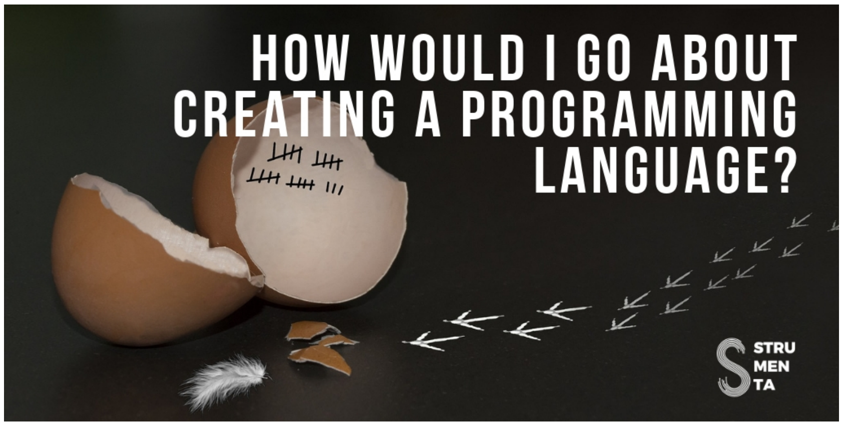
I am planning to create my own language first. A language that would have simplistic syntax.

How would I go about creating a programming language?

May 8, 2019 / in Create a programming language / by Federico Tomassetti



The title of this article reflects a question I hear over and over in forums or in emails I receive.

I think all curious developers asked it at least once. It is normal to be fascinated by how programming languages work. Unfortunately, most answers we read are very academic or theoretical. Some others contain too much implementation details. After reading them we still wonder how things work *in practice*.

So we are going to answer it. Yes, we will see what is the process for creating you own full language with a compiler for it and what not.

The overview

Most persons who wants to learn how to “create a programming language” are effectively looking for information on how to build a compiler. They want to understand the mechanics that permit to execute a new programming language.
A compiler is a fundamental piece of the puzzle but making a new programming language requires more than that:

1) A language has to be *designed*: the language creator has to take some fundamental decisions about the paradigms to be used and the syntax of the language
2) A compiler has to be created
3) A standard library must be implemented
4) Supporting tools like editors and build systems have to be provided

These are the only steps of the process I am interested in. A standard library and supporting tools are only required if I need to make the language more widespread. I will only concern myself with the first two steps for the final product.

Let’s see more in details what each of these points entail.

Designing a programming language

If you want just to write your own compiler to learn how these things work, you can skip this phase. You can just take a subset of an existing language or come up with a simple variation of it and get started. However, if you have plans for creating your very own programming language, you will have to give it some thought.

I think of designing a programming language as divided two phases:

1. The big-picture phase
2. The refinement phase

In the first phase we answer the fundamental questions about our language.

- What execution paradigm do we want to use? Will it be imperative or functional? Or maybe based on state machines or business rules [https://tomassetti.me/a-complete-tutorial-on-the-drools-business-rule-engine/] ?

  I'll use a imperative paradigm as that is what I am more familiar with. I also believe that such a design is more user friendly.
- Do we want static typing or dynamic typing?

  This is important, this will largely depend on what language I choose to use for the compiler.
- What sort of programs this language will be best at? Will it be used for small scripts or large systems?

  Small Scripts
- What matters most to us: performance? Readability?

  Readability
- Do we want it to be similar to an existing programming language? Will it be aimed at C developers or easy to learn for who is coming from Python?

  Similar to HyperTalk/ Python due to their readability
- Do we want it to work on a specific platform (JVM, CLR)?

  Nope
- What sort of metaprogramming capabilities do we want to support, if any? Macros? Templates? Reflection?

  No need for Such capabilities

In the second phase we will keep evolving the language as we use it. We will run into issues, into things that are very difficult or impossible to express in our language and we will end up evolving it. The second phase might not be as glamorous as the first one, but it is the phase in which we keep tuning our language to make it usable in practice, so we should not underestimate it.

Building a compiler

Building a compiler is the most exciting step in creating a programming language. Once we have a compiler we can actually bring our language to life. A compiler permits us to start playing with the language, use it and identify what we miss in the initial design. It permits to see the first results. It is hard to beat the joy of executing the first program written in our brand new programming language, no matter how simple that program may be.

But how do we build a compiler?

As everything complex we do that in steps:

1. **We build a parser**: the parser is the part of our compiler that takes the text of our programs and understand which commands they express. It recognizes the expressions, the statements, the classes and it creates internal data structures to represent them. The rest of the parser will work with those data structures, not with the original text

   This will be the most important step for my language as it will function similar to a higher level abstraction of another language.

2. *(optional)* **We translate the parse tree into an Abstract Syntax Tree**. Typically the data structures produced by the parser are a bit low level as they contain a lot of details which are not crucial for our compiler. Because of this we want frequently to rearrange the data structures in something slightly more higher level
3. **We resolve symbols**. In the code we write things like `a + 1`. Our compiler needs to figure out what `a` refers to. Is it a field? Is it a variable? Is it a method parameter? We examine the code to answer that

   This makes me wonder whether I would implement symbols in my language or not

4. **We validate the tree**. We need to check the programmer did not commit errors. Is he trying to sum a boolean and an int? Or accessing an non-existing field? We need to produce appropriate error messages

   This is very important as handling errors is something I had not thought of before.

5. **We generate the machine code**. At this point we translate the code in something the machine can execute. It could be proper machine code or bytecode for some virtual machine
6. *(optional)* **We perform the linking**. In some cases we need to combine the machine code produced for our programs with the code of static libraries we want to include, in order to generate a single executable

These steps are not important as I won't be converting to machine code but compiling using another language as a base

Do we always need a compiler? No. We can replace it with other means to execute the code:

- We can write an interpreter: an interpreter is substantially a program that does steps 1-4 of a compiler and then directly executes what is specified by the Abstract Syntax Tree
- We can write a transpiler [https://tomassetti.me/transpiling-languages-from-vba-to-vb-net/] : a transpiler will do what is specified in steps 1-4 and then output some code in some language for which we have already a compiler (for example C++ or Java)

This seems more like what I would be building

These two alternatives are perfectly valid and frequently it makes sense to choose one of these two because the effort required is typically smaller.

We wrote an article explaining how to write a transpiler [https://tomassetti.me/transpiling-languages-from-vba-to-vb-net/] . Take a look at it if you want to see a practical example, with code.

In this article we explain in more details the difference between a compiler and an interpreter [https://tomassetti.me/difference-between-compiler-interpreter/] .

A standard library for your programming language

Any programming language needs to do a few things:

- Printing on the screen This is the one functionality I will probably add
- Accessing the filesystem
- Use network connections
- Creating GUIs

These are the basic functionalities to interact with the rest of the system. Without them a language is basically useless. How do we provide these functionalities? By creating a standard library. This will be a set of functions or classes that can be called in the programs written in our programming language but that will be written in some other language. For example, many languages have standard libraries written at least partially in C.

A standard library can then contain much more. For example classes to represent the main collections like lists and maps, or to process common formats like JSON or XML. Often it will contain advanced functionalities to process strings and regular expressions.

In other words, writing a standard library is a lot of work. It is not glamorous, it is not conceptually as interesting as writing a compiler but it is still a fundamental component to make a programming language viable.

There are ways to avoid this requirement. One is to make the language run on some platform and make it possible to reuse the standard library of another language. For example, all languages running on the JVM can simply reuse the Java standard library.

This seems like something I would be able to adapt for my language.

Supporting tools for a new programming language

To make a language usable in practice we frequently need to write a few supporting tools.

The most obvious is an editor. A specialized editor with syntax highlighting, inline error checking, and auto-completion is nowadays a must have to make any developer productive.

But today developers are spoiled and they will expect all sort of other supporting tools. For example, a debugger could be really useful to deal with a nasty bug. Or a build system similar to maven or gradle could be something that users will ask later on.

At the very beginning an editor could be enough but as your user base grows also the complexity of projects will grow and more supporting tools will be needed. Hopefully at that time there will be a community willing to help building them.

Summary

Creating a programming language is a process that seems mysterious to many developers. In this article we tried to show that it is just a process. It is fascinating and not easy, but it can be done.

You may want to build a programming language for a variety of reasons. One good reason is for fun, another one is for learning how compilers work. Your language could end up being very useful or not, depending on many factors. However if you have fun and/or learn while building it then it is worth investing some time on this.

And of course you will be able to brag with your fellow developers.

If you want to learn more about creating a language take a look at the other resources we created: learn how to build languages [https://tomassetti.me/learning-build-languages/] .

You may also be interested in some of our articles:

- 68 Resources To Help You To Create Programming Languages [https://tomassetti.me/resources-create-programming-languages/]
- The complete guide to (external) Domain Specific Languages [https://tomassetti.me/domain-specific-languages/]