Codeling with Scratch:
An interactive addition quizzer

A look at how Scratch 2.0 may be used to offer students challenges in problem solving combined with coding. The emphasis is on how students may learn to create software by thinking mathematically to solve a problem.

Coding is in the air. But so far I have not seen much detail, despite Ministers for Education and other gurus and pundits urging us to include coding in the school curriculum.

In fact, ever since Seymour Papert (famous in his later years for constructionism, a learning theory that married conceptual constructivism with concrete experience) created Logo programming, many schools have been coding using Logo, the educational computer programming software that has the famous ‘turtle’ on the computer screen. Importantly, Papert’s intention was to provide a ‘virtual’ interactive world—he called it Mathland in which the native language was mathematics, with the hope that children would learn to ‘speak mathematics’ in the organic intuitive way they learn to speak their native tongue:

… the idea of talking mathematics to a computer can be generalised to a view of learning mathematics in “Mathland”; that is to say, in a context which is to learning mathematics what living in France is to learning French.
(Papert, 1980, p 6).

I have published books, and articles, and conference papers explaining how to use Logo, or LogoWriter, or MicroWorlds—all variants of Logo programming. About 10 years ago, a new version of Logo programming software was provided free. This is Scratch (also known as BYOB, or Build Your Own Brick). It can be used online, or as a free download at the MIT Scratch website, the current version being Scratch 2.0. (Papert first created Logo at Massachusetts Institute of Technology.)

It is very easy to download Scratch 2.0 and begin coding. But where to start? What to do? Scratch’s Logo programming can be used to learn how to make animated cartoons, play music, investigate geometry, explore logical reasoning, practise arithmetic, play do-it-yourself video games, learn about coordinates, maps and graphs, record and edit sounds, and create interactive learning software.

This article focuses on the final possibility. Of course, the real point is not to create software. (There are apps for that! Why reinvent the wheel?) The point is to learn how to create software. That is, learning by thinking mathematically, which is, essentially, problem-based learning.

As you will see, the article outlines an interactive computational piece of software that is even simpler than the excellent game examples in Woodcock (2016) and Setford and Woodcock (2017). Look carefully at the lines of programming—the coding. Notice the way everyday English words are used as command words, telling the computer what to do: ‘saying’, ‘setting’, ‘asking’, ‘answering’, and ‘assessing’ the software-user’s input or answer.

Importantly, consider the logical processes of one command after another. How does the Scratch program (the coding) work? Notice the way random processes generate numbers, and assign (‘set’) them as values for named variables. Observe the way one (randomly chosen) addend (the variable called ‘firstnumber’) is combined with another to create a total sum—which is to be the correct answer for an addition question. Then consider the way the user’s typed input—the answer to the question (ask)—is assessed. Is the user’s ‘answer’ (one of the command words that is the consequence of that command word ‘ask’) is equal to the assigned (set) total sum, then, logically, …

Papert’s purpose, and mine, is not to provide software that can be used to practise adding numbers—although...
that is clearly possible, or will be. (As I said, there are apps for that!) The purpose is to offer challenges to the user or student: understand the definitions (command words), and the logical connectives (such as if then), and work with named variables ('firstnumber', 'second number', etc.), and the interactive procedure ('say', 'ask', and 'answer').

Constructing computer programs, or using the language of Papert's Mathland, is comparable to constructing a mathematical proof. From these initial assumptions and definitions, and these logical steps of reasoning—supported by creative problem solving—we reach the theorem at the end, or, in the case of a computer program, the end-result of the process—a sequence of coding that does what we want it to do. Problem solved!

Figure 1 shows the exemplar Scratch program, or coding. This is, perhaps, like being given a paragraph of a foreign, but indirectly familiar language. What does it mean? How does the language work? Answering these questions also needs mathematical thinking, formal definitions, logical analysis—problem solving! In short, once Scratch, or Logo programming exist, they belong in the mathematics curriculum. Because, as a programming language that is closely based on simple English command words, and simple logic, Scratch (like the original Logo) is simple to learn, it belongs in the Primary mathematics curriculum. (I will leave as homework the challenge to find learning outcomes or curriculum statement for Information Communication Technology that occur within the Australian Curriculum: Mathematics.)

Let me explain. The following discussion will make more sense if you go to the Scratch web-site, or download Scratch, and open a new Scratch project (or ‘file’)—see for yourself!

In Figure 1, the different lines of colour indicate different types of Logo (in colour when in Scratch) commands. When you look at a Scratch screen you will see—reading visually, left to right—three main sections, or panels:

- a Scratch-cat in the left-hand panel, where the coding is active;
- a colour-coded set of words at the top of the middle panel: a dictionary of programming command words; and
- a larger open panel at the right: this is where you will create any coding.

In the middle panel you will see a tab-entry for Scripts. This provides drop-down menu options for all the Scratch command-words, which are colour-coded according to the different types of commands:

![Scratch drop-down menu.](image)

Reading the whole program in Figure 1, the first dark-brown command is:

![Scratch command.](image)

![An interactive Scratch program that practises addition.](image)
This is one of the Event command-words that are used to start a program. In this case, when the user presses the letter a on the keyboard this initiates the following event. (Notice that at the right of the letter a in the command there is an inverted triangle. Clicking on that triangle reveals other possible keys that could be used, instead of a, to trigger the subsequent event. I chose the letter a to stand for ‘addition’, which is what the whole event is about.)

The next line of the program is one of the medium-blue. Looks commands. It does exactly what it suggests: for two seconds it shows (or ‘says’) the following message, as typed by the programmer in the slot after the command-word ‘say’.

Figure 4. Scratch command: “Hello! I will ask you an addition question. Type your answer please,...”.

The words “Hello! I will ask you …” appear in the left-hand panel of the Scratch screen. This is where the program enacts the successive lines of command in a way that a user can see and react to.

The third line of the program combines the light-brown Control command word ‘set’ with the light-green Operator command ‘pick’. In this case, the line is assigning (setting) a numerical value to a variable called ‘firstnumber’, by picking a random whole-number between 1 and 60.

At this point it is necessary to explain how command words selected in the middle-panel of the Scratch screen are placed in the right-hand panel where we create programs. Click on one of the colour-coded script-options, such as ‘Event’ or ‘Sounds’, and then click on one of the command-words in the corresponding colour-coded drop-down menu-option, and drag that (a typical interactive-computer ‘click and drag’ action) into the right-hand panel of the Scratch screen, and drop it. (This uses the typical click-and-drag computer action activated by a computer-mouse, or a touch-pad.) In this case, a click-and-drag process has placed the light-green Operator, ‘pick random’, that initially looks like this:

Figure 5. ‘Pick random’ command.

The pick random command is dropped inside the medium-brown ‘set’ command that initially looks like this:

Figure 6. ‘firstnumber’.

The white window, initially showing 0 or zero, at the right-hand end of ‘set’ is the place or slot where the ‘pick random’ command is dropped.

When a command being clicked-and-dragged is correctly over a slot, the slot becomes slightly brighter and larger—ready to be dropped into. If you make a mistake, try again. With a little trial and error, clicking-dragging-and-dropping into ‘slots’ is easy.

That variable, ‘firstnumber’ has been created by the programmer by clicking on the dark-brown Script option, Data. Then the programmer has clicked on the option:

Figure 7. ‘Make a variable’ option.

Prompts were then given to type a name for the newly created variable. I deliberately chose the name “firstnumber” because we will be adding a first number to a second number, resulting in a total number. It is a good idea to choose meaningful names for variables. That helps you remember which variable is which, and what each variable is for. Incidentally, it is sensible to click in the little square check-box at the left of the variable’s name-slot, to prevent the variable being displayed in the left-hand section of the screen. Otherwise that could be distracting for students who use the program. Obviously the next two lines of the program assign (that is ‘set’) values for the two further variables being used, the second number being added, and the resulting total of the addition. Note the way the light-green Operator that does the adding initially looks like this:

Figure 8. Light-green Operator.

The two white ‘slots’ or circular holes at the right and left of the + symbol have been filled (by the usual clicking and dragging) with the two variables firstnumber and secondnumber, obtained using the medium-brown Data option for Scripts.

Each of the command-words (sometimes referred to as tiles, or bricks) can connect together, as we are clicking and dragging. How? Notice the small jutting-below ‘tabs’ at the bottom-left of command words, and the small cut-out lug-holes at the top-left of the command words. Later, if you change your mind, and do not want to use a colour-coded command-word, you can temporarily pull apart the program by carefully clicking and dragging the part or parts you want to
remove, dragging down, or out and away, or up and away. With a little practice, how this works will be easy and obvious. If you no longer need a command-word in the right-hand programming panel, you remove it by clicking and dragging it across into the middle panel or column of the screen, and it will disappear when you ‘drop’ it. Again, this process of building, and removing, becomes easy and obvious with practice. It is part of the user-friendliness of Scratch. Don’t be afraid. You will not break anything by doing a little trial and error.

The sixth line in the program is where the interactivity begins. It does more or less what the plain English words suggest.

Figure 9. “What is firstnumber + secondnumber = ?”

It asks the user a question, and waits for the user to make a reply, by typing in a slot that appears at the bottom of the left-hand panel, and then pressing the enter or return key to activate the answering. The numbers or text that are typed in the slot at the bottom of the left-hand panel (while answering an ‘ask’ command) are referred to by the light-blue Sensing command, ‘answer’.

What are those light-green ‘join’ words doing? The command word ‘join’, one of the Operators commands, initially looks like this:

Figure 10. The command word and two slots.

Superficially, and pointlessly, this would literally join together the words ‘hello’ and ‘world’. (Pointless, because you might as well type those two words together.) But the join command is useful when it joins together different kinds of information or programming. In our lengthy ‘ask’ command, the first ‘join’, at the left, joins the text that has been typed in the left-hand slot with another join, which contains the variable ‘firstnumber’ in its left-hand slot, and then a further ‘join’, and … and so on—four nested joins, combining three pieces of typed text and two variables. The result looks like a single question with words and numbers and punctuation. (The ‘join’ command is surprisingly effective!)

The last three lines of the program are, collectively, part of an ‘if-then-else’ command, one of the light-brown Control commands. Successively, as the English words in the lines make clear, these three lines test if the user’s input, referred to as ‘answer’, is equal to the value of the variable ‘totalsum’ (which was set to be the addition of the two variables ‘firstnumber’ and ‘secondnumber’), and if this is the case, then the succession of ‘join’ commands are used to ‘say’ that ‘Yes’, this is the correct answer to the question, or else, to ‘say’ what the correct answer should have been.

At this point, the program ends. By pressing ‘a’ again, a fresh question will be asked, and this can be repeated again and again, indefinitely.

Finally, having explained (as briefly, but as clearly and thoroughly as needed) how this Scratch program works, I can end by suggesting that the program can be expanded, or simplified, in many ways. For example, in lines 3 and 4, the highest number for choosing randomised values for ‘firstnumber’ or for ‘secondnumber’ can be simplified to, perhaps, 20, or 10, or can be changed to a higher value, such as 100. Simply click in the final small white ‘window’ in the ‘pick random’ operator-command, and type the number you want.

A different modification would be to replace the addition-operator used in setting the variable, ‘totalsum’, so that the question being asked is about a subtraction, or a multiplication. That is, use these operator-commands:

The asterisk is used as the computer symbol for the traditional multiplication symbol, ×, to avoid confusion with the alphabet letters x and X. In making these changes you might define alternative variables to ‘totalsum’, naming them, perhaps, difference, or product. Modifying the program so it asks questions about division is also possible! And decimals! And other calculations!

Also, two or three other variables could be created, and assigned initial values of zero: one variable would count the number of correct answers, another would count the errors, and a third would count the number of questions asked. To do this, you would also need to use a different way initiating the successive questions. But perhaps that is another story, or problem…

References and further reading


