# Introduction

*This book is an experiment. It is an exploratory book, designed to guide readers from already acquired fundamentals of programming to complex real-world applications, with an emphasis on learning by doing. It builds upward from basic concepts, encouraging readers to engage with programming environments. And it assumes using Large Language Models (LLMs), which assist you with feedback, code suggestions, and deeper understanding.*

**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: Case Studies.** Real-world applications often provide the best learning experiences. In this section, we examine real-world virtual machines and discuss their importance in software development. We explore case studies of well-known VM implementations, such as those used in Java, and WebAssembly. By analysing

their architectures and trade-offs, readers gain a deeper understanding of how abstract computational models shape modern programming environments. Additionally, we look at how these principles apply to game engines, high-performance computing, and embedded systems.

**8: Advanced and Miscellaneous Topics.** This final section delves into more complex and specialised topics, expanding upon prior knowledge. We explore advanced data structures such as graphs, trees, and heaps, and discuss their applications in real-world scenarios. Other topics include garbage collection strategies, memory management techniques, and concurrency models. Additionally, we introduce graphical programming, shader development, and real-time rendering, providing a glimpse into computational graphics. This section serves as a gateway to further exploration, equipping readers with the tools to pursue advanced projects and research.

## 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 date back 20 years, originating from a manuscript for a Java programming textbook that was never published (intended it to be an update of an earlier textbook). Much of the content on virtual machines was written long before the advent of LLMs and was subsequently posted on GitHub.

However, with the announcement of ChatGPT 4.o, I reconsidered the prospect of writing a new textbook—one that I would study myself. This time, the book will shift its focus from learning a single programming language, to understanding how computers work starting with virtual machines. With the help of LLMs, this approach is now feasible.

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

[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.

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, DeepSeek 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 on exercises

This material doesn't include traditional exercises. Usually, exercises help guide learning and let readers or teachers check understanding and progress. But

with large language models (LLMs) now available, the focus may shift towards project-based learning. LLMs can assist with projects and support learning, but more importantly, they highlight a new way of learning—one that values active participation, creativity, and deeper understanding instead of repetitive drills. You can find relevant code, a few exercises, starting points, project ideas, and other related info on GitHub.

That said, while LLMs are powerful tools, relying on them too much can hinder the development of critical thinking and problem-solving skills. It's important to use them in balance with independent learning and reasoning.

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