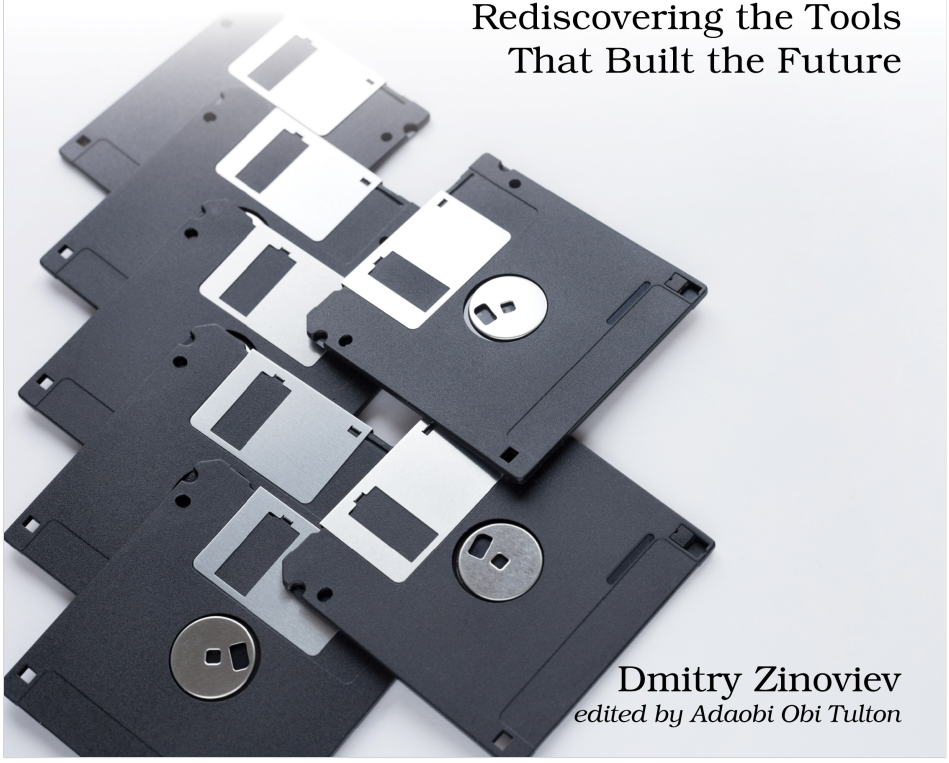
The Pragmatic Programmers

Seven Obscure Languages in Seven Weeks

Rediscovering the Tools That Built the Future



Dmitry Zinoviev
edited by Adaobi Obi Tulton

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Whan that Aprill, with his shoures soote The droghte of March hath perced to the roote And bathed every veyne in swich licour, Of which vertu engendred is the flour.

➤ *Geoffrey Chaucer, English poet, author, and civil servant*

Introduction

The epigraph to this Introduction is taken from *The Canterbury Tales*, written by Geoffrey Chaucer around 1387. At first glance, it is written in English. At a second glance, it is not, for the amount of misspelled words is enormous, and the grammar looks weird, too. Indeed, the *Tales* are composed in Middle English, a language spoken and written in England in the 14th century but gradually transitioned into Modern English from the late 15th century onward, joining the ranks of largely forgotten, and obscure for any practical purpose, *natural languages*, such as Akkadian, Ancient Egyptian, Old Norse, Latin, Aramaic, Manx, and Etruscan.

Natural languages originate (yes, even nowadays: Afrikaans in Dutch South Africa, Modern Hebrew in Israel, Tok Pisin in Papua New Guinea), evolve, and die. Programming languages are similar to them: they go through the same cycle but at a much faster rate. This book is about forgotten, obscure programming languages.

Why would one care about obscure languages? Well, for more than one reason!

- Maintaining Legacy Systems: Much of today's code was written in older languages. Knowing these languages helps maintain and update these systems.
- Problem-Solving Skills: Programming languages are designed to solve particular problems efficiently. You can broaden your problem-solving skills and get new perspectives on tackling problems.
- Improved Code Understanding: Studying different programming languages improves your ability to understand code, making you more versatile.
- Different Paradigms: Some obscure languages introduced programming paradigms that are still relevant today, even if the languages are not widely used anymore.
- Employability: Knowledge in a obscure language can lead to unique and well-paying job opportunities in niche markets.

- Learning from History: Studying older programming languages can provide insight into why modern languages are designed the way they are and why certain features exist.

The rest of the Introduction explains the choice of languages, outlines the intended audience, and specifies the required software.

About the Languages

The Online Historical Encyclopaedia of Programming Languages[1] (HOPL) lists approximately nine thousand languages arranged into two regnums (endogenous and exogenous), four phyla (algorithmic, functional, structural, and reflexive), twelve classes (conversational, imperative, spatially algorithmic, operation-oriented, expression-oriented, state/flow, lambda calculus, structural generic, structural specific, phenomenological, simulating, and close mapping), and forty orders (too many to list). Regardless of whether we take this classification as ground truth, the variety of coding paradigms is astonishing.

Most of the languages mentioned by the HOPL are dead languages. Some of them were popular at creation but went out of favor later. Some were "stillborn": created but never having caught the momentum. The others lived a long, respectable life and retired, superseded by more advanced tools.

The surviving languages could be grouped by their popularity among programmers. The position of a language on the TIOBE Index is an acceptable proxy for popularity.[2] As of the writing of this book, the Top 20 list includes Python, C, Java, C++, C#, Visual Basic and Classic VB, JavaScript, SQL, PHP, Go, Assembly language(s), MATLAB, Delphi, Scratch, R, Fortran, Ruby, Rust, and Swift. These languages are broadly used in industry and academia and cover the full development stack: front end, back end, and databases. They are loved and remembered—even Fortran, the oldest coding language.

TIOBE individually rates the next thirty languages, including COBOL, Perl, Objective-C, Ada, Lua, Lisp, Haskel, Kotlin, Scala, and Prolog. They are not forgotten, either. However, TIOBE lumps the next fifty languages together: their popularity is low and barely above the statistical margins of error (ActionScript, BCPL, Erlang/Elixir, Forth, J, Occam, PL/I, Scheme, Smalltalk, Tcl, and VHDL are in this group). And this is where things become getting intriguing.

1. https://hopl.info
2. https://www.tiobe.com/tiobe-index

A vast amount of literature exists on Erlang and Elixir (including Pragmatic's own *Programming Erlang [Arm13]* and *Modern Erlang for Beginners [Ost18]*). Scheme is popular in higher education for teaching organization of programming languages and similar topics. COBOL is a niche language: the entire financial and insurance industries run on it, but outside coders could not care less. Haskel and Lua are confined to their niches, too. Kotlin (2011) is relatively new in the field; let us give it a couple of years to shine in its full glory (or not). Every language in this and the following groups has a story of success and eventual failure—the story that almost nobody remembers. These are the languages that I call obscure.

The format of this book allows me to tell you only seven obscure language stories. Painful as it was, I had to choose those seven. I wanted the selection to be diverse and include feature-rich, bizarre, and promising languages, even if their promises would fail to be fulfilled.

As a result, I picked Snobol, Simula, APL, Forth, m4, Occam, and Starset. They feature, among other things, pattern matching, computer simulation, array and stack processing, macroprogramming, message passing, and set programming. Only m4 needs an explanation. It is not obscure because it was once known—it is "obscure" because it was never exposed to most software developers. Heavily used today by various system configuration tools, it deserves much more credit as a practical, Turing-complete language.

You can see the rise and fall of the "obscure" languages in the table below. The introduction years are taken from Wikipedia. The peak and decline years have been inferred from the Google Books Ngram Viewer, which is imperfect yet gives some ballpark estimates of what was hot and what was not.

	Introduced	Peaked	Declined
APL	1962–1966	1981	1995
Snobol	1962–1967	1972, 1982	1990
Simula	1962–1967	1985	2007
Forth	1968	1988	2013
m4	1977	n/a	active
Occam	1983	1989	1995
Starset	1991	n/a	n/a

From this point on, I will not put the word "obscure" in quotation marks. I hope we agree on its meaning: the "obscure" languages are not truly forgotten. They are in the dark. Together, we can bring them back to daylight.

About the Tips

This book is peppered with tips highlighting critical conceptual connections between the obscure languages and those currently in active use. Such subtle connections are not only a tribute to the groundbreaking ideas from the glorious past but the pulling ropes that could take the obscure languages out of obscurity and the rainbows pointing to the pots of software development gold waiting to be rediscovered.

The function `foobar()` in an Obscure Language loosely resembles the function `barfoo()` in a Popular Existing Language.

Think of these tips as a way to build and reinforce associations between Obscure Languages and Popular Existing Languages. The generative power of a network of associations is in its density. The denser the network, the more associations and creative ideas it generates in the reader, which is one of the reasons for reading the book.

About You

This book is intended for software developers seeking new, unorthodox, inspirational ideas to better their coding skills and theoretical understanding of computer language organization. However, it is also a crucial resource for IT managers and team leads. Understanding the older languages enables them to make informed decisions about legacy system maintenance, staff training, and system integration. Additionally, tech enthusiasts and software historians will find this book captivating, offering a deep dive into the evolutionary layers of coding languages. Whether you are hands-on with code or overseeing teams and projects, this book provides invaluable insights into obscure programming.

I hope that you, the reader, will see the connections between the concepts in the past and their implementations today (e.g., the first OOP language was designed for computer simulation; digital humanities date back to the early 1970s; indentation as a syntax feature is forty years old). Finally, you would know how living in a world of specialized programming languages rather than the general-purpose C, C++, Java, and Python could feel.

About the Software

In my youthful days, amid the stark realities of the former Soviet Union—known today as Russia—my only way of learning a new programming language

was to imagine myself being that language interpreter and interpreting the code written on paper. In the twenty-first century, you have better options. Any programming language, with few exceptions, has been eventually brought to life, one way or another—even Starset, the most obscure of the seven. The question is not whether an interpreter or a compiler of a language exists but where to find one. Being a devoted Linux user, I will advise you on how to get a grip on Linux versions of the interpreters if they exist.

The luckiest of the obscure languages, m4, is not truly forgotten. Rather, it is obscured by more specialized tools and unfamiliar to the users and even programmers. I know a good programmer who believes that m4 is no more. However, m4 is alive and kicking and at the heart of the GNU Autoconf system and Ratfor, a structured version of Fortran 66. As such, it is a part of any good Linux distribution; even if it is not, you would have no trouble installing it.

For unknown reasons, but most likely not because of their exceptional practical significance, three more languages made it into the GNU ecosystem: APL, Forth, and Simula. Their implementations became known as GNU APL, GNU Forth (gforth), and GNU Simula (cim), and they are reasonably well maintained.

Snobol4.2 from the University of Minnesota is closed-source and was compiled for Microsoft DOS when the grass was greener. The fact that it does not crash when executed under the DOSBox MS-DOS emulator must be due to witchcraft.

KRoC is the Kent Retargetable occam Compiler developed at the University of Kent. It is open-source but works only with 32-bit architectures. KRoC implements Occam-pi—a modern flavor of Occam 2.5 with some elements of π-calculus.

Finally, a Starset interpreter, christened “Suffolk StarSet” or s3, is developed and maintained by the team at my own Suffolk University. I promise to keep its current whereabouts public.

The following list shows links to the obscure development tools’ repositories at the time of writing the book.

- APL: GNU APL (apl),[3] Dyalog APL/S (dyalog, proprietary, but free for non-commercial use)[4]

3. https://www.gnu.org/software/apl/
4. https://www.dyalog.com/products.htm

- Forth: GNU Forth (gforth),[5] pForth (pforth)[6]
- m4: GNU m4 (m4)[7]
- Occam: Kent Retargetable occam Compiler (KRoC in Docker)[8]
- Simula: GNU Simula (cim)[9]
- SNOBOL: Minnesota SNOBOL4.2 (snobol.exe[10] for Microsoft DOS, runs on Linux with dosbox[11])
- Starset: Suffolk Starset (s3)[12]

An obligatory note: while every effort has been made to ensure that a functional compiler or interpreter of each obscure language exists for at least one popular platform (macOS, Linux, or Windows), this idyllic situation is hard to preserve due to the very definition of "being forgotten." If you want to enjoy this book thoroughly, get the software mentioned above while you can!

Writing Something Big

Forgotten or not, any programming language claiming a right to exist must be good at least for something beyond printing "Hello, world." The second-to-last section of each chapter is called "Writing Something Big." It presents a moderately sized (a page or two), relatively practical, and self-contained example of the chapter's language use. If the example feels offensively incomprehensible, you can safely skip it and proceed to the next chapter.

Further Reading

Each chapter concludes with a section titled "Further Reading" (just like this one). That section contains a curated list of further suggested reading on the subject.

One cannot expect to achieve a complete mastery of seven such diverse languages in seven weeks, and not only are they diverse, but each is a mind-breaker. Fortunately, many reference books and textbooks on most of these languages have been published during their peak popularity. To save you the

5. https://www.gnu.org/software/gforth/
6. http://www.softsynth.com/pforth/
7. https://www.gnu.org/software/m4/
8. https://github.com/omegahm/kroc
9. https://www.gnu.org/software/ciml/
10. http://berstis.com/snobol4.htm
11. https://www.dosbox.com/
12. https://missing.url

effort of searching for these materials, I've compiled a comprehensive list of books for each chapter. Please note, some of these resources may be hard to find, some only exist in scanned format, and a few are only available in Russian due to the absence of English translations.

However, if a particular language captivates you, say, like *the (as yet) unwritten content*, you will hopefully appreciate my cataloging effort made to support your learning journey. The books are listed chronologically to illuminate the flows and ebbs of the language popularity.

What to Do Next?

Choose the first obscure language to explore. Download and install its interpreter or compiler or grab a pencil and sheet of paper. Start reading and coding in awe.