1

Introduction

This book assumes that you are committed to learning to program, and want to do well. Most likely, you are taking a programming course in college or university. Perhaps you don't have much experience programming yet, or perhaps you have programmed a fair bit, but now you are interested in how to improve the quality of the programs you write. This book aims to help you learn how to write *good* programs, in any language. It's the book I wish I'd had available to me, nearly forty years ago when I started programming; and the book I wish I could have recommended to my students, especially my first-year undergraduate students, over many years since.

Let us tackle one thing head on. People sometimes talk as though students could be divided into programming sheep and non-programming goats: as though programming ability were innate. My experience over more than twenty-five years of teaching, and most current research, suggests that this is *simply not true*. I have lost count of the number of times I have seen students really struggle to begin with, perhaps even failing their first programming course, but go on to become excellent programmers. It also sometimes happens that people come in feeling very confident, perhaps having more programming experience than most of those around them, and later realise that they had hardly begun to tackle the most interesting challenges in software development.

Some people love programming from the very beginning. These people may have started coding at a young age, and often choose to sit up late into the night doing so. That's great, and if you're one of them, I hope you will benefit from this book. But, full disclosure: I was not one of those people. Indeed, when, as a child, I was first

introduced to programming, I didn't really see the point. I didn't start to spend a lot of time on programming until, in my twenties, I encountered a problem I couldn't solve without writing a program. I learned to program because I had a problem I needed to solve, which I couldn't solve any other way.

Tip

To write good programs, you don't *have to* love programming. Moreover, even people who love programming do not automatically write good programs: everyone has to learn how.

The great thing is that there's a virtuous cycle. The better your programs get, the more fun it is to write them.

Perhaps you think it will take longer to write your programs so that they are good, and wonder whether this is something you want to invest in. Surprisingly, as you will discover through using this book, writing good programs *saves* you time and effort, compared with writing any old programs. If you like, you can spend that time and effort on writing more programs. If not, you can spend it on something else.

This book will not teach you any particular programming language – for that you will need a different book, a course, or an online tutorial, and there are plenty to choose from. This book will help you in the process of learning any programming language, and especially, it will help you to learn to write it *well*, and with deep understanding that you can also transfer to your next language. It covers things that programming courses tend to assume students will pick up by osmosis, but that are often, in practice, stumbling blocks. Unlike books aimed at professionals, which assume you can always express what you want to express in the language, this book will help you develop ways of getting unstuck, unconfused and debugged.

You'll learn to write code that you can understand and modify not only when you are at your cleverest, but also when you're not. This will lower your stress levels. It'll let you be lazy, in the best possible way.

There's a certain satisfaction in doing a thing well, though: that's how the virtuous circle works.

Robert Martin, in his wonderful book for professional software developers, *Clean Code*, talks about programmers needing to have "code sense". Code sense is what lets a seasoned professional tell good code from bad, and, much harder, systematically develop good code. If you are starting out with programming, this is what you need to develop. You won't develop it in a day, a week, or a year, but, by paying attention in the way this book aims to encourage, you will gradually increase your code sense.

1.1 Who Is This Book For?

If you are learning to program, this book is for you.

If you are helping other people to learn to program, this book is for you.

If you are a professional programmer, this book is not intended for you – but you are welcome to read it anyway. Perhaps you would like to recommend it to someone. I'd welcome your comments.

1.2 About the Boxes

We use various kinds of boxes. There are tips, like this one:

Tip

A note on spelling. If you spell in British English, you may expect the spelling "programme" rather than "program". However, by long convention, when we write about computer programs, we use the American spelling. Very, very occasionally this is useful disambiguation; computer science events may involve both programs and a programme. Normally, it's just one of those things you have to know.

Explanations of terms, like this:

Terminology: Coding, programming, software engineering

Coding, *programming* and *software engineering* overlap, and all involve giving a computer instructions. They are in increasing order of sophistication. A software engineer can program, and a programmer can code, but not necessarily the other way round. A coder might only translate precise English instructions into a programming language. A programmer takes responsibility for deciding what to write and when it is good. A software engineer typically works as part of a team, and solves real-world problems with high-quality software.

After absorbing the content of this book, you will be well equipped to progress to software engineering: more on that in Chapter 15.

There are stories, like this:

Story

Some people are hooked by programming from the very first time they meet the idea. I wasn't one of those. I thought it was quite cute that you could do things like writing programs to output a list of prime numbers, but I was never interested in writing video games, which, when I was young, looked like the only other thing you could do with a computer. The first program I really cared about was one I wrote when I was doing my PhD in mathematics. I had a conjecture that I thought was true for all integers n (it was to do with a certain collection of esoteric mathematical structures, the Weyl modules for $GL(2, \mathbb{Q})$). The calculations to check it got too tedious to do by hand after about $n = 5$, though. So I wrote a program to check my conjecture, and was quite easily able to find out that my conjecture was true, at least, for all n up to 10,000. Of course that wasn't a proof, but it gave me confidence to look for a proof, and eventually, I found one.

And, of course, there are examples of programs. Here's one example in Python:

Python example

```
print("Hello, World!")
```

Note that programs are not always complete. For example, Java code has to be inside a method inside a class, but I usually omit the lines that show that, writing

Java example

```
System.out.println("Hello, World!");
```

rather than something like

Java example

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Do not worry if any of the program examples do not immediately make sense to you, but do have a look at them. This book is supposed to support whatever language you are learning – only occasionally are there points that are really specific to one language. You will probably find that you can get the gist of an example in a language you do not know, if you read it in conjunction with the surrounding text. Learning to think beyond the confines of whatever language you are studying at present, and to transfer your skills between languages, is an important part of becoming a good software developer. If you are at the beginning of a programming career, the language you will use most may not even have been invented yet. I have chosen to include examples in Java, Python and Haskell: these are all common languages for early programming courses, and they contrast interestingly with one another, so that between them they allow us to cover a lot of ground.

In order to guide you to further reading, and to information that will help you fit what you are learning into the context of the

programming language you are learning, there are often suggestions for things to put into your favourite search engine, like this:

some language issue *your_language*

1.3 Structure of This Book

The nature of learning to program is that you will improve many skills in parallel; yet the nature of a book is that chapters need topics. I have tried to include many cross-references between the chapters, while leaving you plenty of freedom to dip into the book as and when you wish.

Chapters 1 (you're reading that now) to 3 get us going. Chapter 4 will help you to place the language you are learning in the landscape of all programming languages. Chapters 5 to 11 are the heart of the book; you're likely to flip between these chapters frequently. Chapters 12 and 13 are specifically about how to do well in a programming course; you might skip these entirely if, for example, you are teaching yourself to program. Chapters 14 and 15 are the farewell chapters, setting the scene for what I hope will be your lifetime of writing good programs.

1.4 Acknowledgements

I am very grateful to all of my students, colleagues and friends who have commented on drafts of this book, including the following: Alejandra Amaro Patiño; Paul Anderson; Julian Bradfield; Robin Bradfield; Carina Fiedler; Vashti Galpin; Lilia Georgieva; Jeremy Gibbons; Kris Hildrum; Lu-Shan Lee; James McKinna; Greg Michaelson; Hugh Pumphrey; Don Sannella; Jennifer Tenzer; Tom Ward.

Thanks are also due to all at Cambridge University Press, especially my editor David Tranah, and to the anonymous readers for helpful suggestions.

Of course, all remaining errors are mine. Feedback would be most welcome.

Perdita Stevens
phowto@stevens-bradfield.com