|  |  |
| --- | --- |
| Rust de toekomst van de opleiding? | Abstract  Dit onderzoek gaat over de toepasbaarheid van de taal Rust. Er wordt onderzocht of Rust een goede aanvullende taal is voor de opleiding HBO-ICT met afstudeerrichting Technische Informatica. Eerst zal er gekeken worden naar wat de taal Rust te bieden heeft later wordt er gekeken waar de taal Rust mogelijk een plek zou kunnen hebben binnen de opleiding  Mike Hilhorst  1676029 |

# Over rust

Rust

**Rust is the most loved language of developers. Here’s why**

[Rust](https://www.rust-lang.org/en-US/), a new systems programming language from Mozilla is becoming increasingly popular, due to features such as memory safety and its performance and has rapidly grown – it now has 90 [Rust User Groups](https://www.rust-lang.org/en-US/user-groups.html) in over 35 countries.

**Why is Rust growing in popularity?**

According to the 2018 Stack [overflow developer survey](https://insights.stackoverflow.com/survey/2018/#most-loved-dreaded-and-wanted), Rust was the “Most Loved” programming language for the third year in a row and in the top ten of “Most Wanted” languages. This continues the trend of the language’s popularity surge.

(For the sixth year in a row, [JavaScript](https://stackoverflow.com/jobs?sort=i&q=javascript) was the most commonly used programming language. Python surpassed C# this year, much like it surpassed PHP last year. Python has a solid claim to being the [fastest-growing major programming language](https://stackoverflow.blog/2017/09/06/incredible-growth-python/).)

So, why has Rust become so popular and endured the test of time, in the developer community?

Evan Wallace, CTO of interface design tool company [Figma](https://www.figma.com/) says of its use when redeveloping a server: “We chose Rust for this rewrite because it combines best-in-class speed with low resource usage while still offering the safety of standard server languages. Low resource usage was particularly important to us because some of the performance issues with the old server were caused by the garbage collector”.

(Garbage collection is a form of automatic memory management; used by Go, Java or .NET Framework to name a few examples. Rust doesn’t use it: instead, memory and other resources are managed through “RAII”, or [resource acquisition is initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization)).

This is in keeping with the languages core features, focused on providing low-level functionality whilst[providing:](https://blog.figma.com/rust-in-production-at-figma-e10a0ec31929)

* zero-cost abstractions
* guaranteed memory safety
* threads without data races
* type inference
* minimal runtime

**What is it being used for?**

Open Source tech firm Mozilla originally [created](https://www.rust-lang.org/en-US/faq.html" \l "project)Rust to solve the problems of its precursors, namely safety, concurrency, control over resources and other such antiquated language features.

Furthermore, Rust was designed as a systems language which is being used in applications such as [Servo](https://servo.org/) (a Mozilla created browser engine) and [Redox](https://www.redox-os.org/), an operating system with Unix undertones, depicting its usefulness within the context of low-level systems which require uncompromising speed and efficiency.

Overall, its blazingly fast speed, coupled with the control that it is giving to developers working in such contexts, is paying dividends with the developer community.

**Going forward**

It will be interesting to see how teams adopt the language for more common application use cases, in the same way that Figma did, who spoke [very positively](https://blog.figma.com/rust-in-production-at-figma-e10a0ec31929) in their review of the language: “While we hit some speed bumps, I want to emphasize that our experience with Rust was very positive overall. It’s an incredibly promising project with a solid core and a healthy community. I’m confident these issues will end up being solved over time.”

It seems at this rate, Rust will continue to be refined (no pun intended) into a language that will have a great impact on the developer community.

*Want to learn Rust? Also known as “The Book”,*[*The Rust Programming Language*](https://doc.rust-lang.org/book/)*is the primary official document of the language.*

# <https://www.cbronline.com/analysis/rust-language>

# Beantwoording deelvragen

Hier worden de deelvragen beantwoordt

## Voor welke platformen is/wordt Rust ontwikkeld?

Rust ondersteunt vele platformen. Denk aan bijvoorbeeld windows, MacOS, Linux en embedded microcontrollers. Rust ondersteunt deze platvormen op verschillende niveaus. Er zijn 4 niveaus, deze verschillen per ondersteuning denk hierbij aan ondersteuning voor de standaard bibliotheek, Rustc en cargo.   
Niveau 3 is per ongeluk ontstaan en nieuwe platformen zullen dit niveau nooit krijgen.

**Niveau 1**

Dit niveau is goed ondersteunt en kan gezien worden als “Gegarandeerd werkend”.   
Deze platformen hebben de volgende eigenschappen:

* Officiële binary’s worden geleverd voor deze platformen.
* Automatische testen bestaan voor deze platformen.
* Veranderingen in de taal zijn master worden pas doorgelaten na testen geslaagd zijn
* Documentatie over gebruik is beschikbaar voor deze platformen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target | std | rustc | cargo | notes |
| i686-apple-darwin | ✓ | ✓ | ✓ | 32-bit OSX (10.7+, Lion+) |
| i686-pc-windows-gnu | ✓ | ✓ | ✓ | 32-bit MinGW (Windows 7+) |
| i686-pc-windows-msvc | ✓ | ✓ | ✓ | 32-bit MSVC (Windows 7+) |
| i686-unknown-linux-gnu | ✓ | ✓ | ✓ | 32-bit Linux (2.6.18+) |
| x86\_64-apple-darwin | ✓ | ✓ | ✓ | 64-bit OSX (10.7+, Lion+) |
| x86\_64-pc-windows-gnu | ✓ | ✓ | ✓ | 64-bit MinGW (Windows 7+) |
| x86\_64-pc-windows-msvc | ✓ | ✓ | ✓ | 64-bit MSVC (Windows 7+) |
| x86\_64-unknown-linux-gnu | ✓ | ✓ | ✓ | 64-bit Linux (2.6.18+) |

**Niveau 2**   
Dit niveau is ondersteunt en kan gezien worden als “Gegarandeerd bouwend”.   
Deze platformen hebben de volgende eigenschappen:

* Officiële binarys worden geleverd voor deze platformen.
* Automatische testen zijn opgezet voor deze platformen, maar deze kunnen misschien niet gerund worden.
* Veranderingen in de taal zijn master worden goedgekeurd door platform bouwen. Voor sommige talen houdt dit in dat alleen de standaard bibliotheek gecompileerd moet worden voor andere Rustc en Cargo ook.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target | std | rustc | cargo | notes |
| aarch64-apple-ios | ✓ |  |  | ARM64 iOS |
| aarch64-fuchsia | ✓ |  |  | ARM64 Fuchsia |
| aarch64-linux-android | ✓ |  |  | ARM64 Android |
| aarch64-unknown-linux-gnu | ✓ | ✓ | ✓ | ARM64 Linux |
| aarch64-unknown-linux-musl | ✓ |  |  | ARM64 Linux with MUSL |
| arm-linux-androideabi | ✓ |  |  | ARMv7 Android |
| arm-unknown-linux-gnueabi | ✓ | ✓ | ✓ | ARMv6 Linux |
| arm-unknown-linux-gnueabihf | ✓ | ✓ | ✓ | ARMv6 Linux, hardfloat |
| arm-unknown-linux-musleabi | ✓ |  |  | ARMv6 Linux with MUSL |
| arm-unknown-linux-musleabihf | ✓ |  |  | ARMv6 Linux, MUSL, hardfloat |
| armv5te-unknown-linux-gnueabi | ✓ |  |  | ARMv5TE Linux |
| armv7-apple-ios | ✓ |  |  | ARMv7 iOS, Cortex-a8 |
| armv7-linux-androideabi | ✓ |  |  | ARMv7a Android |
| armv7-unknown-linux-gnueabihf | ✓ | ✓ | ✓ | ARMv7 Linux |
| armv7-unknown-linux-musleabihf | ✓ |  |  | ARMv7 Linux with MUSL |
| armv7s-apple-ios | ✓ |  |  | ARMv7 iOS, Cortex-a9 |
| asmjs-unknown-emscripten | ✓ |  |  | asm.js via Emscripten |
| i386-apple-ios | ✓ |  |  | 32-bit x86 iOS |
| i586-pc-windows-msvc | ✓ |  |  | 32-bit Windows w/o SSE |
| i586-unknown-linux-gnu | ✓ |  |  | 32-bit Linux w/o SSE |
| i586-unknown-linux-musl | ✓ |  |  | 32-bit Linux w/o SSE, MUSL |
| i686-linux-android | ✓ |  |  | 32-bit x86 Android |
| i686-unknown-freebsd | ✓ | ✓ | ✓ | 32-bit FreeBSD |
| i686-unknown-linux-musl | ✓ |  |  | 32-bit Linux with MUSL |
| mips-unknown-linux-gnu | ✓ | ✓ | ✓ | MIPS Linux |
| mips-unknown-linux-musl | ✓ |  |  | MIPS Linux with MUSL |
| mips64-unknown-linux-gnuabi64 | ✓ | ✓ | ✓ | MIPS64 Linux, n64 ABI |
| mips64el-unknown-linux-gnuabi64 | ✓ | ✓ | ✓ | MIPS64 (LE) Linux, n64 ABI |
| mipsel-unknown-linux-gnu | ✓ | ✓ | ✓ | MIPS (LE) Linux |
| mipsel-unknown-linux-musl | ✓ |  |  | MIPS (LE) Linux with MUSL |
| powerpc-unknown-linux-gnu | ✓ | ✓ | ✓ | PowerPC Linux |
| powerpc64-unknown-linux-gnu | ✓ | ✓ | ✓ | PPC64 Linux |
| powerpc64le-unknown-linux-gnu | ✓ | ✓ | ✓ | PPC64LE Linux |
| s390x-unknown-linux-gnu | ✓ | ✓ | ✓ | S390x Linux |
| sparc64-unknown-linux-gnu | ✓ |  |  | SPARC Linux |
| sparcv9-sun-solaris | ✓ |  |  | SPARC Solaris 10/11, illumos |
| wasm32-unknown-unknown | ✓ |  |  | WebAssembly |
| wasm32-unknown-emscripten | ✓ |  |  | WebAssembly via Emscripten |
| x86\_64-apple-ios | ✓ |  |  | 64-bit x86 iOS |
| x86\_64-fuchsia | ✓ |  |  | 64-bit Fuchsia |
| x86\_64-linux-android | ✓ |  |  | 64-bit x86 Android |
| x86\_64-rumprun-netbsd | ✓ |  |  | 64-bit NetBSD Rump Kernel |
| x86\_64-sun-solaris | ✓ |  |  | 64-bit Solaris 10/11, illumos |
| x86\_64-unknown-cloudabi | ✓ |  |  | 64-bit CloudABI |
| x86\_64-unknown-freebsd | ✓ | ✓ | ✓ | 64-bit FreeBSD |
| x86\_64-unknown-linux-gnux32 | ✓ |  |  | 64-bit Linux |
| x86\_64-unknown-linux-musl | ✓ |  |  | 64-bit Linux with MUSL |
| x86\_64-unknown-netbsd | ✓ | ✓ | ✓ | NetBSD/amd64 |
| x86\_64-unknown-redox | ✓ |  |  | Redox OS |

**Niveau 3**

Dit niveau is ondersteunt en kan gezien worden als “Gegarandeerd bouwend”, maar dan zonder beschikbare versies via Rustup. Er worden geen automatische testen gerund dus er wordt niet gegarandeerd dat de geproduceerd build werkt, maar meestal werken deze platformen best goed. Deze platformen hebben de volgende eigenschappen:

* Automatische testen zijn opgezet voor deze platformen, maar deze kunnen misschien niet gerund worden.
* Veranderingen in de taal zijn master worden goedgekeurd door platform bouwen. Voor sommige talen houdt dit in dat alleen de standaard bibliotheek gecompileerd moet worden voor andere Rustc en Cargo ook.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target | std | rustc | cargo | notes |
| aarch64-unknown-cloudabi | ✓ |  |  | ARM64 CloudABI |
| armv7-unknown-cloudabi-eabihf | ✓ |  |  | ARMv7 CloudABI, hardfloat |
| i686-unknown-cloudabi | ✓ |  |  | 32-bit CloudABI |
| powerpc-unknown-linux-gnuspe | ✓ |  |  | PowerPC SPE Linux |
| sparc-unknown-linux-gnu | ✓ |  |  | 32-bit SPARC Linux |

**Niveau 4**

De Rust codebase heft ondersteuning voor deze platformen, maar worden niet gebouwd of getest. Deze platformen kunnen misschien niet werken, verder zijn er geen officiële versies beschikbaar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target | std | rustc | cargo | notes |
| i686-pc-windows-msvc (XP) | ✓ |  |  | Windows XP support |
| i686-unknown-haiku | ✓ |  |  | 32-bit Haiku |
| i686-unknown-netbsd | ✓ |  |  | NetBSD/i386 with SSE2 |
| le32-unknown-nacl | ✓ |  |  | PNaCl sandbox |
| mips-unknown-linux-uclibc | ✓ |  |  | MIPS Linux with uClibc |
| mipsel-unknown-linux-uclibc | ✓ |  |  | MIPS (LE) Linux with uClibc |
| msp430-none-elf | \* |  |  | 16-bit MSP430 microcontrollers |
| sparc64-unknown-netbsd | ✓ | ✓ |  | NetBSD/sparc64 |
| thumbv6m-none-eabi | \* |  |  | Bare Cortex-M0, M0+, M1 |
| thumbv7em-none-eabi | \* |  |  | Bare Cortex-M4, M7 |
| thumbv7em-none-eabihf | \* |  |  | Bare Cortex-M4F, M7F, FPU, hardfloat |
| thumbv7m-none-eabi | \* |  |  | Bare Cortex-M3 |
| x86\_64-pc-windows-msvc (XP) | ✓ |  |  | Windows XP support |
| x86\_64-unknown-bitrig | ✓ | ✓ |  | 64-bit Bitrig |
| x86\_64-unknown-dragonfly | ✓ | ✓ |  | 64-bit DragonFlyBSD |
| x86\_64-unknown-haiku | ✓ |  |  | 64-bit Haiku |
| x86\_64-unknown-openbsd | ✓ | ✓ |  | 64-bit OpenBSD |
| [NVPTX](https://github.com/japaric/nvptx#targets) | \*\* |  |  | --emit=asm generates PTX code that runs on NVIDIA GPUs |

\* Dit zijn “bare-metal” microcontrollers doelen, deze hebben alleen toegang tot de kern bibliotheek en niet de standaard bibliotheek.

\*\* Er is back-end ondersteuning voor deze doelen, maar geen doelwit dat (nog) in rustc is ingebouwd. Je moet uw eigen doelspecificatiebestand schrijven. Deze doelen ondersteunen alleen de kernbibliotheek.

Dit zijn niet de enige platforms die Rust kan compileren! Dat zijn degenen met ingebouwde doeldefinities en/of standaard bibliotheekondersteuning. Tijdens het linken van de kernbibliotheek, kan Rust zich richten op "bare metal" in de x86-, ARM-, MIPS- en PowerPC-families, hoewel hiervoor mogelijk aangepaste doelspecificaties moeten worden gedefinieerd. Al dergelijke scenario's zijn niveau 4.

## Hoe werkt Rust intern op low- en highlevel?

## Nutteloos niet van belang voor dit onderzoek, mss memory menagment

## Vanuit welke talen leent rust concepten, wat dat zijn deze concepten?

Rust is een relatieve nieuwe taal en leent concepten uit al bestaande talen.   
Waarom leent Rust concepten uit andere talen? Omdat alles van de grond af zelf te maken is tijdsintensief en garandeert geen succes. De geleende concepten zijn bewezen goed te werken omdat ze al jaren in andere talen gebruikt worden.  
  
De eerste release van Rust gebruikte meer concepten dan nu. Deze concepten zijn weggevallen daarom staan dus niet in de onderstaande lijst. In de onderstaande lijst staan de huidige nog gebruikte concepten die geleend worden. Links staat de taal waarvan een of meerder concept(en) van geleend wordt. Rechts staan de concepten die geleend worden.

|  |  |
| --- | --- |
| Taal | Concepten |
| [SML](http://www.macs.hw.ac.uk/ultra/skalpel/html/sml.html), [OCaml](https://ocaml.org/) | algebraic data types,  pattern matching,  type inference,  semicolon statement separation |
| [C++](http://www.cplusplus.com/info/) | references,  Resource Acquisition Is Initialization (RAII),  smart pointers,  move semantics,  monomorphization,  memory model |
| [ML Kit](http://wiki.c2.com/?MlKit), [Cyclone](https://cyclone.thelanguage.org/) | region based memory management |
| [Haskell (GHC)](https://www.haskell.org/) | typeclasses, type families |
| [Newsqueak](http://dbpedia.org/page/Newsqueak), [Alef](https://en.wikipedia.org/wiki/Alef_(programming_language)), [Limbo](https://en.wikipedia.org/wiki/Limbo_(programming_language)) | channels, concurrency |
| [Erlang](https://www.erlang.org/) | message passing, thread failure |
| [Swift](https://developer.apple.com/swift/) | optional bindings |
| [Scheme](https://www.scheme.com/tspl4/) | hygienic macros |
| [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) | attributes |
| [Unicode Annex #31](http://www.unicode.org/reports/tr31/) | identifier and pattern syntax |

[(2)](#_Bronnen)

## Wat zijn de voor- en nadelen van Rust tegenover C++ in de scope van de lesstof van de opleiding HBO-ICT met afstudeerrichting Technische Informatica?

Een van de voordelen kan performance zijn. Hierdoor zouden grotere programma’s uitgevoerd kunnen worden op bestaande hardware (Denk hierbij aan [ARM Cortex M3](https://store.arduino.cc/arduino-due) chips die de Hogeschool Utrecht gebruikt.) of bestaande programma’s kunnen worden uitgevoerd op kleinere (minder snelle) hardware (denk hieraan de ATmega328 van de [Arduino Nano](https://store.arduino.cc/arduino-nano)). Dit zou geld kunnen besparen want de Arduino Due kost €35 [(3.0)](#_Bronnen) en de Arduino Nano kost €20. [(3.1)](#_Bronnen).

Korte uitleg over de data hier onder

* Source: in welke taal de code is uit gevoerd.
* Seconde: uitvoer tijd van de code.
* Geheugen: geheugen gebruikt tijdens de uitvoer (in bytes).
* Grootte: grootte van de source files (in bytes).
* CPU lading: hoeveel elke kern beladen werd tijdens de uitvoer, deze code is uitgevoerd op een quad core en daarom ook 4 percentages.

Er zijn 10 verschillende algoritmes geschreven voor 4 verschillende talen namelijk: C++, Rust, Python en Haskell. Deze zijn uitgevoerd dit zijn de resultaten. De source code van deze programma’s is te vinden in de appendix.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Reverse-complement](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/revcomp.html#revcomp) | | | | [(1.0)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 2,94 | 980.524 | 2.280 | 15% | 50% | 52% | 40% |
| Rust | 1,60 | 995.212 | 1.376 | 24% | 25% | 96% | 30% |
| Python 3 | 16,76 | 1.005.252 | 814 | 65% | 21% | 44% | 17% |
| Haskell | 5,67 | 501.336 | 1.020 | 26% | 12% | 12% | 40% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [N-body](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/nbody.html#nbody) | | | | [(1.1)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 9,42 | 1.712 | 1.763 | 100% | 1% | 2% | 0% |
| Rust | 13,25 | 1.808 | 1.805 | 0% | 0% | 1% | 100% |
| Python 3 | 14min | 8.212 | 1.196 | 91% | 0% | 1% | 9% |
| Haskell | 22,01 | 3.936 | 1.883 | 99% | 99% | 100% | 99% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [K-nucleotide](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/knucleotide.html#knucleotide) | | | | [(1.2)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 3,83 | 156.104 | 1.624 | 72% | 73% | 98% | 72% |
| Rust | 5,98 | 137.956 | 1.648 | 78% | 49% | 90% | 85% |
| Python 3 | 79,79 | 250.948 | 1.967 | 98% | 96% | 96% | 99% |
| Haskell | 35,51 | 604.996 | 1.486 | 89% | 93% | 90% | 87% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Regex-redux](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/regexredux.html#regexredux) | | | | [(1.3)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 1,83 | 203.680 | 1.315 | 49% | 87% | 57% | 51% |
| Rust | 2,44 | 194.804 | 765 | 85% | 41% | 20% | 16% |
| Python 3 | 15,56 | 439.964 | 512 | 25% | 92% | 32% | 32% |
| Haskell | Bad output |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Binary-trees | | | | [(1.4)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 3,67 | 118.620 | 809 | 75% | 78% | 99% | 76% |
| Rust | 4,14 | 175.692 | 721 | 90% | 90% | 91% | 100% |
| Python 3 | 92,72 | 448.844 | 589 | 87% | 90% | 96% | 87% |
| Haskell | 12,59 | 478.012 | 592 | 95% | 90% | 88% | 94% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Fasta](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/fasta.html#fasta) | | | | [(1.5)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 1,33 | 1.744 | 2.711 | 82% | 82% | 81% | 81% |
| Rust | 1,46 | 3.112 | 1.906 | 84% | 83% | 84% | 89% |
| Python 3 | 62,88 | 680.736 | 1.947 | 60% | 56% | 48% | 62% |
| Haskell | 9,46 | 4.980 | 969 | 100% | 2% | 2% | 3% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Spectral-norm](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/spectralnorm.html#spectralnorm) | | | | [(1.6)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 1,98 | 1.168 | 1.044 | 100% | 99% | 99% | 99% |
| Rust | 1,97 | 2.600 | 1.126 | 100% | 100% | 100% | 99% |
| Python 3 | 193,86 | 50.556 | 443 | 98% | 98% | 99% | 99% |
| Haskell | 4,17 | 3.812 | 987 | 99% | 97% | 99% | 98% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Fannkuch-redux](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/fannkuchredux.html#fannkuchredux) | | | | [(1.7)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 10,07 | 1.856 | 980 | 99% | 100% | 100% | 95% |
| Rust | 9,87 | 1.848 | 1.020 | 100% | 95% | 100% | 100% |
| Python 3 | 9 min | 48.052 | 950 | 99% | 100% | 97% | 100% |
| Haskell | 18,20 | 4.024 | 842 | 99% | 100% | 95% | 100% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Mandelbrot](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/mandelbrot.html#mandelbrot) | | | | [(1.8)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 1,82 | 29.092 | 1.002 | 98% | 97% | 97% | 100% |
| Rust | 1,74 | 33.712 | 1.332 | 98% | 100% | 98% | 98% |
| Python 3 | 279,86 | 49.334 | 688 | 100% | 100% | 100% | 100% |
| Haskell | 11,60 | 38,744 | 782 | 100% | 100% | 100% | 100% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Pidigits](https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/pidigits.html#pidigits) | | | | [(1.9)](#_Bronnen) | | | |
| Source | Seconde | Geheugen | Grootte | CPU lading | | | |
| C++ | 1,89 | 4,312 | 513 | 1% | 1% | 99% | 1% |
| Rust | 1,74 | 4.520 | 1.366 | 1% | 3% | 0% | 99% |
| Python 3 | 3,51 | 10.500 | 385 | 1% | 1% | 0% | 100% |
| Haskell | 4,20 | 9.724 | 585 | 27% | 8% | 8% | 81% |

### Hoe is dit gemeten?

Deze testen zijn uitgevoerd op een computer met:

Processor: Intel Q6600 quad-core op 2,4Ghz

RAM: 4GB ram.

OS: Ubuntu™ 18.10 Linux x64 4.18.0-10-generic.

Elk programma is uitgevoerd en gemeten bij de kleinste input waarde, de output werd opgeslagen in een bestand om vergeleken te worden met de verwachte output. Zolang de output overeen blijft komen met de verwachte output werd het programma getest met een grotere input. Totdat alle inputs waren doorlopen. Als het programma de verwachte output bleef geven tijdens het testen (binnen een redelijke tijd) dan werd er nog 5 keer gemeten. Tijdens deze testen werd de output geledigd (/dev/null).

Als het programma niet de verwachte output leverde binnen een redelijke tijd dan werd het programma geforceerd gestopt. Als de metingen met kleinere input wel de verwachte uitvoer leverde dan werd deze 5 keer gemeten. Ook tijdens deze testen werd de output geledigd /dev/null). [(1.10)](#_Bronnen)

### Hoe zijn de programma’s getimed?

Elk programma is uitgevoerd als een “Child-process” van een Python script.  
Er werd gebruik gemaakt van “time.time()” voordat het proces werd gestart en na dat deze gesloten was. [(1.10)](#_Bronnen)

### Hoe is het geheugen gebruik gemeten?

Door “GLIBTOP\_PROC\_MEM\_RESIDENT” te samplen voor het hoofd proces en zijn “Child-process” elke 0,2 seconde. [(1.10)](#_Bronnen)

### Hoe is de source file gemeten?

De code werd in een bestand gezet zonder commentaar, extra spaties en extra witregels. Op dit bestand werd een lichte GZip compressie uitgevoerd. De grootte van dit bestand werd gemeten. [(1.10)](#_Bronnen)

### Hoe is de CPU lading gemeten?

De GTop CPU inactief en de GTop CPU totaal werden gemeten voor de opstarten van het “Child-process” en na het afsluiten hiervan. De percentages zijn het totaal min de inactieve waardes. [(1.10)](#_Bronnen)

### Conclusie performance

**Uitvoer snelheid in seconde**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Taal | RC | NB | KN | RR | BT | FA | SN | FR | MB | PD |
| Rust | 1,60 | 13,25 | 5,98 | 2,44 | 4,14 | 1,46 | 1,97 | 9,87 | 1,74 | 1,74 |
| C++ | 2,94 | 9,42 | 3,83 | 1,83 | 3,67 | 1,33 | 1,98 | 10,07 | 1,82 | 1,89 |

Uit dit tabel is te halen dat rust en c++ soms veel en soms weinig van elkaar afzitten met snelheid en dat beide talen 5 keer de snelste waren. Maar als we wat gaan rekenen dan blijkt dat als C++ sneller is dat het dan gemiddeld is met ongeveer 22% en als Rust sneller is dan is dat gemiddeld ongeveer 12%.

**Geheugen gebruik in bytes**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Taal | RC | NB | KN | RR | BT | FA | SN | FR | MB | PD |
| Rust | 995.212 | 1.808 | 137.956 | 194.804 | 175.692 | 3.112 | 2.600 | 1.848 | 33.712 | 4.520 |
| C++ | 980.524 | 1.712 | 156.104 | 203.680 | 118.620 | 1.744 | 1.168 | 1.856 | 29.092 | 3.212 |

Als we kijken naar het tabel van geheugengebruik dan blijkt dat het gebruik hoger light dan die van C++, met ongeveer 15%. Dat is een vors verschil zeker voor microcontrollers. Bijvoorbeeld de Arduino Due die de hogenschool gebruikt, deze microcontroller heeft maar 96 KB SRAM (twee banken van 64KB en 32KB) [(3.0)](#_Bronnen). Dit is al weinig geheugen, dus elke byte telt hier.

De source file grootte van Rust ligt gemiddeld ook boven die van C++

C++ heeft een gemiddelde grootte van 1044 bytes en Rust heeft een gemiddelde grootte 1319 bytes[(1.10)](#_Bronnen). Rust geeft ongeveer 26% grotere bestande.

In het kort: Rust is gemiddeld 10% langzamer dan C++, gebruikt gemiddeld 15% meer geheugen en geeft gemiddeld 26% grotere source files.

## Wat zou een student leren van Rust tegenover C++ scope van de opleiding HBO-ICT met afstudeerrichting Technische Informatica?

## Waar zou de taal Rust een plek hebben in de opleiding HBO-ICT met afstudeerrichting Technische Informatica?

Leesbaarheid van code bij mensen testen.

Leesbaarheid van code bij ti’ers testen.

En verband trekken met sourcefile size

## Bronnen

1.0: Reverse-complement | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/revcomp.html>

1.1: N-body | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/nbody.html>

1.2: K-nucleotide | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/knucleotide.html>

1.3 Regex-redux | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/regexredux.html>

1.4: Binary-trees | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/binarytrees.html>

1.5: Fasta | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/fasta.html>

1.6: Spectral-norm | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/spectralnorm.html>

1.7: Fannkuch-redux | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/fannkuchredux.html>

1.8: Mandelbrot | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/mandelbrot.html>

1.9: Pidigits | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/pidigits.html>

1.10: How programs are measured | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van <https://benchmarksgame-team.pages.debian.net/benchmarksgame/how-programs-are-measured.html>

2.0: Appendix: Influences - The Rust Reference. (z.d.). Geraadpleegd op 27 november 2018,  
 van <https://doc.rust-lang.org/reference/influences.html>

2.1: Rust Platform Support · The Rust Programming Language. (z.d.). Geraadpleegd op 27 november   
 2018, van <https://forge.rust-lang.org/platform-support.html>

3.0: Arduino Due. (z.d.). Geraadpleegd op 27 november 2018, van   
 <https://store.arduino.cc/arduino-due>

3.1: Arduino Nano. (z.d.). Geraadpleegd op 27 november 2018, van   
 <https://store.arduino.cc/arduino-nano>

5

6

7

8

9

10

11

12

13

14