From Basics to Bytecode

From Basics to Bytecode: A Guide to Computers and Programming

Set Lonnert
with AI Collaboration

Adobe and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Raspberry Pi is a trademark of Raspberry Pi Ltd.

All other trademarks are the property of their respective owners.

All images are in the public domain unless otherwise stated.

Cover illustration in: Journal for Manufactures and Household Management, Scheutz, Stockholm, 1825, third issue, between pp. 96–97. Uppsala University Library. https://urn.kb.se/resolve?urn=urn:nbn:se:alvin:portal:record-95006
An illustration of the Jacquard mechanism. A device that uses punched cards to control the lifting of individual warp threads on a loom, making it possible to automate the weaving of intricate patterns with precision.

This work is marked with CC0 1.0 Universal.
To view a copy of this license, visit:
https://creativecommons.org/publicdomain/zero/1.0/

Authors: Set Lonnert with AI Collaboration 2025
Publisher: BoD, Books on Demand, Stockholm, Sweden
Printer: Libri Plureos GmbH, Hamburg, Germany
ISBN: 978-91-XXXX-XXX-X

Contents

Introduction **i**

1 Foundations **1**
1.1 Prerequisites . . . 1
1.2 Foundations of Programming . . . 1
1.3 Simple data types . . . 2
1.3.1 Integers in binary . . . 2
1.3.2 Floating-point numbers . . . 4
1.3.3 Characters and ASCII . . . 4
1.3.4 Strings . . . 5
1.3.5 Representations and types . . . 5
1.3.6 Summary . . . 6
1.4 Variables . . . 7
1.4.1 Assignment . . . 7
1.4.2 Mutable and immutable variables . . . 9
1.4.3 Summary . . . 12
1.5 Control structures . . . 13
1.5.1 Conventional control structures . . . 13
1.5.2 Control structures in computers . . . 17
1.5.3 Summary . . . 18
1.6 Functions . . . 19
1.6.1 Calling functions . . . 20
1.6.2 Summary . . . 23
1.7 Practice . . . 24

2 Understanding VMs **29**
2.1 Simple VMs . . . 29
2.1.1 The stack . . . 30
2.1.2 Interpreter technique . . . 31
2.1.3 VM1 implementation . . . 31

2.1.4 REGVM implementation . . . 33
2.1.5 Portability . . . 36
2.1.6 Summary . . . 38
2.2 Stack-based VM . . . 39
2.2.1 Comparisons . . . 41
2.2.2 Iterations . . . 42
2.2.3 Error handling . . . 43
2.2.4 Summary . . . 45
2.3 Memory and functions . . . 46
2.3.1 Frame pointer . . . 51
2.3.2 Local storage . . . 53
2.3.3 Memory management . . . 54
2.3.4 Frame stack . . . 55
2.3.5 Summary . . . 60
2.4 Practice . . . 62

3 Development Environment 67
3.1 Prerequisites . . . 67
3.2 Basic tools . . . 67
3.3 Debugging . . . 68
3.3.1 Process . . . 70
3.3.2 Tools . . . 71
3.3.3 Summary . . . 78
3.4 Optimisation . . . 78
3.4.1 Memory . . . 83
3.4.2 Time . . . 85
3.4.3 Confusion matrix . . . 87
3.4.4 Summary . . . 93
3.5 Tests and testing . . . 93
3.5.1 Automated testing and continuous integration . . . 97
3.5.2 Summary . . . 100
3.6 Practice . . . 101

4 Building and Experimenting 105
4.1 Prerequisites . . . 105
4.2 The computer as hardware . . . 105
4.2.1 Hardware . . . 108
4.2.2 The Pico . . . 110
4.2.3 Summary . . . 113
4.2.4 Practice . . . 114
4.3 Input/Output . . . 115

4.3.1 The Pico pins . . . 116
4.3.2 Light switching circuit . . . 117
4.3.3 Programmable I/O . . . 118
4.3.4 State machines . . . 121
4.3.5 Circuit for traffic lights . . . 124
4.3.6 Pedestrian crossing . . . 126
4.3.7 Temperature measurements . . . 128
4.3.8 Temperature indicator . . . 129
4.3.9 Summary . . . 132
4.3.10 Practice . . . 134
4.4 Secondary memory . . . 138
4.4.1 Save and load temperatures . . . 140
4.4.2 Summary . . . 140
4.4.3 Practice . . . 140
4.5 External communication . . . 140
4.5.1 UART . . . 140
4.5.2 Connect Two Picos . . . 142
4.5.3 Client-Server . . . 145
4.5.4 Wireless . . . 148
4.6 Security . . . 149
4.6.1 Encryption/Decryption . . . 149
4.6.2 Example . . . 150
4.6.3 Program 1: Encrypting Program . . . 152
4.6.4 Program 2: Decrypting Program . . . 153
4.6.5 Secure communication . . . 155
4.6.6 Data integrity . . . 156
4.6.7 Hashing for Security . . . 157
4.6.8 Symmetric/Asymmetric Encryption . . . 159
4.6.9 Summary . . . 161
4.6.10 Practice . . . 162
4.7 Direct memory access . . . 163
4.7.1 Pico and DMA . . . 164
4.7.2 Pico and PicoDisplay . . . 165
4.7.3 Text . . . 168
4.7.4 Pixel graphics . . . 168
4.7.5 Vector graphics . . . 168
4.7.6 Summary . . . 169
4.7.7 Practice . . . 169
4.8 Combining I/O . . . 169
4.8.1 Simplified web server . . . 169
4.8.2 Summary . . . 169

4.8.3 Practice . . . 169
4.9 Handling errors . . . 169
4.9.1 Types . . . 170
4.9.2 Approaches . . . 170
4.9.3 Summary . . . 173
4.10 Timing and timers . . . 174
4.10.1 Timing and Timers in the Pico . . . 175
4.10.2 Interrupt handling . . . 177
4.10.3 Summary . . . 177
4.10.4 Practice . . . 177
4.11 Concurrency and multithreading . . . 178
4.12 Power management . . . 183
4.13 Practice . . . 188

5 The Compiler Pipeline 189
5.1 Prerequisites . . . 189
5.2 Compilers, Interpreters, Assembler, and VMs . . . 189
5.3 Syntax . . . 192
5.3.1 Tokenisation or Lexical Analysis . . . 193
5.3.2 Parsing . . . 194
5.3.3 Production Rules and Grammars . . . 196
5.3.4 Bottom-up . . . 198
5.3.5 Top-down . . . 204
5.3.6 Summary . . . 208
5.4 Semantics . . . 209
5.5 Intermediate Representations (IR) . . . 210
5.5.1 Abstract Syntax Trees (ASTs) . . . 216
5.6 Semantic Analysis . . . 218
5.6.1 Three-Address Code (TAC) . . . 220
5.6.2 Role of IR in Optimisation . . . 221
5.6.3 Summary . . . 223
5.7 Code Generation Techniques . . . 224
5.8 Loader and Linker . . . 225
5.9 Practice . . . 230

6 Philosophy and Methodology 233
6.1 Philosophy and style . . . 233
6.1.1 Programming as craft . . . 234
6.1.2 Building process . . . 235
6.1.3 Blending philosophies . . . 238
6.1.4 Methodologies . . . 241

6.1.5 References . . . 243
6.1.6 A personal note . . . 245
6.1.7 Risks . . . 247
6.1.8 Summary . . . 248
6.2 Problems and methods . . . 249
6.2.1 Example: Low customer satisfaction . . . 249
6.2.2 Example: Thermostat . . . 250
6.2.3 Generalisations . . . 251
6.2.4 Methodology in the craft philosophy . . . 255
6.3 Reflections on the history and future . . . 257
6.3.1 AI example: LLMs . . . 259
6.3.2 Summary . . . 262
6.4 Method examples . . . 262
6.4.1 Mock-ups and prototypes . . . 263
6.4.2 Code review . . . 267
6.4.3 Null hypothesis in programming . . . 275
6.4.4 Summary . . . 279
6.5 Practice . . . 280

7 Case Studies 281
7.1 JVM . . . 283
7.2 PostScript . . . 283
7.3 Language and Virtual Machine Projects . . . 288
7.4 Systems Programming Projects . . . 288
7.5 Tools and Frameworks . . . 288
7.6 Simulators and Emulators . . . 289
7.7 Data Science and AI Projects . . . 289
7.8 Creative and Visual Projects . . . 289

8 Advanced Topics 291
8.1 Abstract data structures . . . 291
8.2 Object-orientation . . . 291
8.3 Functional languages . . . 291
8.3.1 Design patterns . . . 291
8.4 Design patterns . . . 301
8.4.1 Factory . . . 303
8.4.2 Observer . . . 305
8.4.3 Command . . . 307
8.4.4 Strategy . . . 310
8.4.5 Adapter . . . 312
8.4.6 Composite . . . 315

8.4.7 Decorator 316
8.5 Security 318
8.6 Garbage collection 318
8.7 Memory management 319
8.8 Graphics 319
8.9 Different VMs 319

Register **322**

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 deeply with both small-scale and large-scale 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, and functions. Along the way, we'll explore these topics in greater depth and cover related foundational concepts. 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.

2: Understanding Virtual Machines. Transition from basic programming into the concept of virtual machines to understand computers. We will use simple examples of virtual machines, and gradually introduce more complex virtual machines, linking each one to programming concepts previously covered. Also many variations will be presented, illustrating the differences of computers through virtual machines.

3: Development Environment. In this part, you'll learn systematic debugging techniques to identify and resolve issues effectively. We will explore optimisation strategies to enhance code performance. Lastly, we'll cover the fundamentals of testing, including unit testing, integration testing, and using automated testing tools to ensure code reliability and robustness.

4: Building and Experimenting. Here we encourage you to modify and experiment with the provided examples on real hardware, the Raspberry Pi Pico.

This part includes project ideas for readers to explore. Use the accompanying workbook to extend the examples, or apply them to solve specific problems.

5: The Compiler Pipeline. We introduce compilers, parsing, optimisation, and code generation, with some practical examples. We show how virtual machines conceptually earlier can be used to run compiled code. Also there are references to hands-on projects where you build simple compilers or interpreters for your own small languages.

6: Methods and Tools. There are many ways to approach programming, through style and philosophy of the programmer. Here we make a choice of the 'craftsperson'.

7: Case Studies. Examine real-world virtual machines and discuss their importance in software development. Explore how the concepts you've learned apply to modern programming environments.

8: Advanced Topics. Delve into more advanced topics such as abstract data structures, garbage collection, memory management, and graphics. ...

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

[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.

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 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.