68 RESOURCES

on

Creating Programming Languages

FEDERICO TOMASSETTI
Software Architect

Learn more at https://tomassetti.me

Here it is a new guide, to collect and organize all the knowledge that you need to create your programming language from scratch.

Creating a programming language is one of the most fascinating challenge you can dream of as a developer.

The problem is that there are a lot of moving parts, a lot of things to do right and it is difficult to find a well detailed map, to show you the way. Sure, you can find a tutorial on writing half a parser there, an half-baked list of advices on language design, an example of a naive interpreter. To find those things you will need to spend hours navigating forums and following links.

We thought it was the case to save you some time by collecting relevant resources, evaluate them and organize them. So you can spend time using good resources, not looking for them.

We organized the resources around the three stages in the creation of a programming language: design, parsing, and execution.

Table of contents

Designing the Language

When creating a programming language you need to take ideas and transform them in decisions. This is what you do during the design phase.

Before You Start…

Some good resources to beef up your culture on language design.

Articles

- Designing the next programming language? Understand how people learn!, a few considerations on how to design a programming language that it's easy to understand.
- Five Questions About Language Design, (some good and some random) notes on programming language design by Paul Graham.

Books

- Design Concepts in Programming Languages, if you want to make deliberate choices in the creation of your programming language, this is the book you need. Otherwise, if you don't already have the necessary theoretical background, you risk doing things the way everybody else does them. It's also useful to develop a general framework to understand how the different programming languages behave and why.
- Practical Foundations for Programming Languages, this is for the most part a book about studying and classifying programming languages. But by understanding the different options available it can also be used to guide the implementation of your programming language.
- Programming Language Pragmatics, 4th Edition, this is the most comprehensive book to understand contemporary programming

languages. It discusses different aspects of everything from C# to OCaml and even the different kinds of programming languages such as functional and logical ones. It also covers the several steps and parts of the implementation, such as an intermediate language, linking, virtual machines, etc.

- Structure and Interpretation of Computer Programs, Second Edition, an introduction to computer science for people that already have a degree in it. A book widely praised by programmers, including Paul Graham directly on the Amazon Page, that helps you developing a new way to think about programming language. It's quite abstract and examples are proposed in Scheme. It also covers many different aspect of programming languages including advanced topics like garbage collection.

Type Systems

Long discussions and infinite disputes are fought around type systems. Whatever choices you end up making it make sense to know the different positions.

Articles

- These are two good introductory articles on the subject of type systems. The first discuss the dichotomy Static/Dynamic and the second one dive into Introspection.
- What To Know Before Debating Type Systems, if you already know the basics of type systems this article is for you. It will permit you to understand them better by going into definitions and details.
- Type Systems (PDF), a paper on the formalization of Type Systems that also introduces more precise definitions of the different type systems.

Books

- Types and Programming Languages, a comprehensive book on understanding type systems. It will impact how your ability to design

programming languages and compilers. It has a strong theoretical support, but it also explains the practical importance of individual concepts.

- Functional programming and type systems, an interesting university course on type systems for functional programming. It is used in a well known French university. There are also notes and presentation material available. It is as advanced as you would expect.
- Type Systems for Programming Language, is a simpler course on type system for (functional) programming languages.

Parsing

Parsing transform the concrete syntax in a form that is more easily manageable by computers. This usually means transforming text written by humans in a more useful representation of the source code, an Abstract Syntax Tree.

There are usually two components in parsing: a lexical analyzer and the proper parser. Lexers, which are also known as tokenizers or scanners, transform the individual characters in tokens, the atom of meaning. Parsers instead organize the tokens in the proper Abstract Syntax Tree for the program. But since they are usually meant to work together you may use a single tool that does both the tasks.

Tools

- Flex, as a lexer generator and (Berkeley) Yacc or Bison, for the generation of the proper parser, are the venerable choices to generate a

complete parser. They are a few decades old and they are still maintained as open source software. They are written in and thought for C/C++. They still works, but they have limitations in features and support for other languages.

- ANTLR, is both a lexer and a parser generator. It's also more actively developed as open source software. It is written in Java, but it can generate both the lexer and the parser in languages as varied as C#, C++, Python, Javascript, Go, etc.
- *Your own lexer and parser.* If you need the best performance and you can create your own parser. You just need to have the necessary computer science knowledge.

Tutorials

- Flex and Bison tutorial, a good introduction to the two tools with bonus tips.
- Lex and Yacc Tutorial, at 40 pages this is the ideal starting point to learn how to put together lex and yacc in a few hours.
- Video Tutorial on lex/yacc in two parts, in an hour of video you can learn the basics of using lex and yacc.
- ANTLR Mega Tutorial, the renown and beloved tutorial that explains everything you need to know about ANTLR, with bonus tips and tricks and even resources to know more.

Books

- lex & yacc, despite being a book written in 1992 it's still the most recommended book on the subject. Some people say because the lack of competition, others because it is good enough.
- flex & bison: Text Processing Tools, the best book on the subject written in this millennium.
- The Definitive ANTLR 4 Reference, written by the main author of the tool this is really the definitive book on ANTLR 4. It explains all of its

secrets and it’s also a good introduction about how the whole parsing thing works.

- Parsing Techniques, 2nd edition, a comprehensive, advanced and costly book to know more than you possibly need about parsing.

Execution

To implement your programming language, that is to say to actually making something happens, you can build one of two things: a compiler or an interpreter. You could also build both of them if you want. Here you can find a good overview if you need it: Compiled and Interpreted Languages.

The resources here are dedicated to explaining how compilers and/or interpreters are built, but for practical reasons often they also explain the basics of creating lexers and parsers.

Compilers

A compiler transforms the original code into something else, usually machine code, but it could also be simply any lower level language, such as C. In the latter case some people prefer to use the term *transpiler*.

Tools

- LLVM, a collection of modular and reusable compiler and toolchain technologies used to create compilers.
- CLR, is the virtual machine part of the .NET technologies, that permits to execute different languages transformed in a common intermediate language.
- JVM, the Java Virtual Machine that powers the Java execution.

Articles & Tutorials

- Building Domain Specific Languages on the CLR, an article on how to build internal DSL on the CLR. It's slightly outdated, since it's from 2008, but it's still a good presentation on the subject.

- The digital issue of MSDN Magazine for February 2008 (CHM format), contains an article on how to Create a Language Compiler for the .NET Framework. It's still a competent overview of the whole process.
- Create a working compiler with the LLVM framework, Part 1 and Part 2, a two-part series of articles on creating a custom compiler by IBM, from 2012 and thus slightly outdated.
- A few series of tutorials from the LLVM Documentation, this is three great linked series of tutorial on how to implement a language, called Kaleidoscope, with LLVM. The only problem is that some parts are not always up-to-date.
- My First LLVM Compiler, a short and gentle introduction to the topic of building a compiler with LLVM.
- Creating an LLVM Backend for the Cpu0 Architecture, a whopping 600-pages tutorial to learn how to create a LLVM backend, also available in PDF or ePub. The content is great, but the English is lacking. On the positive side, if you are a student, they feel your pain of transforming theoretical knowledge into practical applications, and the book was made for you.
- A Nanopass Framework for Compiler Education, a paper that present a framework to teach the creation of a compiler in a simpler way, transforming the traditional monolithic approach in a long series of simple transformations. It's an interesting read if you already have some theoretical background in computer science.
- An Incremental Approach to Compiler Construction (PDF), a paper that it's also a tutorial that develops a basic Scheme compiler with an easier to learn approach.

Books

- Compilers: Principles, Techniques, and Tools, 2nd Edition, this is the widely known Dragon book (because of the cover) in the 2nd edition (purple dragon). There is a paperback edition, which probably costs less

but it has no dragon on it, so you cannot buy that. It is a theoretical book, so don't expect the techniques to actually include a lot of reusable code.
- Engineering a Compiler, 2nd edition, it is another compiler book with a theoretical approach, but that it covers a more modern approach and it is more readable. It's also more dedicated to the optimization of the compiler. So if you need a theoretical foundation and an engineering approach this is the best book to get.

Interpreters

An interpreter directly executes the language without transforming it in another form.

Articles & Tutorials

- A simple interpreter from scratch in Python, a four-parts series of articles on how to create an interpreter in Python, simple yet good.
- Let's Build A Simple Interpreter, a twelve-parts series that explains how to create a interpreter for a subset of Pascal. The source code is in Python, but it has the necessary amount of theory to apply to another language. It also has a lot of funny images.
- How to write an interpreter, a screencast, with source code available, on how to write an interpreter of a simple language with Python.
- How to write a simple interpreter in JavaScript, a well organized article that explains how to create a simple interpreter in JavaScript.

Books

- Writing An Interpreter In Go, despite the title it actually shows everything from parsing to creating an interpreter. It's contemporary book both in the sense that is recent (a few months old), and it is a short one with a learn-by-doing attitude full of code, testing and without 3-rd party libraries. We have interviewed the author, Thorsten Ball.

- Crafting Interpreters, a work-in-progress and free book that already has good reviews. It is focused on making interpreters that works well, and in fact it will builds two of them. It plan to have just the right amount of theory to be able to fit in at a party of programming language creators.

General

This are resources that cover a wide range of the process of creating a programming language. They may be comprehensive or just give the general overview.

Tools

In this section we include tools that cover the whole spectrum of building a programming language and that are usually used as standalone tools.

- Xtext, is a framework part of several related technologies to develop programming languages and especially Domain Specific Languages. It allows you to build everything from the parser, to the editor, to validation rules. You can use it to build great IDE support for your language. It simplifies the whole language building process by reusing and linking existing technologies under the hood, such as the ANTLR parser generator.
- JetBrains MPS, is a projectional language workbench. Projectional means that the Abstract Syntax Tree is saved on disk and a projection is presented to the user. The projection could be text-like, or be a table or diagram or anything else you can imagine. One side effect of this is that you will not need to do any parsing, because it is not necessary. The term *Language Workbench* indicates that Jetbrains MPS is a whole system of technologies created to help you create your own programming language: everything from the language itself to IDE and supporting tools designed for your language. You can use it to build every kind of language, but the possibility and need to create everything makes it ideal to create Domain

Specific Languages that are used for specific purposes, by specific audiences.

- Racket, is described by its authors as "a general-purpose programming language as well as the world's first ecosystem for developing and deploying new languages". It's a pedagogical tool developed with practical ambitions that has even a manifesto. It is a language made to create other languages that has everything: from libraries to develop GUI applications to an IDE and the tools to develop logic languages. It's part of the Lisp family of languages, and this tells everything you need to know: it's all or nothing and always the Lisp-way.

Articles

- Create a programming language for the JVM: getting started, an overview of how and why to create a language for the JVM.
- An answer to How to write a very basic compiler, a good answer to the question that gives an overview of the steps needed and the options available to perform the task of building a compiler.
- Creating Languages in Racket, a great overview and presentation of Racket from the ACM Journal, with code.
- A Tractable Scheme Implementation (PDF), a paper discussing a Scheme implementation that focuses on reliability and tractability. It builds an interpreter that will generate a sort of bytecode on the fly. This bytecode will then be immediately executed by a VM. The name derives from the fact that the original version was built in 48 hours. The full source code is available on the website of the project.

Tutorials

- Create a useful language and all the supporting tools, a series of articles that start from scratch and teach you everything from parsing to build an editor with autocompletion, while building a compiler targeting the JVM.

- There is a great deal of documentation for Racket that can help you to start using it, even if you don't know any programming language.
- There is a good amount documentation for Xtext that can help you to start using it, including a couple of 15 minutes tutorials.
- There is a great deal of documentation for JetBrains MPS, including specialized guides such as one for expert language designers. There is a video channel with videos to help you use the software and an introduction on Creating your first language in JetBrains MPS.
- Make a language in one hour: stacker, the tutorial provides a tour of Racket and its workflow.
- Make Your Own Programming Language, a 5-parts series that provides a simple example on the principles of creating a programming language works, build with JavaScript.
- Create Your Own Programming Language, an article that shows a simple and hacky way of creating a programming language using JavaCC to create a parser and the Java reflection capabilities. It's clearly not the proper way of doing it, but it presents all the steps and it's easy to follow.
- Writing Your Own Toy Compiler Using Flex, Bison and LLVM, it does what it says, using the proper tools (flex, bison, llvm, etc.) but it's slightly outdated since it's from 2009. If you want to understand the general picture and how everything fit together this is still a good place where to start.
- Project: A Programming Language, this is a chapter of the book Eloquent Javascript. It shows how to create a simple programming language using JavaScript and parsing with regular expressions. This is all so wrong, yet it's also bizarrely good. The author does it to demystify the creation of programming language. You shouldn't do any of that stuff, but you might find useful to read it.
- Designing a Programming Language I, "Designing a language and building an interpreter from beginning to end". It is more than an article and less than a book. It has a good mix of theory and practice and it

implements what it calls Duck Programming Language (inspired from Duck-Typing). A Part ii, that explained how to create a compiler, was planned but never finished.

- Introduction to building a programming language, A 100 minutes video that implements a subset of PHP in JavaScript: it is a bit troubling, but it is also undeniably cool.
- Writing a compiler in Ruby, bottom up, a 45-parts series of articles on creating a compiler with Ruby. For some reason it starts bottom up, that is to say from the code generation to end up with the parser. This is the reverse of the traditional (and logical) way of doing things. It's peculiar, but also very down-to-earth.
- Implementing Programming Languages Using C# 4.0, the approach is a simple one and the libraries are quite outdated, but it's a neat article to read a good introduction on how to build an interpreter in C#.
- How to create your own virtual machine! (PDF), this tutorial explains how to create a virtual machine in C#. It's surprisingly interesting, although not necessarily with a practical application.

Books

- How to create pragmatic, lightweight languages, the focus here is on making a language that works in practice. It explains how to generate bytecode, target the LLVM, build an editor for your language. Once you read the book you should know everything you need to make a usable, productive language. Incidentally, we have written this book.
- How To Create Your Own Freaking Awesome Programming Language, it's a 100-page PDF and a screencast that teach how to create a programming language using Ruby or the JVM. If you like the quick-and-dirty approach this book will get you started in little time.
- Writing Compilers and Interpreters: A Software Engineering Approach, 3rd edition, it's a pragmatic book that still teaches the proper approach to compilers/interpreters. Only that instead of an academic focus, it has an

engineering one. This means that it's full of Java code and there is also UML sprinkled here and there. Both the techniques and the code are slightly outdated, but this is still the best book if you are a software engineer and you need to actually do something that works correctly right now, that is to say in a few months after the proper review process has completed.

- Language Implementation Patterns, this is a book from the author of ANTLR, which is also a computer science professor. So it's a book with a mix of theory and practice, that guides you from start to finish, from parsing to compilers and interpreters. As the name implies, it focuses on explaining the known working patterns that are used in building this kind of software, more than directly explaining all the theory followed by a practical application. It's the book to get if you need something that really works right now. It's even recommended by Guido van Rossum, the designer of Python.
- Build Your Own Lisp, it's a very peculiar book meant to teach you how to use the C language and how to build you own programming language, using a mini-Lisp as the main example. You can read it for free online or buy it. It's meant you to teach about C, but you have to be already familiar with programming. There is even a picture of Mike Tyson (because… lisp): it's all so weird, but fascinating.
- Beautiful Racket: how to make your own programming languages with Racket, it's a good and continually updated online book on how to use Racket to build a programming language. The book is composed of a series of tutorials and parts of explanation and reference. It's the kind of book that is technically free, but you should pay for it if you use it.
- Programming Languages: Application and Interpretation, an interesting book that explains how to create a programming language from scratch using Racket. The author is a teacher, but of the good and understable

kind. In fact, there is also a series of recordings of the companion lectures, that sometimes have questionable audio.

- Implementing Domain-Specific Languages with Xtext and Xtend, 2nd edition, is a great book for people that want to learn with examples and using a test-driven approach. It covers all levels of designing a DSL, from the design of the type system, to parsing and building a compiler.
- Implementing Programming Languages, is an introduction to building compilers and interpreters with the JVM as the main target. There are related materials (presentations, source code, etc.) in a dedicated webpage. It has a good balance of theory and practice, but it's explicitly meant as a textbook. So don't expect much reusable code. It's the typical textbook also in the sense that it can be a great and productive read if you already have the necessary background (or a teacher), otherwise you risk ending up confused.
- Implementing functional languages: a tutorial, a free book that explains how to create a simple functional programming language from the parsing to the interpreter and compiler. On the other hand: "this book gives a practical approach to understanding implementations of non-strict functional languages using lazy graph reduction". Also, expect a lot of math.
- DSL Engineering, a great book that explains the theory and practice of building DSLs using language workbenches, such as MPS and Xtext. This means that other than traditional design aspects, such as parsing and interpreters, it covers things like how to create an IDE or how to test your DSL. It's especially useful to software engineers, because it also discusses software engineering and business related aspects of DSLs. That is to say it talks about why a company should build a DSL.
- Lisp in Small Pieces, an interesting book that explain in details how to design and implement a language of the Lisp family. It describes "11 interpreters and 2 compilers" and many advanced implementation details

such as the optimization of the compiler. It’s obviously most useful to people interested in creating a Lisp-related language, but it can be an interesting reading for everybody.

Summary

Here you have the most complete collection of high-quality resources on creating programming languages. You have just to decide what you are going to read first.

At this point we have two advices for you:

1. Get started. It does not matter how many amazing resources we will send you, if you do not take the time to practice, trying and learning from your mistake you will never create a programming language
2. If you are interested in building programming languages you should subscribe to our newsletter. You will receive updates on new articles, more resources, ideas, advices and ultimately become part of a community that share your interests on building languages

You should have all you need to get started. If you have questions, advices or ideas to share feel free to write at federico@tomassetti.me. We read and answer every email.

Our thanks to Krishna for a few good suggestions.

What's next?

If you have questions or doubts you can always look for more content on my blog.

There is a category specific for Language Engineering:

https://tomassetti.me/category/language-engineering

If you cannot find an answers write to me and I will try my best to help you. You will find me at federico@tomassetti.me .

If you think that a language will help your organization you can schedule a roadmap call and we can figure out together what is the best course of action.