

With a partner fill out the following table about variables in these stories

Story:	Describe a specific element or quantity in the story that is changing	What would be a good, meaningful name for the variable?	What are some of the ***specific values*** of the variable within the story?	How would you describe ***all possible values*** this variable might take?
1 				
2 				
3 				
4 				

Taken from Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school Shuchi Grover et al 2019 with permission.

I adapted this idea to use with my year (9-10) year old pupils after we had worked with variables in a more supported environment, see Parsons problems and modelling the four levels of abstraction below. In pairs they had to answer the top four questions for three stories before choosing one to convert into an algorithm and then code.

My mum puts £20 into my bank account every month. I bought a new plastic cover for my phone which cost £3.99 yesterday and today I paid for a bus ride home which cost £2.

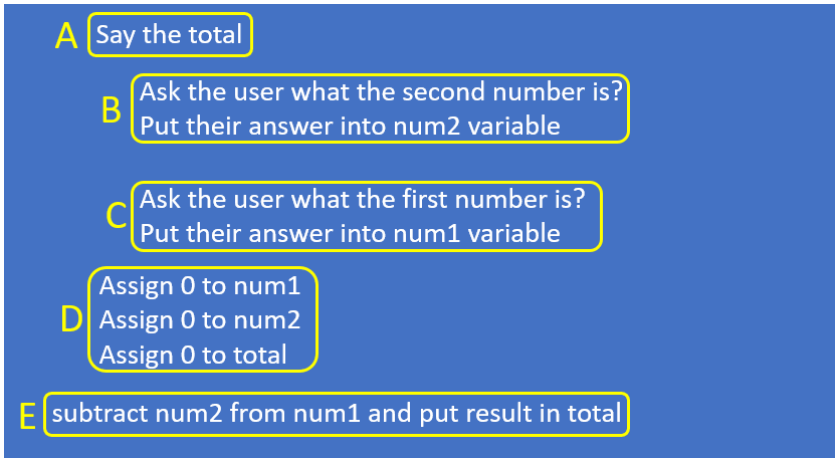
What is changing in the story?	What might you call the variable or variables?	Can you say what value any variable might have at any **specific** time?	What would the variable or variables start at (initialisation)?
The money they have in their bank account	money in bank account · take 1 · take 2 · total · result	£20 = bank account (beggining) £16.1 = total (middle) £14.1 = result (end)	all start on zero.

Thinking about the concept before programming using a story was really effective and something I will definitely use again. You can see the ones they turned into projects here. My improved planner can be found here.

Other useful strategies to increase pupils variable understanding

Parsons problems

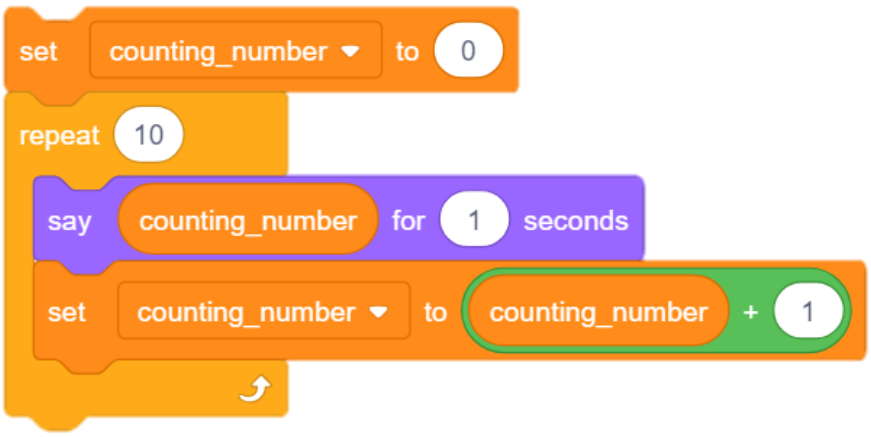
A Parsons problem is where code or algorithm is given to the pupil, but they have to rearrange it into the right order. We used these early on whilst pupils were still coming to grips with assigning values to variables and interacting with multiple variables.



Parsons problems can also be used within block based programming languages, asking pupils to rearrange completed sections of code to complete a project.

Use, Modify, Create

Alongside Parsons problems, examples that pupils can use and modify help to increase understanding. A few good questions can also focus their learning on some of the key issues.



You may notice in this counting program that we have chosen to avoid using the change variable by Scratch block. As we used all four operations it was easier to teach all of them using varable = variable which is much closer to later text-based programming examples such as Python.

PRIMM

Primm is a variation on *Use, modify, create* developed by Sue Sentence, chief learning officer for the Raspberry Pi foundation. PRIMM stands for Predict, run, investigate, modify, make. Pupils are encouraged to think deeply about the programming and **predict** what it might do before **run**ning it. **Investigation** might involve lots of activities designed to maximise pupils understanding. These could include

'trace through the code, comment the code, answer questions about it, label particular concepts, highlight it, draw the flow of control, etc.'[3]

Although designed for secondary pupils it can be adapted and used with primary pupils. You can find out more about it here.

Modelling the four levels of abstraction

The four levels of abstraction are the stages we go through to think through and create a programming project. Although superficially a top to bottom process from idea to execution each level can inform and change levels above as well as below. For example, a programmer spots how a particular code effect works when it is executed which encourages them to adapt their design or even modify their idea. Jane Waite describes these stages for primary pupils in her important research[4].

We used these levels when modelling writing a program to calculate the perimeter of an equilateral triangle.

Idea Level

We started with a really complete idea full of lots of useful information.

Create a program to calculate the perimeter of an equilateral triangle, where all three sides are the same length, by asking the user to input the length of one side and multiplying that length by three.

Planning Level

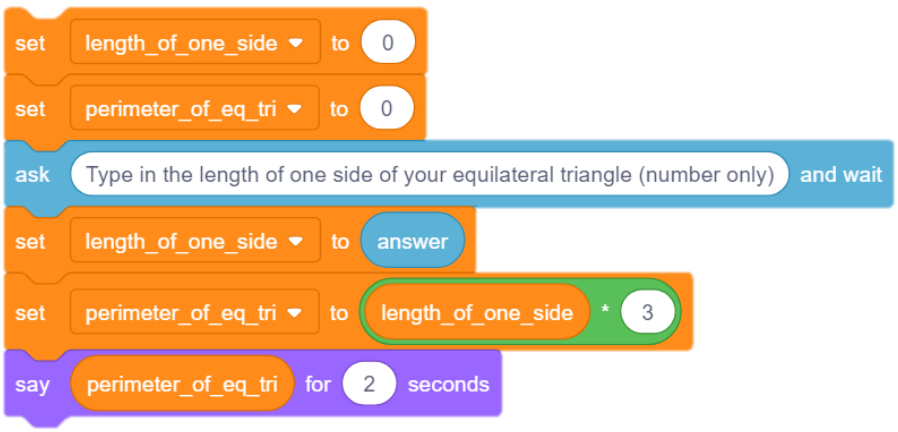
In the planning level we identified the key variables such as the **perimeter_of_eq_tri** and the **length_of_one_side** as well as the maths we would need to create to calculate the perimeter of an equilateral triangle **length_of_one_side x by 3**.
We also decided to initialise both variables to 0

We then wrote this algorithm

Assign 0 to length_of_one_side
Assign 0 to perimeter_of_eq_tri
Ask the user what the length of one side of their equilateral triangle is?
Assign their answer to length_on_one_side variable
Multiply_length_of_one_side by 3 and assign result to perimeter_of_eq_tri
Say the perimeter_of_eq_tri

Programming level

In this we demonstrated turning each line of the algorithm into Scratch code.



Execution level

Finally, we tested and debugged it

Next steps

Pupils then created a similar program to calculate the perimeter of a regular 2d shape of their choosing. Pupils subsequent work definitely benefitted from seeing the process modelled. Making them aware that they would need to observe closely during the process as they would be building a similar program themselves kept them very focussed. Over subsequent weeks this support was gradually reduced until pupils were creating all parts of the process themselves. This gradual fading of support is an important principle of cognitive load theory which you can read more about here. My variables cheat sheet with algorithm and coding examples has also been useful for pupils as a reference when reducing scaffolding.

Conclusion

In conclusion, the opening table outlines the amount of information pupils need to know to fully understand how variables work. If you are using block-based programming, then you can clearly see how much less pupils need to know by looking at all the additional red text needed to understand variables in text-based programming. Shuchi Grover et al.'s (2019) paper is called 'Concepts before coding' and it is an important principle that my own informal findings teaching programming to primary pupils bear witness to. Leaving the programming process to learn about and explore an idea can seem like a loss of precious programming time but as it develops deeper understanding and enables greater agency it is worth it.

Phil Bagge March 2019

References

1 See Article What is in the box Hello World 7th Edition by Jane Waite, Felienne Hermans and Efthimia Aivaloglou

2 Shuchi Grover, Nicholas Jackiw & Patrik Lundh (2019): Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school, Computer Science Education, DOI: 10.1080/08993408.2019.1568955

3 Sue Sentence PRIMM

4 Jane Waite, Abstraction in Action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming, 2017 Jane names the levels as shown above but calls the last level running the code