Semester 2 2019

What Everyone Should Know About Coding

www.cpl.asn.au/journal

JPL JOURNAL OF PROFESSIONAL LEARNING

Ben Zunica demystifies coding for students and educators alike…

Coding for everyone

Computer coding has been taught in schools around Australia for the last 25 years in one form or another. However, it was mainly the province of senior high school and coding was only chosen by students who had a particular interest in the subject.

The increasing presence of algorithmic and automated technologies in school means that all teachers will need to become more familiar with principles of coding design in order to understand and determine how such technologies are used in schools.

Furthermore, in recent times, this way of offering coding has been turned on its head by curriculum authorities, with coding a compulsory subject for all learners in the junior years of secondary schooling, through the new Stage 4 Technology syllabus (New South Wales Education Standards Authority, 2017).

What is coding?

Computer coding is defined as "a list of step-by-step instructions that get computers to do what you want them to do" (Australian Government Department of Education and Training, 2017). While this definition is true, it is extremely broad and does not explain the heart of what coding is. The reason that it does not explain the heart of coding is that writing the step by step instructions comes at the end of the real cognitive work involved in computer coding.

Below is an outline of the whole process of computer coding:

1. Computer coding is a problem-solving process. The first step in computer coding is to define the problem that you are going to solve. This is an easy process if the problem is well defined, but can be difficult if the problem is ill-defined and/or not easily understood. Thus, the problem or question is a human one and necessary framing of the problem is open to human interpretation.

2. The second step in the coding process is finding patterns that occur that will solve the problem and expressing those patterns as processes in natural language. As an example, in Mathematics we can solve number sequences by finding the pattern.

    2, 4, 6, __ , __
    1, 4, 9, ___ , ___
    6, 28, 496, ___

Finding these patterns are easy in the first two instances, but almost impossible in the last instance (they are called perfect numbers). The coder needs to find patterns in the problem that they are trying to solve and be able to write the patterns as processes if they are going to be able to code the solution on a computer. The patterns that are identified by humans do not always fit each situation perfectly, and, while being correct on most occasions, are open to flaws.

3. The third step is writing the processes using instructions that can be easily converted to computer code using a programming language. Coders normally use pseudocode, which is "like" a programming language but is more natural. It is sometimes called structured English.

4. The last step is coding the pseudocode using instructions that conform to a programming language.

5. It should be noted that steps 3 and 4 can be simplified using block-type programming environments that can write computer code for the user. This simplifies the process of turning processes into computer programs, but, in doing so, sacrifices the freedom of the coder by constraining their ability to modify block structures. Examples of block type coding are Lego EV3, EdWare for Edison and Scratch.

Introducing coding

Well, the good news is that we complete steps 1 and 2 in every-day life. All of us have problems to solve and find processes that we apply to solve them. So in fact you are already an expert! Initially, I would not go near a computer. When I start teaching people to code, I will not touch a computer for two to three weeks. I will start with showing students how to define problems, find patterns and write instructions that are unambiguous.

For me, the first step is to write the processes involved in making a piece of toast. It seems like a very well-defined problem with a simple pattern, but it is not. Each student will interpret the problem differently and go to differing levels of detail. Then I get into the kitchen with a recipe for spaghetti bolognaise. Some students will follow the recipe well and make a delicious meal, while others will not be so successful. Then I question them about why that is, focusing on process and ambiguous instructions that can be interpreted differently. Other factors which can be discussed include cultural experience and familiarity with what these foods are and what the end product is supposed to be.

Once these are done, I then move to simple problems from day-to-day life that are well defined. For example, what should a driver do when approaching traffic lights? When should I take an umbrella with me for the day? What is the five times table? Then move on to problems with differing levels of interpretation, such as when should I watch Netflix? Which subject should I study for first? For all of these I would get the students to write out the answer in natural language.

What to emphasise first

In the initial stages of writing answers to problems using natural language, there are three very important concepts that should be emphasised. They are:

1. The inputs and the outputs. Make a very clear connection between what goes in and what must come out. Having a clear knowledge of these will help you greatly in defining the problem correctly and determining the processes required to create the output. For example, when completing the spaghetti, make explicit that the ingredients are the inputs, the bowl of bolognaise is the output and the recipe steps are the processes. Talk about this for each problem.

2. All of the processes that you look at will conform to three control structures or combinations of them. Processes will be a sequence, such as making a piece of toast, a decision (sometimes called branching), like deciding whether to take an umbrella, or, a repetition of actions, like when writing the five times table, you are always repeating the thought of adding by five.

3. What data do you need to keep? So, in the example of deciding what to study, you need to have the data of an examination timetable in order to make the correct decision on what to study first.

Emphasising these three concepts will make coding on a computer much more simple, as coding involves converting inputs to outputs, the preservation of important data using variables and writing code as sequences, decisions and loops (repetition).

Programming language?

Once you can understand and solve simple problems using processes expressed in natural language, then it is time to start "real coding". Convert one of your problems into pseudocode (this part will be the most challenging for students) and then use that pseudocode to write code in the programming language of your choice. The coding is not as challenging as you may think. Anything from your pseudocode you do not know how to put into the programming language can be googled, and, in the vast majority of cases, an answer to your problem will be available. Most computer coders are not worried about changing programming languages because they know that if they get the problem solving right, then converting it to a programming language is relatively easy.

Conclusion

One of the worst mistakes anyone can make with coding, algorithms and automation is to assume it is all about computers and copying and pasting slabs of code. This is not coding, and ultimately it will lead to frustrating experiences as students realise that they are still unable to code, and teachers may find the outputs of the coding do not solve complex human problems, or worse, create new problems.

A much richer experience is what I have outlined above. In seeking to understand this process, you are well placed to give students tools to solve different kinds of problems, which can ultimately be coded using a programming language where this is appropriate. In addition, we will not become reliant on one programming language or mystified by coding, but have skills that can be transferred to a multitude of different languages and situations.

Whilst engaging in such low technology, initial processes may be less thrilling, less expensive and less magical, if you stay strong and do the intellectual work first, we can all reap the benefits later.

References:

Australian Government. Department of Education and Training. Learning Potential. (2017). *What is Coding?* Retrieved from https://www.learningpotential.gov.au/what-is-coding (December 9, 2018)

NSW Education Standards Authority. (2017). *Technology Mandatory Years 7-8 Syllabus.* Retrieved from https://educationstandards.nsw.edu.au/wps/portal/nesa/k-10/learning-areas/technologies/technology-mandatory (December 9, 2018)

Ben Zunica is currently undertaking PhD studies at Monash University, focusing on IT and Mathematics education. He has taught IT and Mathematics to secondary students in both NSW and Victoria over the last 15 years. Ben has been a Senior Judge Marker, a member of the HSC examination committee for Software Design and Development, and a member of the Board of ICT Educators of NSW.