Introduction

This book is an experiment—an exploratory guide that leads readers from their existing foundations in programming to more complex, real-world applications, with a strong emphasis on learning by doing. It builds upward from basic concepts and invites active engagement with programming environments, assuming the use of Large Language Models (LLMs) for feedback, code suggestions, and deeper insight. Some ideas and sections are still taking shape, and the work as a whole should be read as part of an ongoing development rather than a final, finished survey. Even so, the intention is to offer a clear, accessible, and encouraging companion—one that will continue to improve and grow alongside its readers.

1: Foundations of Programming. We begin with an introduction to fundamental programming concepts, including variables, control structures, functions, and data structures. Along the way, we'll explore these topics in greater depth and cover related foundational concepts such as e.g. recursion. To enhance your understanding, you'll be encouraged to use LLMs alongside your learning process. These tools will offer immediate feedback, code suggestions, and explanations, helping you grasp the material more effectively. By the end of this section, you will have built small but functional programs, laying the groundwork for more complex applications.

2: Understanding Virtual Machines. Transition from basic programming into the concept of virtual machines to understand computers at a deeper level. We introduce the role of instruction sets, memory models, and execution flow in virtualised environments. We will use simple examples of virtual machines and gradually introduce more complex ones, linking each to programming concepts previously covered. Many variations will be presented, illustrating how different architectures influence software execution. This chapter also serves as a bridge between programming and low-level computation, reinforcing an understanding of how code translates into machine operations.

3: Development Environment. Writing code is only one part of programming; understanding how to debug, optimise, and test effectively is equally essential. In this part, you'll learn systematic debugging techniques to identify and resolve issues efficiently. We will explore optimisation strategies to enhance code performance, considering both algorithmic improvements and touch on hardware-aware optimisations. Lastly, we'll cover the fundamentals of testing, including unit testing, integration testing, and the use of automated testing tools to ensure code reliability and robustness. Alongside practical exercises, this section encourages experimentation with profiling tools to analyse and improve program efficiency.

4: Building and Experimenting. Here, we encourage you to modify and experiment with the provided examples on real hardware, such as the Raspberry Pi Pico. This section is highly interactive, with a focus on hands-on experimentation and project-based learning. We explore how embedded systems operate, how to interface with sensors, and how to control external components. But we will also view common abstractions in a practical way, e.g., client/server relationship or encryption. This part includes project ideas for readers to explore, ranging from simple experiments to more advanced applications. Use the accompanying workbook to extend the examples, modify them to suit new purposes, or apply them to solve specific problems.

5: The Compiler Pipeline. Compilers are often indispensable to programming, translating high-level code into machine-executable instructions. We introduce the stages of compilation, including lexical analysis, parsing, optimisation, and code generation, with practical examples. We show how virtual machines conceptualised earlier can be used to execute compiled code, bridging theory with practical applications. Readers will have the opportunity to construct small interpreters or compilers for custom languages, reinforcing the mechanics of language processing and the role of abstraction in software development.

6: Philosophy and Methodology. Programming involves personal style and philosophy. We explore the 'craftsperson' mindset, focusing on clear, maintainable, and efficient code while balancing pragmatism with theory. This approach carries risks but encourages continuous improvement. Examples like customer satisfaction and thermostat control illustrate problem-solving techniques. Large Language Models (LLMs) challenge traditional programming roles, while methods such as mock-ups, code reviews, and hypothesis-driven development support a systematic approach to robust software design.

7: The Mechanics. We begin with an introduction to classic general data algorithms, then some mechanisms that underlie program execution, examining how low-level control flow, memory management, and processor instructions shape the behaviour of computing systems. We explore major programming paradigms to show how each offers a distinct way of expressing solutions, some elements in building architectures, design patterns that provide reusable strategies, and touch on systems-level illustrations which demonstrate how these ideas scale into server and distributed applications.

8: Perspectives and Frontiers. The last chapter starts with observing the computing's essence unfolds through systemic principles like abstraction and resilience, while Turing's machine reveals unbreakable limits, proving some problems, like halting, are unsolvable. Information theory quantifies data flow, and AI's evolution—from neural networks to Transformers—brings power and risks, reshaping programming and testing. Hoare logic, property-based testing and formal type systems further ensure reliability, painting a field both boundless and bound by unyielding truths.

Introduction

In the early 1980s, I was optimistic about AI, especially logic programming, but became disillusioned during the AI winter.[1] The internet's rise in the 1990s revolutionised access to information, and I believed AI would eventually transform human-computer interaction. With the advent of Large Language Models (LLMs) in the 2020s, this vision is finally being realised, reshaping how we interact with technology.

Some parts of this book were first drafted more than twenty years ago. My thinking about code, programming, theory, and methods has developed over several decades, shaped by both teaching and professional software development. Much of the discussion on virtual machines, for instance, predates the emergence of large language models (LLMs) and was later made available on GitHub. The release of ChatGPT-4 prompted me to revisit these materials and to consider shaping them into a structured textbook—one intended not only for others but also as a reference for my own continuing study.

This time, I didn't want to follow the usual path of starting with a single

[1]AI winter refers to periods of reduced funding and interest in artificial intelligence research, often caused by unmet expectations and limited progress in AI development. These winters occurred in the 1970s and late 1980s to early 2000s, when initial optimism about AI's potential faced setbacks, leading to disillusionment in both academia and industry.

programming language. Instead, I wanted to approach the bigger question: how do computers actually work, beginning with the idea of virtual machines? With the help of LLMs, this feels much more doable. They can generate code instantly, which makes it easy to explore and test concepts in practice. It becomes possible to reach the theory through the code itself, instead of treating code as something that only follows after the theory.

In the early 1970s, there was significant debate about the use of *calculators* in schools, with concerns that students might lose the ability to perform basic arithmetic. Over time, this perspective evolved, and calculators became widely accepted in classrooms. While mental arithmetic is undoubtedly a valuable skill, one might question whether the time and effort required to master it are justified, given the efficiency and availability of calculators.

Similarly, when learning programming, the use of an *interactive language* or one with a fast compiler provides immediate feedback, allowing for quick iteration and testing. This approach is an incredibly powerful tool for accelerating the learning process. My own introduction to programming in the very early 1980s was through the interactive language BASIC. While BASIC may not have been the most sophisticated language, it served as a practical tool for gaining familiarity with and understanding the coding process.

Reflecting on these experiences highlights *the value of embracing new tools in education.* Learning what a tool replaces, how and when to use it, and understanding its underlying mechanisms becomes an integral part of what coding and programming will evolve into in the future.

Purpose

The book is intended for *individuals with prior experience in programming* who are looking to expand their knowledge by leveraging *Large Language Models* (LLMs) for interactive and personalised learning. Prominent examples of these AI models, as of the time of writing, include Google's Gemini, Meta's LLaMA, Anthropic's Claude, DeepSeek's DeepSeek, xAI's Grok, and OpenAI's ChatGPT. These models significantly enhance the learning experience by offering real-time assistance and guidance, creating a more engaging and responsive educational environment.

The proposed structure of the book follows a project-based learning approach. Each chapter will focus on building small, practical projects that reflect real-world applications. LLMs will be integrated as virtual assistants throughout these projects, offering real-time support to answer questions, clarify concepts, and provide guidance.

- *AI as a learning partner.* AI can play an active role in education by serving as a mentor, providing instant feedback, and generating code snippets.

- *Interactive learning with LLMs.* A programming book for beginners can use LLMs to offer personalised and interactive education.
- *Project-based approach.* The book will focus on hands-on projects, with the assumption of LLMs assisting you throughout the learning process.
- *Interactive languages.* The use of interactive languages, such as Python, highlights the benefits of immediate feedback in learning programming.
- *Value of new tools.* Understanding and embracing new tools in education is crucial for advancing teaching methods and learning experiences.

Prerequisites

You are expected to have some prior experience with programming. In this book, we will focus on three widely recognised programming languages: Python, C, and JavaScript. They are used as starting points here, but naturally not exclusive to learn programming. Each represents a certain approach to programming. JavaScript is particularly accessible, as it can be run directly in web browsers. There's no need to install a dedicated development environment—simply write your code in a text editor, save it, and open it in a browser. Python is widely used, user-friendly, and supported by a vast amount of readily available help. C, though considered an older language, remains highly valuable for its efficiency and simplicity, making it an excellent language for understanding fundamental programming concepts.

Installing Python and C can vary depending on the computer and operating system you are using. Fortunately, there are plenty of resources available online to guide you through the process. You can search for installation guides specific to your operating system, or even use an LLM to assist you with step-by-step instructions and troubleshooting tips.

You are also expected to engage in some hands-on work with hardware, particularly in connection with the Raspberry Pi Pico. While the choice of using the Pico over other platforms, such as Arduino, STM32, or similar microcontrollers, is entirely arbitrary, it is important to note that the Pico offers considerable processing power at a relatively low cost. Additionally, it boasts high availability, making it an excellent option for both beginners and experienced users alike. This versatility allows for a wide range of projects, enhancing your understanding of programming in conjunction with hardware.

The book presents a very steep learning curve, requiring readers to actively engage with LLMs to acquire new and complementary knowledge in all relevant fields.

A friendly warning on LLMs

At the time of writing, LLMs (Large Language Models) have already had a significant influence on society, where they are *expected to disrupt conventional learning approaches and redefine the skills we prioritise*. This emerging technology is anticipated to change not only how we acquire knowledge but also how we apply it in practice.

While LLMs can rapidly generate useful code snippets or explanations, they often lack full context or deeper understanding of specific problems. This is where human intuition and expertise become essential. Programmers must critically evaluate the suggestions, spot potential logical inconsistencies, and decide when to trust or override the LLM's output. Experienced developers are more likely to recognise subtle issues that might not trigger immediate errors but could lead to inefficient or incorrect behaviour down the line. Therefore, while LLMs can be powerful tools for learning and problem-solving, they should complement, not replace, traditional problem-solving skills and thoughtful analysis.

Here are some problems identified by an LLM (ChatGPT 4.o) itself:

- LLMs can provide incorrect or outdated information as they rely on patterns from training data without fact-checking or cross-referencing.
- They may struggle with context, leading to irrelevant responses or misinterpreting student questions.
- LLMs can reflect biases from their training data, raising concerns about fairness and equity in education.
- Over-reliance on LLMs might hinder students' critical thinking and problem-solving skills.
- Privacy and data security concerns arise when using LLMs, as they may require sharing personal information.
- LLMs offer generalised responses, which may not suit individual learning styles or adapt to unique student needs.
- Ethical concerns exist around academic honesty, with LLMs potentially enabling cheating.
- Not all institutions have the resources to implement LLM-based tools, limiting access in under-resourced settings.

While LLMs have the potential to greatly enhance education by providing instant feedback, explanations, and resources, it is crucial to address these challenges to ensure they are used effectively and ethically.

Balancing the use of LLMs with traditional educational methods and human oversight can help mitigate these issues.

Note to the reader

This book is written to explain ideas clearly, not to provide exhaustive technical listings or formal references. The code examples that appear throughout are best viewed as conceptual scaffolding: simplified fragments that illuminate how systems work, rather than complete or production-ready implementations. Details such as input validation, error handling, or platform-specific configuration are often omitted to focus attention on the underlying logic.

In keeping with this goal of clarity, the printed edition does not include references or detailed bibliographies. Instead, a living companion repository on GitHub collects verified source code, references, and additional materials that can evolve alongside the subject matter (QR codes and URLs after each chapter). This arrangement also reflects how contemporary learning increasingly relies on interactive digital tools and large language models (LLMs), which can now provide explanations, elaborations, or practical advice as needed.

Traditional end-of-chapter exercises are replaced by opportunities for project-based exploration. LLMs can serve as assistants and collaborators in these projects, helping to design experiments, clarify concepts, and extend examples. Yet it remains essential to maintain an independent grasp of the ideas themselves: the real value lies in the combination of active reasoning and intelligent assistance. The GitHub repository complements this by providing starting points, references, and updates that bridge explanation and practice.

From Code to Computation and **Workbook**
https://github.com/Feyerabend/bb/tree/main
Scan the QR code to access more information, errata, exercises, code, resources and updates.