CASE STUDY

Programming as a mathematical activity

Peter Rowlett, Department of Engineering and Mathematics, Sheffield Hallam University, Sheffield, UK. Email: p.rowlett@shu.ac.uk. https://orcid.org/0000-0003-1917-7458.

Abstract

Programming in undergraduate mathematics is an opportunity to develop various mathematical skills. This paper outlines some topics covered in a second year, optional module 'Programming with Mathematical Applications' that develop mathematical thinking and involve mathematical activities, showing that practical programming can be taught to mathematicians as a mathematical skill.

Keywords: programming, mathematical thinking, applications.

1. Introduction

In a mathematics degree, choice of taught content should support the development of the skills needed by graduate mathematicians. Programming can play a considerable role here: directly, by involving mathematical computation and being related to many graduate jobs including in areas such as data science; and indirectly, being closely related to mathematical skills such as clear thinking, problem-solving and precise communication. This article discusses ways in which a module on programming has been used to develop mathematical skills. First, a description is given of the teaching context. Following this, mathematical teaching and assessment activities are discussed as involving mathematical thinking and mathematical applications.

2. A module 'Programming with Mathematical Applications'

Programming is taught to undergraduate mathematics students via an optional second-year, 20-credit module, 'Programming with Mathematical Applications'. This follows on from a little programming experience as part of a core first-year module, plus considerable variety of formal and informal prior knowledge. In two years of operation, the module has attracted between 20-30 students, who work on laptops around tables in an informal classroom environment. Teaching is mostly via written notes with code examples, motivated by diverse prior knowledge and the fact that students will learn a lot more from running their own code than watching me do so. The self-paced nature is commented on by students as a positive. For students already familiar with the content, a series of challenge problems are offered, as well as the freedom to explore programming interests beyond the taught curriculum.

The module comprises taught content on programming fundamentals and mathematical functionality, graphics, GUIs and data manipulation, presentation of mathematical information on the web, and database queries. The choice of topics is based somewhat around their popularity observed through roles taken by our graduates. Following the directly taught content, students take the lead via individual projects completed partly in class time with staff support via in-class discussion and two scheduled progress meetings.

Assessment is via staged independence. A first piece of coursework asks students to write a series of functions to perform specific tasks, for example "Write a function that takes as its input a number x and returns the value of $\frac{2x}{x-1}$". A second coursework gives a detailed program specification and asks students to write a program to meet this functionality. The task is specified at a higher level than the first coursework, for example "Generate 10 HTML files containing certain pseudo-

randomised mathematical information". Finally, an individual project assignment gives a flexible brief or area of interest and allows the student greater freedom in demonstrating their programming abilities. Students were asked to submit a program (language not specified) along with a HTML page which contained: an explanation of the mathematical statistical or technical content; a description of how the program works; and, a critical review of the work.

3. Programming as mathematical thinking

3.1. Programming vs. coding

There are lots of program constructs that are common to many programming languages, such as if statements, loops and functions. I define *programming* as the process of using program constructs to express an algorithm in a way that a computer can interpret and act on. *Coding*, by contrast, is the business of writing a program in the syntax of a particular language.

Programming, requiring clear, systematic thinking and unambiguous communication, is a skill closely linked to mathematics and the practice of the professional mathematician. In teaching mathematics students, it seems appropriate to focus principally on programming. In order to do so, though, it is necessary to write code in a particular language. I tell students that the fact we are coding in a particular language is secondary; the important thing is that they are learning how to program. This also has the advantage that learning to understand program constructs and the process of combining these to implement an algorithm is a skill that can be readily transferred to other programming languages.

I tell my students syntax errors are essentially inconsequential, in order to emphasise programming skills over coding. Rote learning of specific syntax is not a worthwhile goal of programming teaching. People who know several languages might sometimes use the syntax from one in another, particularly if they are using a language they haven't used for a while. This is a minor issue, easily overcome. I tell students they should not feel they are failing as programmers if they have to search for help with syntax. I say that learning how a for loop works and what it can be used for is important, but remembering the precise syntax to make one happen in a particular language is secondary. I also say that if they are regularly working in a particular language, they will find they start to memorise syntax for that language much more easily. If they stop using a language for a while, they will become rusty on syntax, but the idea is that their understanding of the programming fundamentals will persist.

3.2. Teaching directed to programming that isn't coding

I deliberately open the first class in my programming module with a piece of teaching that does not use computers. We are in a standard teaching room with laptops available at discretion of the lecturer, so telling students they are not getting laptops out is a stunt, designed to focus attention on programming skills beyond writing code. (You might feel it is similarly a stunt to have the single reference in an article about programming be from 1896, or indeed to wait until two thirds of the way through the article before naming the language in which students are coding.)

I give students a paper worksheet on propositional logic that opens with this statement:

> Computers do what you tell them to do, not what you really *meant* them to do, so when programming it helps to think strictly logically. This worksheet is designed to refresh some relevant concepts and encourage you to think clearly and carefully.

The worksheet covers propositions and predicates, NOT, AND, OR, XOR and truth tables, and contains some exercises designed to emphasise clear communication, examples of which are given below.

Students are asked to identify propositions – statements that have a truth value – such as “Two plus two equals four” and “The moon is made of green cheese”, from that list that includes statements that are not propositions, such as “Would you like a cup of tea?”

Students are asked to think about the difference between OR and XOR (exclusive OR) in some natural language sentences, such as “Did you see Claire or Alex earlier?” and “Is the door open or closed?” An example exercise in thinking and communicating clearly is given in figure 1(a). In this, social convention dictates that the first ‘or’ is an XOR, while the second is an OR. There is room for interesting discussion here, because while the waiter may intend the first as an XOR, it isn’t strictly the case because if the customer answered “both” to the first question, they would presumably be served both (by the principle ‘the customer is always right’).

Another truth table activity relates to implication. Say p is “$5 = 7$” and q is “you get a first class degree”. Students are asked to consider $p \Rightarrow q$ using a truth table. Then they are asked to resolve some sorites by Lewis Carroll, which are a series of statements which can be linked to form a single chain of predicates. An example is given in figure 1(b) (taken from Carroll, 1896, p. 112). This is designed to support clear thinking and lead into the right kind of thinking for programming logic using if statements.

(a) Consider the following conversation.

```
WAITER: Would you like tea or coffee?

CUSTOMER: Tea, please.

WAITER: Would you like milk or sugar?

CUSTOMER: Both, please.
```

Which type of ‘or’, OR or XOR, is being used by the waiter in each of the two questions?

(b) Write each of the following three statements as an implication.

(i) All babies are illogical.

(ii) Nobody is despised who can manage a crocodile.

(iii) Illogical persons are despised.

Arrange these implications into a chain so that you reach a consistent conclusion.

Figure 1. Sample exercises targeted at precise communication and clear thinking.

Later in the first class, after the students have spent some time getting Python up and running and writing a ‘Hello, World!’ program, another worksheet is based around algorithms. I tell students algorithms are a sequence of precise instructions. In an electives choice session, where students are presented with details about each optional module in order to choose which to study, I show a picture of some cakes and tell them these were produced by my son and me by following an algorithm. In class, I ask students to write algorithms for every day and mathematical activities, such

as “putting on a t-shirt”, “making a cup of tea” and “differentiating the product of two functions”. I then ask them to show their algorithm to someone else, with the instruction:

> When you are shown someone else’s, try to be as pedantic as possible and wilfully misinterpret the instructions. The point is to think about how to really tightly specify a sequence of actions.

This is a fun activity, and a little silly. Some students are reluctant, but I believe benefit from being pushed to participate. A reluctant student might simply say “put your arms in the arm holes and your head through the head hole”. A response might be “but now the t-shirt is inside out or back-to-front!” A particularly witty student I observed answered this kind of loosely-specified algorithm with the question “isn’t it uncomfortable to be wearing a t-shirt that has a coat hanger inside?”

4. Programming using mathematical applications

As well as embedding precise and mathematical thinking in activities, there are plenty of opportunities to make use of mathematical applications in programming exercises.

Some example mathematical activities used in examples and exercises while learning and practising basic programming:

- calculating whether a given year is a leap year (if statements);
- generating terms of the Fibonacci sequence (for loop);
- prime factorisation (while loop);
- computing factorials (recursion);
- asking the user to input a number with some property (user input);
- import data from a CSV file into a spreadsheet (file access);
- generating charts from data using a GUI interface (graphics).

Further mathematical and statistical functionality is included via Python modules, and programming skills can be practised while using these, for example:

- math: mathematical functions, e.g. log, sin;
- cmath: mathematical functions for complex numbers;
- numpy: support for matrices and related mathematical functions;
- scipy: scientific computing, e.g. ODE solvers, linear algebra, etc.;
- matplotlib: 2D and 3D plotting;
- pandas: data analysis tools;
- sympy: symbolic computation.

It is possible to include mathematical applications when testing programming skills in slightly more complicated tasks, for example:

- simple operations: convert a complex number from Cartesian to polar coordinates.
- statistical methods: hypothesis testing.
- numerical methods: find the eigenvalues of a given matrix.
- brute-force number theory: testing for certain properties of a number, e.g. whether it is abundant.
- simulation: simulate flipping a coin 1000 times and report how many heads occurred and what was the longest chain of heads in a row.

Project-based learning, involving more in-depth, open-ended tasks, is an opportunity to include more advanced mathematical activities while assessing programming skills. Students are given a choice of programming language for this task. Most choose either Python or VBA, which are taught in the module, with a small number choosing alternative software such as Matlab. Some example project areas are given below.

- Exploring computer functionality, e.g. investigation of how numbers are represented in computers and the resultant errors that occur; methods for generating pseudo-random numbers; manipulation and calculation of dates.
- Numerical or computational methods, e.g. showcasing linear algebra or ODE solver methods, statistical methods or data analysis tools.
- Simulation/modelling, e.g. cellular automata, scheduling a sports league or tournament; iterative prisoner's dilemma; programming a simple combinatorial game.
- Mathematical investigation, e.g. fractals; representations of 3D objects; machine learning, e.g. via artificial neural networks or generic algorithms.
- Historical investigation, e.g. implementing Ada Lovelace's program for computing Bernoulli numbers; investigating methods for approximating π.

5. Discussion

There are many opportunities not explored here. Partly, this is because our degree is in general quite computational. For example, students make use of Matlab in mathematical modelling modules and of statistical and data analysis software in statistics teaching. Even so, there is much opportunity to develop the skills of a graduate mathematician through programming, as I hope this article has conveyed.

6. Acknowledgements

This paper is based on a talk given at the 'Programming in the Undergraduate Mathematics Curriculum' workshop, Middlesex University, 27th June 2019.

7. References

Carroll, L., 1896. *Symbolic Logic: Part I Elementary*. London: Macmillan.