

# Chapter 01

## 1 Tell me more about Rust!

### 1.1 A bit of history

Initially, [Rust](#) started out in 2006 as a personal project from Graydon Hoare—a language designer “by trade” to quote his own words. Developed in his free time during the first three years, Rust was then presented to his employers at Mozilla, which went on and started to actively support it. Indeed, Mozilla Research caught interest in it as something which could do away in the future with various recurrent bugs found in Gecko, Firefox’s rendering engine written in C++.

An original compiler has been written in [OCaml](#) until a bootstrapped compiler written in Rust itself could take over, in 2010. Right now the compilation process uses an [LLVM](#) backend (which you might have seen in action inside [Clang](#)).

### 1.2 The Goodness

So what is so good about Rust? We talked about C++ right at the beginning and you might think that since Mozilla is actively leading its development, it should actually have some significant advantages compared to it.

First, what are people looking for when they want to use C++?

- A [System programming language](#), which means it does not compromise on performance and allows for low-level access to the available resources (pointers, manual memory management, etc.)
- A language with a fairly high-level of abstraction and a large feature-set (OO-programming is one of them) that stays fast and has broad compatibility – along with a large feature set

Now, what are the main features that Rust brings on top of that?

- Safety (and mostly: safe pointer types thanks to a [very efficient memory model](#), no void\* heresy<sup>∅</sup>, no use-after-free, mem leaks,...)

- Native actor-based concurrency inspired from [Erlang](#). This is a big deal for today's programming languages with the rise of multicore processors; in fact, some sort of multithreading is already there on C++ Boost and maybe in C++14 at a later date! –actually, the actor implementation of multithreading consists of explicitly sending message from a lightweight thread (`chan`) and receiving it from another (`port`); it's a safety/efficiency compromise
- Inspirations from [Haskell](#) and [OCaml](#), not only for the functional programming paradigm but also for the type classes (`type`, `impl`)
- Some easier to use functionalities from Python –that being said, Rust does not compromise on safety: it has static typing (with type inference!)
- Optional tracing-GC (WIP) –the `std::gc::Gc` type recently landed on mainline

Rust does not try to be a new fancy thing; rather, it builds up on top of proven useful technology and merges features from some existing languages, noticeably: Erlang's green threads, Haskell trait system and overall functional programming elements of a few languages (it has pattern matching!), some syntax sugar from Python (the `range` iterator is one of them) and it also resurrects a few features from some American research languages developed in the 90s, amongst others. As you can see, along with some features that definitely set it in today's rollup of programming languages, Rust is also the result of a true willingness to reuse concepts from the rich history of programming languages, rather than building up the wheel over again. ;) ~ as you might have seen, that's partly where the name "Rust" comes from!

I was fiddling with a number of names. Primarily it's named after the [pathogenic fungi](#) which are just amazing multi-lifecycle, multi-host parasites, fascinating creatures.

But the name "rust" has other nice features. It fits with our theme of being "close to the metal" and "using old well-worn language technology". It's also a substring of "trust" and "robust". What's not to like?

– Graydon Hoare

Actually, Mozilla is already using it in an experimental browser rendering engine that is being developed simultaneously to Rust and is at the core of the team's pragmatic development approach, just like [D](#)'s—that is, the developers implement features and sometimes change them based on their experience with it; besides, they also take feedback (and contributions!) from the community. Remember, C—today's most successful language—has been built in concert with the UNIX OS, allowing adaptations from what happened during the OS development process. Right now, Rust is still in alpha state which means that developers can still make breaking changes to the API (but not to worry, it is converging towards a stable state which will ultimately happen in 1.0).

There have been other attempts at providing such a new System programming language (e.g. [D](#), [Nimrod](#)) –Go also gets mentioned sometimes but it has been developed for apps/webapps and is oriented towards simplicity and fast compilation-speed.

Right now, it looks like Rust has a big chance to set itself apart because:

- it focuses on safety without compromising performance (D requires GC for safety, and Nimrod does not ensure thread safety), it has low-level features but also lots of sugar for the developer
- it is pretty innovative! –seeing how it reuses a lot of features that have been developed independently, but make sense together
- it is backed by Mozilla, not bad if you are trying to set a standard!

### 1.3 Appendix: Elaborating on C++ and safety

◊You might think: why were these even allowed in the first place? To quote Tony Hoare, the `null_ptr` precursor:

I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

– Tony Hoare

C++ is a mixed-breed: it has been developed from the beginning to build up on top of C with new features that were emerging in the field of programming. Today, it is one of the [Top Five languages](#) because it has been fast at a higher level of abstraction (also complexity!) than the others. Looking back, we realized that the time spared by relaxed programming APIs was often creating much bigger problems down the road—because it is always less time consuming and more convenient to just avoid these programming derps than debugging your program down the road.

Rust tries to ensure safety with some additional syntax markup (which actually makes you to think unambiguously about your desired behavior) that allows for a way more fine-grained static analysis by the compiler and can lead to additional compiler optimizations—indeed, some unsafe code that would pass as valid in most of today's most used languages would get caught at compile-time in Rust. Instructions that are not safe or cannot be statically checked will have to be located inside an `unsafe` code block (that is mostly for the FFI, that allows you to link your programs to C).

### 1.4 Appendix: Elaborating on Servo

What is [Servo](#)? Well—nothing to do with mechanics: it is Mozilla's experimental browser rendering engine.

So, in a way, you could say that it's their biggest [hope](#) and biggest motivation for supporting Rust's development. Safety. Parallelism. (And Performance.)

Its development started in early-2012 (they like contributions too!) and it is evolving along with Rust.

Servo is built on a modular architecture and each engine component is separated from the other in an effort to have parallelized and easy-to-maintain code. Since April 2013, Mozilla and Samsung [are collaborating](#) in the project development.

Note that right now, Mozilla has no plans to integrate Servo onto Firefox, ie. it is as of now a research project and is still in early stages of development. BTW, Servo [passed Acid1](#) recently!

% Chapter 02

## 2 Few things you should know

The Rust API is frequently evolving. While the use of the 0.8 release is recommended on Windows (unless you're willing to [build it](#)), this tutorial will be aimed at current master which has the latest language contents.

Note that much of the “standard” features are pretty much stabilized and shouldn't change much. If you have any concern related to the language or its syntax, just [jump on IRC](#) and ask the others!

## 3 Installing Rust

### 3.1 Windows

Checkout the [building instructions](#).

Note: mingw-w64 support is incoming on latest trunk.

If you're not willing to go through these steps, just download the [0.8 installer](#).

### 3.2 Mac OS X

The installation process on Mac OS is fairly easy with the [Homebrew package manager](#), for the latest trunk:

```
brew install --HEAD rust
```

And to get the latest release:

```
brew install rust
```

### 3.3 Linux

Okay Linux users, let's build the beast:

```
git clone https://github.com/mozilla/rust.git
cd rust
./configure
make
sudo make install
```

And for the 0.8 release, use these sources:

```
curl -O http://static.rust-lang.org/dist/rust-0.8.tar.gz
tar -xzf rust-0.8.tar.gz
cd rust-0.8
./configure
make
sudo make install
```

Note that there are repositories with nightly packages for [Arch](#) and [Ubuntu](#).