Revisiting the survey of current Python implementations

Feng Li (李枫)
hkli2013@126.com
May 26，2022

Agenda

I. Background

- Overview
- Ranking
- Ecosystem
- Limitations
- Tech Stack

II. C-based implementation

- Overview
- Make CPython faster
- Cinder

III. Java-based implementation

- Overview
- Jython
- GraalPython

IV. LLVM-based implementation

- Overview
- Pyston

V. DotNet-based implementation

- Overview
- IronPython
- Pyjion

VI. Wasm-based implementation

- Overview
- Pyodide
- Wasmer Python
- PyScript

VII. Rust-based implementation

- Overview
- RustPython

VIII. RPython-based implementation

- Overview
- PyPy

IX. Tittle-tattle

- Vendor-specific Distributions
- Derivatives

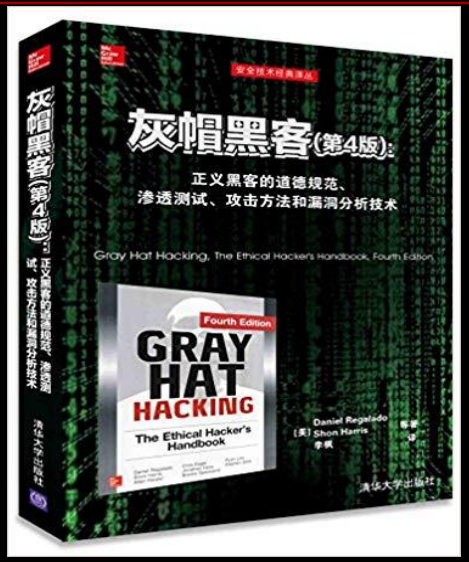
X. Add-ons

- Dev Env
- Benchmarking
- Profiling
- Debugging
- Notebook
- Serverless
- Interoperability
- Gaming
- Full Stack

XI. Wrap-up

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years(~11 years on Mobile Dev)
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. Background

1) Overview

- https://en.wikipedia.org/wiki/Python_(programming_language)

Python was conceived in the late 1980s[39] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL,[40] capable of exception handling and interfacing with the Amoeba operating system.[11] Its implementation began in December 1989.[41] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "Benevolent Dictator For Life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[42] In January 2019, active Python core developers elected a five-member "Steering Council" to lead the project.[43][44]

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode.[45]

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible.[46] Many of its major features were backported to Python 2.6.x[47] and 2.7.x version series. Releases of Python 3 include the `2to3` utility, which automates the translation of Python 2 code to Python 3.[48]

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[49][50] No more security patches or other improvements will be released for it.[51][52] With Python 2's end-of-life, only Python 3.6.x[53] and later are supported.

Python 3.9.2 and 3.8.8 were expedited[54] as all versions of Python (including 2.7[55]) had security issues, leading to possible remote code execution[56] and web cache poisoning.[57]

Design philosophy and features

-

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming[58] and metaobjects (magic methods)).[59] Many other paradigms are supported via extensions, including design by contract[60][61] and logic programming.[62]

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management.[63] It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter`, `map` and `reduce` functions; list comprehensions, dictionaries, sets, and generator expressions.[64] The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.[65]

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:[66]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.[39] It is often described as a "batteries included" language due to its comprehensive standard library.[67]

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one— and preferably only one —obvious way to do it" design philosophy.[66] Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is *not* considered a compliment in the Python culture."[68]

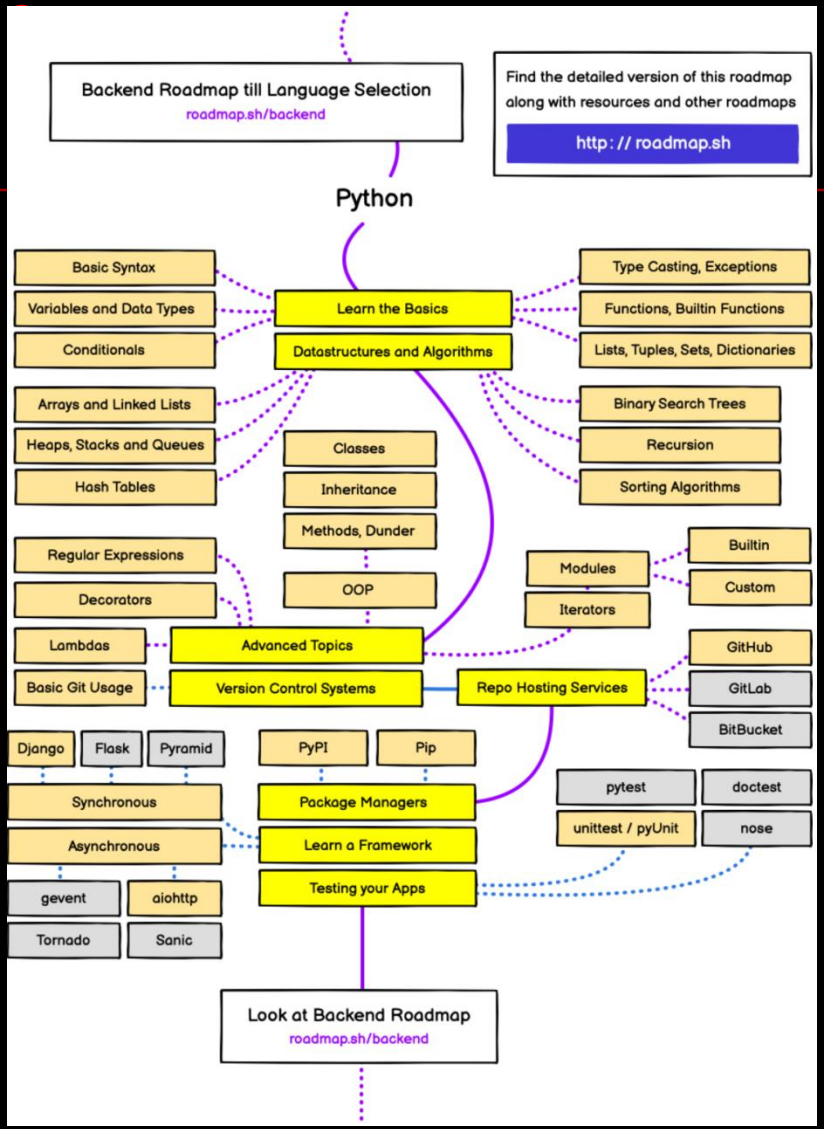
Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[69] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python's developers aim for the language to be fun to use. This is reflected in its name—a tribute to the British comedy group Monty Python[70]—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a Monty Python sketch) instead of the standard foo and bar.[71][72]

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.[73][74]

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonistas*.[75][76]

Developer Roadmaps



Source: https://roadmap.sh/python

2) Ranking

TIOBE

- https://www.tiobe.com/tiobe-index/

May 2022	May 2021	Change	Programming Language	Ratings	Change
1	2	↑	Python	12.74%	+0.86%
2	1	↓	C	11.59%	-1.80%
3	3		Java	10.99%	-0.74%
4	4		C++	8.83%	+1.01%
5	5		C#	6.39%	+1.98%
6	6		Visual Basic	5.86%	+1.85%
7	7		JavaScript	2.12%	-0.33%
8	8		Assembly language	1.92%	-0.51%
9	10	↑	SQL	1.87%	+0.16%
10	9	↓	PHP	1.52%	-0.34%

PYPL

- http://pypl.github.io/PYPL.html

Worldwide, May 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	27.85 %	-2.5 %
2		Java	17.86 %	-0.1 %
3		JavaScript	9.17 %	+0.4 %
4		C#	7.62 %	+0.7 %
5		C/C++	7.0 %	+0.4 %
6		PHP	5.36 %	-1.0 %
7		R	4.34 %	+0.5 %
8	↑↑↑	TypeScript	2.39 %	+0.7 %
9	↓	Objective-C	2.25 %	+0.0 %
10		Swift	2.05 %	+0.3 %

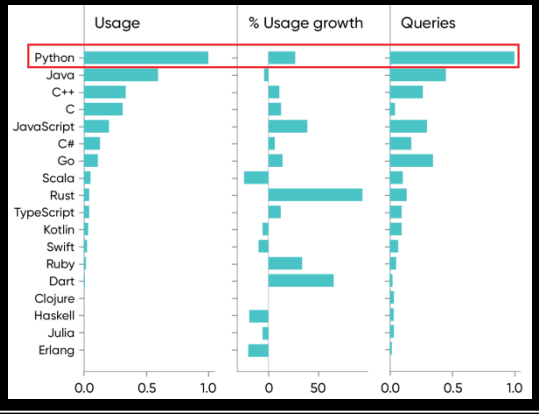
IEEE Spectrum

- **https://spectrum.ieee.org/top-programming-languages/**

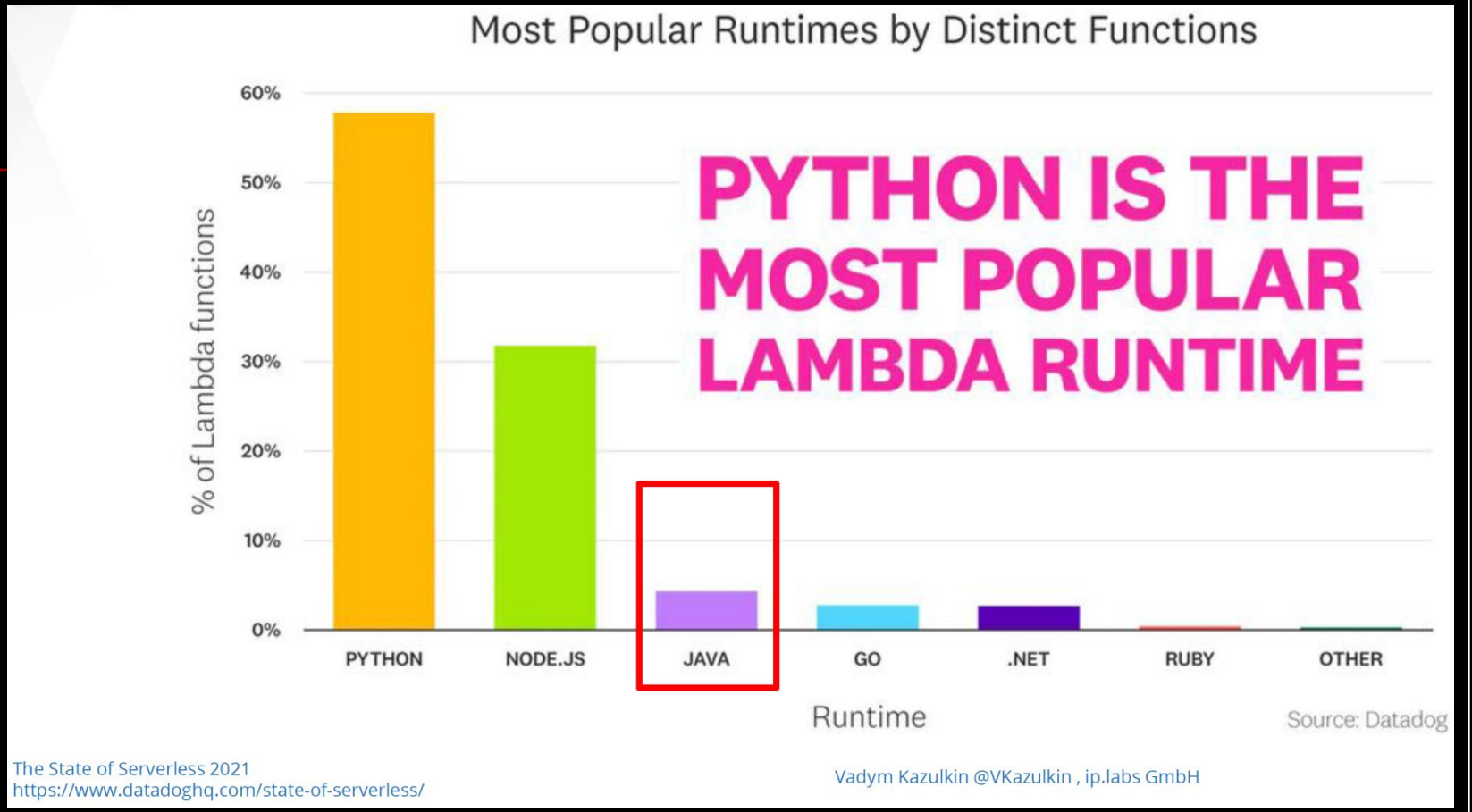
Rank	Language	Type	Score
1	Python		100.0
2	Java		95.4
3	C		94.7
4	C++		92.4
5	JavaScript		88.1
6	C#		82.4
7	R		81.7
8	Go		77.7
9	HTML		75.4
10	Swift		70.4

O'Reilly

- **https%3A//www.oreilly.com/radar/where-programming-ops-ai-and-the-cloud-are-headed-in-2021/**



AWS Lambda



The State of Serverless 2021
https://www.datadoghq.com/state-of-serverless/

...

3) Ecosystem

- https://www.developintelligence.com/the-python-ecosystem-of-2020/
- https://deprogrammaticaipsum.com/the-state-of-python-in-2021/
- …

Famous Python Projects

Build: Meson, SCons, mx…

DevOps: Ansible, SaltStack…

Web: Django, web2py, Flask, Tornado, Pylons, TurboGears, Quixote, Sanic… FastAPI…

Websites: Youtube, Quora, Reddit…

AI: Scikit-learn, Keras, Theano…

Data Analytics: PyData, PySpark…

Scientific Computing: Scipy, Sage…

HPC: Anaconda, CUDA Python…

Cloud/Serverless: OpenStack, LocalStack…

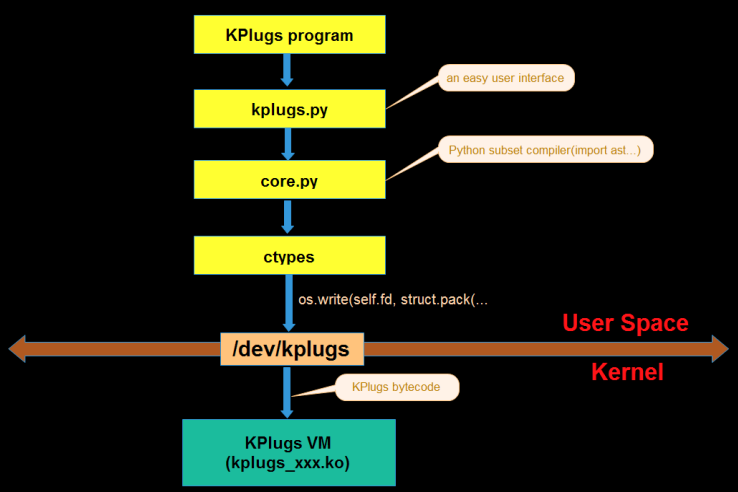
…

4) Limitations

- **Not a system language**

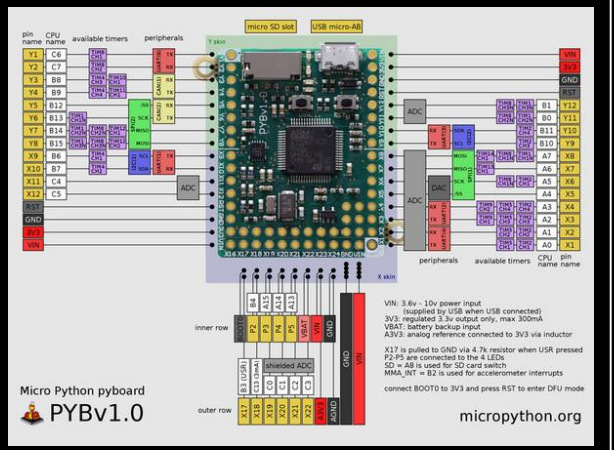
  So it is not a good fit for kernel space or infrastructure development.

  But this is not entirely true:



openstack.

Original author(s)	Rackspace Hosting and NASA
Developer(s)	Open Infrastructure Foundation and community
Initial release	21 October 2010; 11 years ago
Stable release	Yoga[1] / 30 March 2022; 38 days ago
Repository	opendev.org /openstack
Written in	Python
Platform	Cross-platform
Type	Cloud computing
License	Apache License 2.0
Website	www.openstack.org



and more…

- **Performance**

https://github.com/kostya/benchmarks

Base64

Testing base64 encoding/decoding of the large blob into the newly allocated buffers.

Base64

Language	Time, s	Memory, MiB	Energy, J
C/gcc (aklomp)	$0.143_{\pm 0.003}$	$1.89_{\pm 00.03} + 0.00_{\pm 00.00}$	$3.48_{\pm 00.12}$
Rust	$1.117_{\pm 0.018}$	$2.46_{\pm 00.09} + 0.07_{\pm 00.06}$	$24.21_{\pm 01.51}$
C/gcc	$1.181_{\pm 0.034}$	$1.87_{\pm 00.05} + 0.00_{\pm 00.00}$	$24.32_{\pm 01.36}$
V/clang	$1.253_{\pm 0.038}$	$1.69_{\pm 00.06} + 0.17_{\pm 00.12}$	$27.29_{\pm 01.44}$
Nim/gcc	$1.344_{\pm 0.036}$	$2.26_{\pm 00.04} + 4.44_{\pm 00.00}$	$24.70_{\pm 01.26}$
V/gcc	$1.419_{\pm 0.037}$	$2.12_{\pm 00.06} + 0.00_{\pm 00.00}$	$31.28_{\pm 01.15}$
Nim/clang	$1.730_{\pm 0.022}$	$2.76_{\pm 00.06} + 4.38_{\pm 00.00}$	$31.06_{\pm 01.47}$
Crystal	$1.731_{\pm 0.003}$	$3.75_{\pm 00.02} + 1.84_{\pm 00.05}$	$36.14_{\pm 00.19}$
D/ldc2	$1.978_{\pm 0.009}$	$3.57_{\pm 00.04} + 3.67_{\pm 00.00}$	$35.56_{\pm 00.80}$
Vala/gcc	$2.088_{\pm 0.028}$	$0.00_{\pm 00.00} + 0.00_{\pm 00.00}$	$46.61_{\pm 01.25}$
Ruby (--jit)	$2.094_{\pm 0.076}$	$14.47_{\pm 00.04} + 54.36_{\pm 01.33}$	$44.47_{\pm 04.35}$
Ruby	$2.142_{\pm 0.064}$	$14.48_{\pm 00.03} + 54.75_{\pm 01.87}$	$39.50_{\pm 03.31}$
Java	$2.164_{\pm 0.062}$	$38.51_{\pm 00.32} + 318.97_{\pm 39.40}$	$49.59_{\pm 01.34}$
Vala/clang	$2.239_{\pm 0.043}$	$0.00_{\pm 00.00} + 0.00_{\pm 00.00}$	$39.40_{\pm 00.88}$
Kotlin	$2.332_{\pm 0.063}$	$40.71_{\pm 00.14} + 335.77_{\pm 10.77}$	$49.36_{\pm 03.99}$
Scala	$2.364_{\pm 0.040}$	$53.91_{\pm 00.07} + 337.67_{\pm 13.53}$	$54.24_{\pm 02.59}$
Go	$2.587_{\pm 0.005}$	$4.43_{\pm 00.01} + 5.25_{\pm 00.12}$	$51.93_{\pm 00.40}$
C++/g++ (libcrypto)	$2.733_{\pm 0.108}$	$5.55_{\pm 00.12} + 0.07_{\pm 00.00}$	$55.73_{\pm 05.46}$
Node.js	$2.814_{\pm 0.064}$	$32.30_{\pm 00.08} + 36.36_{\pm 00.07}$	$60.55_{\pm 02.04}$
Perl (MIME::Base64)	$2.896_{\pm 0.030}$	$14.11_{\pm 00.03} + 0.08_{\pm 00.00}$	$63.67_{\pm 01.34}$
PHP	$2.960_{\pm 0.049}$	$15.72_{\pm 00.14} + 0.00_{\pm 00.00}$	$50.69_{\pm 01.54}$
Go/gccgo	$3.233_{\pm 0.021}$	$23.39_{\pm 00.25} + 8.76_{\pm 00.33}$	$66.18_{\pm 00.67}$
D/gdc	$3.295_{\pm 0.102}$	$7.04_{\pm 00.02} + 3.52_{\pm 00.00}$	$69.16_{\pm 03.88}$
D/dmd	$3.718_{\pm 0.055}$	$3.80_{\pm 00.07} + 3.61_{\pm 00.00}$	$73.01_{\pm 02.23}$
Python	$3.971_{\pm 0.063}$	$9.96_{\pm 00.03} + 0.18_{\pm 00.00}$	$90.62_{\pm 01.60}$
Zig	$4.513_{\pm 0.010}$	$1.88_{\pm 00.02} + 0.34_{\pm 00.03}$	$82.21_{\pm 01.48}$
Python/pypy	$4.886_{\pm 0.142}$	$65.43_{\pm 00.06} + 45.75_{\pm 00.09}$	$98.29_{\pm 08.01}$
Tcl	$4.998_{\pm 0.026}$	$4.88_{\pm 00.06} + 0.19_{\pm 00.03}$	$84.07_{\pm 02.10}$
Julia	$5.740_{\pm 0.202}$	$218.99_{\pm 00.04} + 63.02_{\pm 00.17}$	$128.13_{\pm 09.62}$
F#/.NET Core	$5.765_{\pm 0.045}$	$37.22_{\pm 00.04} + 36.42_{\pm 08.18}$	$106.41_{\pm 01.19}$
C#/.NET Core	$5.847_{\pm 0.015}$	$34.66_{\pm 00.04} + 40.93_{\pm 07.60}$	$108.42_{\pm 01.65}$
Ruby/truffleruby (--jvm)	$6.302_{\pm 0.210}$	$625.38_{\pm 09.77} + 141.33_{\pm 42.46}$	$137.95_{\pm 11.33}$
C#/Mono	$7.259_{\pm 0.059}$	$20.84_{\pm 00.03} + 18.49_{\pm 00.07}$	$174.00_{\pm 01.80}$
Ruby/jruby	$11.868_{\pm 0.308}$	$184.35_{\pm 02.57} + 138.17_{\pm 15.24}$	$266.12_{\pm 09.10}$
Perl (MIME::Base64::Perl)	$15.947_{\pm 0.181}$	$15.51_{\pm 00.03} + 0.16_{\pm 00.06}$	$362.72_{\pm 04.38}$
Ruby/truffleruby	$20.507_{\pm 0.491}$	$427.34_{\pm 01.68} + 320.53_{\pm 02.44}$	$367.59_{\pm 20.16}$

...

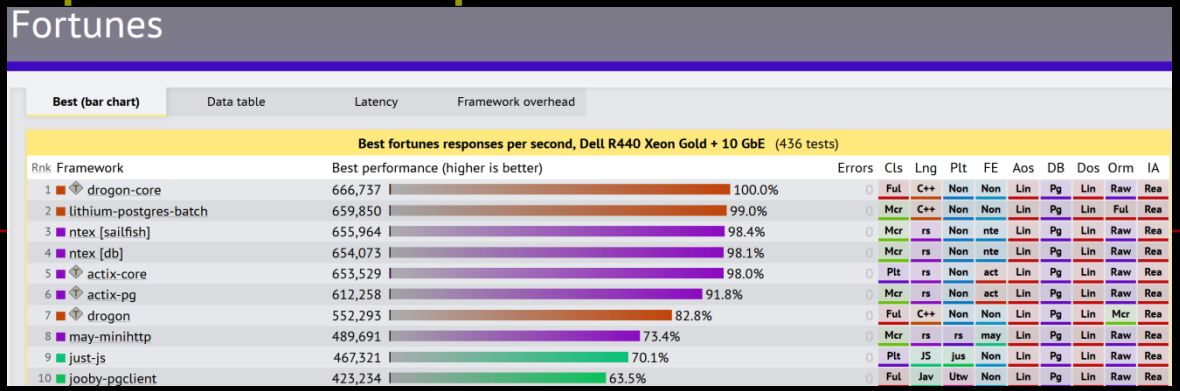
Environment

CPU: Intel(R) Core(TM) i7-10710U

Base Docker image: Debian GNU/Linux bookworm/sid

Language	Version
.NET Core	6.0.201
C#/.NET Core	4.1.0-5.22116.13 (dbffaa4a)
C#/Mono	6.12.0.122
Chez Scheme	9.5.4
Clojure	"1.11.0"
Crystal	1.3.2
D/dmd	v2.099.0
D/gdc	12.0.1
D/ldc2	1.28.1
Elixir	1.12.2
F#/.NET Core	12.0.1.0 for F# 6.0
Go	go1.18
Go/gccgo	12.0.1
Haskell	9.0.1
Java	18
Julia	v"1.7.2"
Kotlin	1.6.20
Lua	5.4.4
Lua/luajit	2.1.0-beta3
MLton	20210117
Nim	1.6.4
Node.js	v17.8.0
OCaml	4.13.1
PHP	8.1.2
Perl	v5.34.0
Python	3.9.12
Python/pypy	7.3.9-final0 for Python 3.9.12
Racket	"8.4"
Ruby	3.1.1p18
Ruby/jruby	9.3.4.0
Ruby/truffleruby	22.0.0.2
Rust	1.59.0
Scala	3.1.1
Swift	5.6
Tcl	8.6
V	0.2.4 509367b
Vala	0.56.0
Zig	0.9.1
clang/clang++	13.0.1
gcc/g++	12.0.1

- https://www.techempower.com/benchmarks/

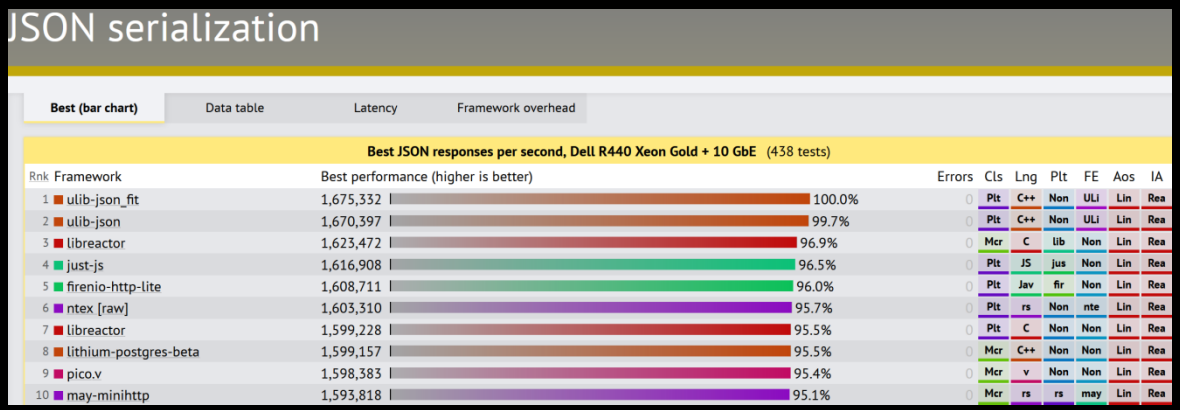


Fortunes

Best (bar chart) | Data table | Latency | Framework overhead

Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (436 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	drogon-core	666,737 100.0%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
2	lithium-postgres-batch	659,850 99.0%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
3	ntex [sailfish]	655,964 98.4%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
4	ntex [db]	654,073 98.1%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
5	actix-core	653,529 98.0%	0	Plt	rs	Non	act	Lin	Pg	Lin	Raw	Rea
6	actix-pg	612,258 91.8%	0	Mcr	rs	Non	act	Lin	Pg	Lin	Raw	Rea
7	drogon	552,293 82.8%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Mcr	Rea
8	may-minihttp	489,691 73.4%	0	Mcr	rs	rs	may	Lin	Pg	Lin	Raw	Rea
9	just-js	467,321 70.1%	0	Plt	JS	jus	Non	Lin	Pg	Lin	Raw	Rea
10	jooby-pgclient	423,234 63.5%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea

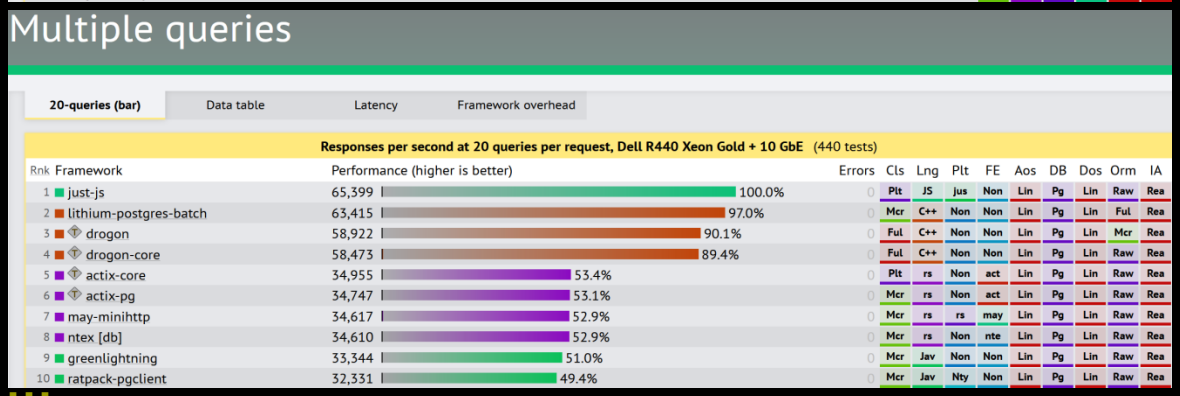


JSON serialization

Best (bar chart) | Data table | Latency | Framework overhead

Best JSON responses per second, Dell R440 Xeon Gold + 10 GbE (438 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	IA
1	ulib-json_fit	1,675,332 100.0%	0	Plt	C++	Non	ULi	Lin	Rea
2	ulib-json	1,670,397 99.7%	0	Plt	C++	Non	ULi	Lin	Rea
3	libreactor	1,623,472 96.9%	0	Mcr	C	lib	Non	Lin	Rea
4	just-js	1,616,908 96.5%	0	Plt	JS	jus	Non	Lin	Rea
5	firenio-http-lite	1,608,711 96.0%	0	Plt	Jav	fir	Non	Lin	Rea
6	ntex [raw]	1,603,310 95.7%	0	Plt	rs	Non	nte	Lin	Rea
7	libreactor	1,599,228 95.5%	0	Plt	C	Non	Non	Lin	Rea
8	lithium-postgres-beta	1,599,157 95.5%	0	Mcr	C++	Non	Non	Lin	Rea
9	pico.v	1,598,383 95.4%	0	Mcr	v	Non	Non	Lin	Rea
10	may-minihttp	1,593,818 95.1%	0	Mcr	rs	rs	may	Lin	Rea



Multiple queries

20-queries (bar) | Data table | Latency | Framework overhead

Responses per second at 20 queries per request, Dell R440 Xeon Gold + 10 GbE (440 tests)

Rnk	Framework	Performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	just-js	65,399 100.0%	0	Plt	JS	jus	Non	Lin	Pg	Lin	Raw	Rea
2	lithium-postgres-batch	63,415 97.0%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
3	drogon	58,922 90.1%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Mcr	Rea
4	drogon-core	58,473 89.4%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
5	actix-core	34,955 53.4%	0	Plt	rs	Non	act	Lin	Pg	Lin	Raw	Rea
6	actix-pg	34,747 53.1%	0	Mcr	rs	Non	act	Lin	Pg	Lin	Raw	Rea
7	may-minihttp	34,617 52.9%	0	Mcr	rs	rs	may	Lin	Pg	Lin	Raw	Rea
8	ntex [db]	34,610 52.9%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
9	greenlightning	33,344 51.0%	0	Mcr	Jav	Non	Non	Lin	Pg	Lin	Raw	Rea
10	ratpack-pgclient	32,331 49.4%	0	Mcr	Jav	Nty	Non	Lin	Pg	Lin	Raw	Rea

How to break the ice

- use a subset of Python, a variant, or a Python-based DSL for system programming
- customize a special Python runtime
- https://en.wikipedia.org/wiki/Source-to-source_compiler
- …

5) Tech Stack

- Managed programming language
- Dynamic programming language

Runtime

- https://en.wikipedia.org/wiki/Runtime_system
- https://en.wikipedia.org/wiki/Register_machine
- https://en.wikipedia.org/wiki/Stack_machine
- https://en.wikipedia.org/wiki/Intermediate_representation
- https://en.wikipedia.org/wiki/Bytecode
- https://en.wikipedia.org/wiki/Compiler
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://en.wikipedia.org/wiki/Ahead-of-time_compilation
- https://en.wikipedia.org/wiki/Source-to-source_compiler
- https://en.wikipedia.org/wiki/Interpreter_(computing)
- …
- https://en.wikipedia.org/wiki/Polyglot_(computing)

  Polyglot — use the best tool for the right jobs: high performance, scripting, web, functional programming, etc

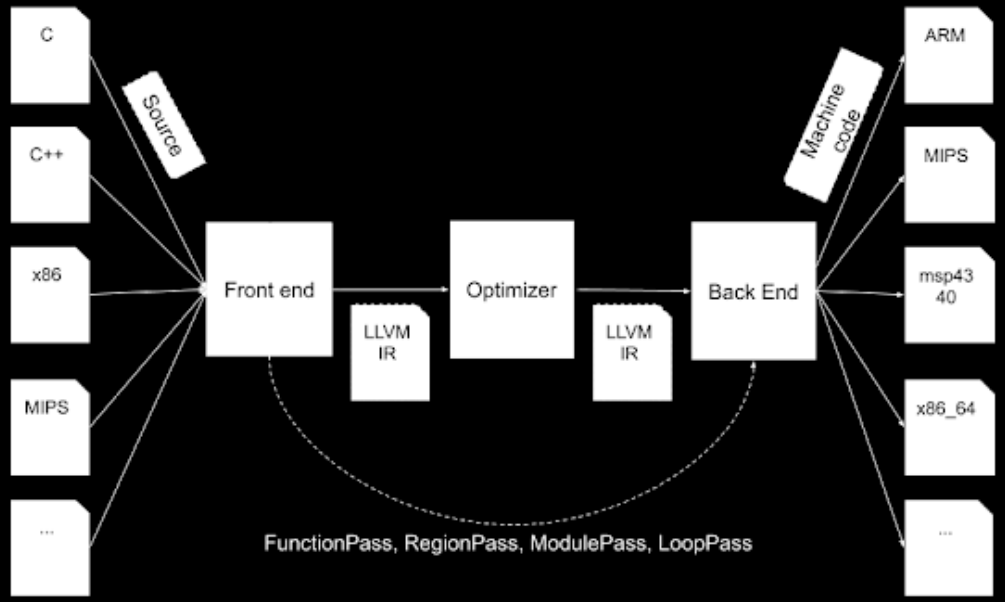
  The most popular Polyglot Runtimes: GraalVM, .Net, WASM…

5.1 LLVM

- https://en.wikipedia.org/wiki/LLVM

LLVM is a set of compiler and toolchain technologies,[5] which can be used to develop a front end for any programming language and a back end for any instruction set architecture. LLVM is designed around a language-independent intermediate representation (IR) that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.[6]

- https://llvm.org
- http://clang.llvm.org/
-



Source: http://blog.k3170makan.com/2020/04/learning-llvm-i-introdution-to-llvm.html

- https://llvm.org/docs/
- https://www.llvm.org/ProjectsWithLLVM/

IR

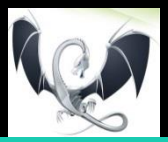
The core of LLVM is the intermediate representation (IR), a low-level programming language similar to assembly. IR is a strongly typed reduced instruction set computing (RISC) instruction set which abstracts away most details of the target. For example, the calling convention is abstracted through *call* and *ret* instructions with explicit arguments. Also, instead of a fixed set of registers, IR uses an infinite set of temporaries of the form %0, %1, etc. LLVM supports three equivalent forms of IR: a human-readable assembly format, an in-memory format suitable for frontends, and a dense bitcode format for serializing. A simple "Hello, world!" program in the IR format:[32]

```
@.str = internal constant [14 x i8] c"hello, world\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], [14 x i8]* @.str, i32 0, i32 0
    %tmp2 = call i32 (i8*, ...) @printf( i8* %tmp1 ) nounwind
    ret i32 0
}
```

- https://llvm.org/docs/LangRef.html
- MLIR:
  https://mlir.llvm.org/
  A hybrid IR which can support multiple different requirements in a unified infrastructure.

- Bitcode:
  https://llvm.org/docs/BitCodeFormat.html
  https://llvm.org/docs/CommandGuide/llvm-bcanalyzer.html
  https://zhuanlan.zhihu.com/p/308201373

LLVM vs GCC



GPL v3	UIUC, MIT
Front-end: CC1 / CPP	Front-end: Clang
ld.bfd / ld.gold	lld / mclinker
gdb	lldb
as / objdump	MC layer
glibc	llvm-libc?
libstdc++	libc++
libsupc++	libc++abi
libgcc	libcompiler-rt
libgccjit	libLLVMMCJIT
. . .	ORC JIT, Coroutines, Clangd, libclc, Falcon...

- http://lld.llvm.org/
- https://llvm.org/docs/Proposals/LLVMLibC.html

5.2 WASM

- https://en.wikipedia.org/wiki/WebAssembly

C source code and corresponding WebAssembly

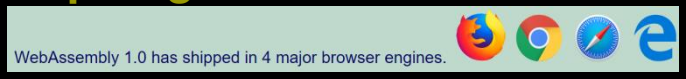
C source code	WebAssembly `.wat` text format	WebAssembly `.wasm` binary format

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`(func (param i64) (result i64)`
	`  local.get 0`
	`  i64.eqz`
	`  if (result i64)`
	`  i64.const 1`
	`  else`
	`  local.get 0`
	`  local.get 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`  end)`
		`00 61 73 6D 01 00 00 00`
		`01 00 01 60 01 73 01 73 06`
		`03 00 01 00 02`
		`0A 00 01`
		`00 00`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`
		`0B 15 17`

- https://webassembly.org/

  WebAssembly(abbr. *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

- https://github.com/WebAssembly/design

- 

- https://webassembly.org/specs/

Limitations

-

1. In general, WebAssembly does not allow direct interaction with the DOM. All interaction must flow through JavaScript interop.
2. Multithreading (although there are plans to address this.)
3. Garbage collection (although there are plans to address this.)
4. Security considerations (discussed below)

WebAssembly is supported on desktops, and mobile, but on the latter, in practice (for non-small memory allocations, such as with Unity game engine) there are "grave limitations that make many applications infeasible to be *reliably* deployed on mobile browsers [..] Currently allocating more than ~300MB of memory is not reliable on Chrome on Android without resorting to Chrome-specific workarounds, nor in Safari on iOS."[62]

All major web browsers allow WebAssembly if Content-Security-Policy is not specified, or if "unsafe-eval" is used, but otherwise the major web browsers behave differently.[63] In practice WebAssembly can't be used on Chrome without "unsafe-eval",[64][65] while a worker thread workaround is available.[66]

Runtime Implementations

- While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts.[37][38] Since WebAssembly's runtime environments (RE) are low level virtual stack machines (akin to JVM or Flash VM) that can be embedded into host applications, some of them have found a way to standalone runtime environments like Wasmtime and Wasmer.[8][13]

  Web browsers [edit]

  In November 2017, Mozilla declared support "in all major browsers",[39] after WebAssembly was enabled by default in Edge 16.[40] The support includes mobile web browsers for iOS and Android. As of July 2021, 94% of installed browsers support WebAssembly.[41] But for older browsers, Wasm can be compiled into asm.js by a JavaScript polyfill.[42]

Compilers:

Because WebAssembly executables are precompiled, it is possible to use a variety of programming languages to make them.[43] This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm. There have been around 40 programming languages reported to support Wasm as a compilation target.[44]

Emscripten compiles C and C++ to Wasm[32] using the Binaryen and LLVM as backend.[45]

As of version 8, a standalone Clang can compile C and C++ to Wasm.[46]

Its initial aim is to support compilation from C and C++,[47] though support for other source languages such as Rust, .NET languages[48][49][44] and AssemblyScript[50] (TypeScript-like) is also emerging. After the MVP release, there are plans to support multithreading and garbage collection[51][52] which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor), F# (supported via Bolero[53] with help of Blazor), Python, and even JavaScript where the browser's just-in-time compilation speed is considered too slow. A number of other languages have some support including Python,[54] Java,[55] Julia,[56][57][58] Zig,[59] and Ruby,[60] as well as Go.[61]

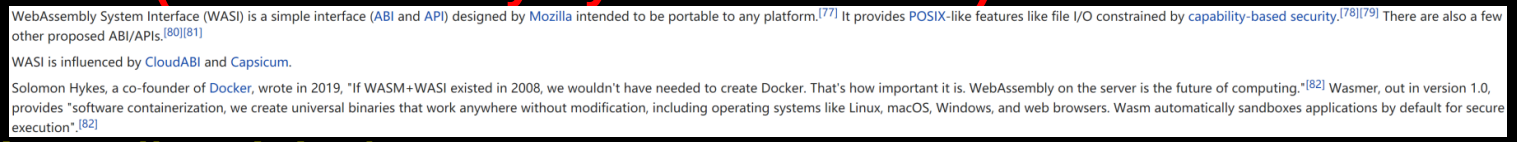
- **https://github.com/appcypher/awesome-wasm-langs**
- **https://github.com/appcypher/awesome-wasm-runtimes**

- **Wasm & Rust**
  https://www.rust-lang.org/what/wasm
  https://rustwasm.github.io/book
  https://github.com/rustwasm

Beyond the browser

- https://webassembly.org/docs/non-web/
- https://www.zdnet.com/article/microsoft-google-back-bytecode-alliance-to-move-webassembly-beyond-the-browser/

- WASI (WebAssembly System Interface)



WebAssembly System Interface (WASI) is a simple interface (ABI and API) designed by Mozilla intended to be portable to any platform.[77] It provides POSIX-like features like file I/O constrained by capability-based security.[78][79] There are also a few other proposed ABI/APIs.[80][81]

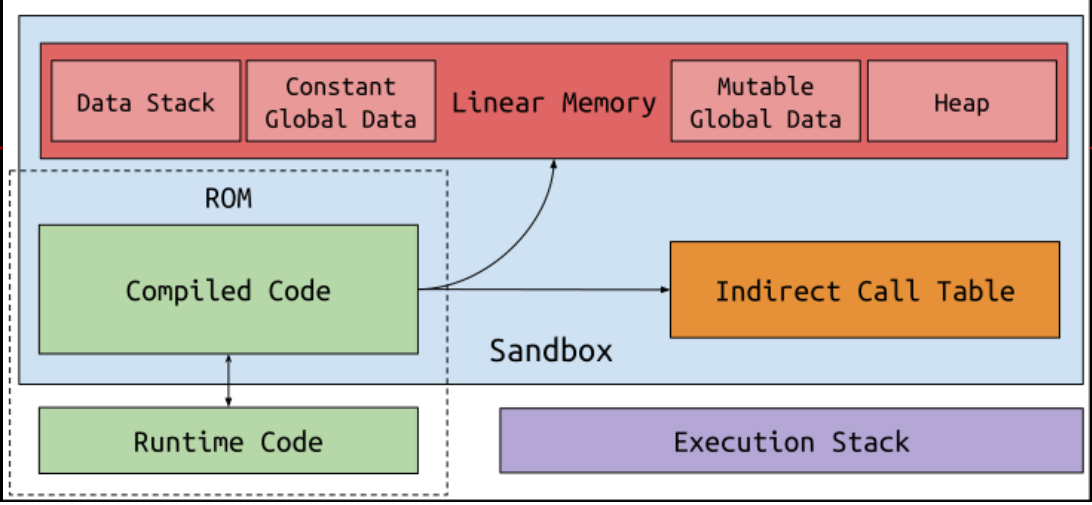
WASI is influenced by CloudABI and Capsicum.

Solomon Hykes, a co-founder of Docker, wrote in 2019, "If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. WebAssembly on the server is the future of computing."[82] Wasmer, out in version 1.0, provides "software containerization, we create universal binaries that work anywhere without modification, including operating systems like Linux, macOS, Windows, and web browsers. Wasm automatically sandboxes applications by default for secure execution".[82]

https://wasi.dev/
https://github.com/WebAssembly/WASI
https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-documents.md
https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
https://training.linuxfoundation.org/announcements/wasi-bringing-webassembly-way-beyond-browsers

Runtime

- **Sandboxing**



Wasm uses a co-design between the compiler, and the dynamic checks of the runtime system to provide the sandbox that isolates the surrounding system from the logic of the contained code. The figure depicts the main aspects of the sandbox. These include:

- *Linear memory* that holds all memory accessed by the sandbox. The compiler emits code that checks that all loads and stores remain within the linear memory, thus preventing errant accesses outside the sandbox. Linear memory is expandable much like a traditional heap.
- The *indirect function call table* that facilitates function pointer calls. To ensure that function pointer invocations are safe (to code generated by the compiler), function pointers reference an *offset* into the table. Each entry includes the type of the function, and ensures that function invocations are well-typed.
- The separation of the *execution stack* -- used to track function calls -- and the *data stack* -- used to contain stack-allocated data that can be referenced, thus must be in linear memory.

The first of these ensures the proper memory isolation of the sandbox, while the latter two provide control-flow integrity of the sandbox.

Source: https://github.com/gwsystems/aWsm/blob/master/doc/design.md

Nanoprocesses

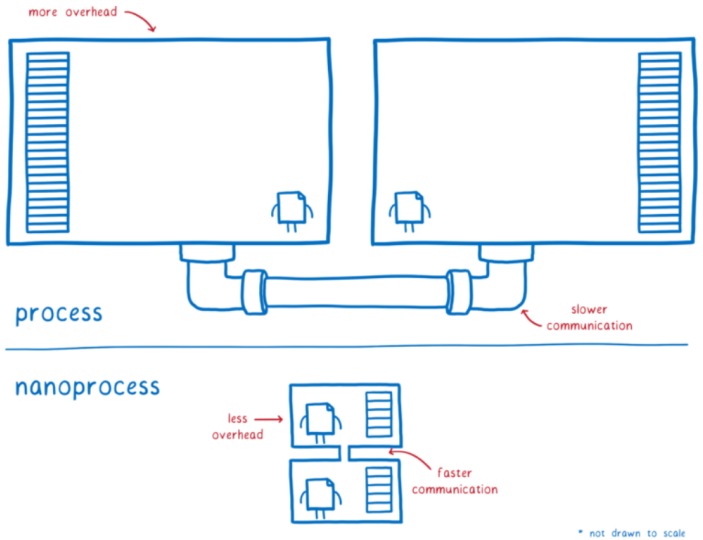
Tomorrow's solution: WebAssembly "nanoprocesses"

WebAssembly can provide the kind of isolation that makes it safe to run untrusted code. We can have an architecture that's like Unix's many small processes, or like containers and microservices.

But this isolation is much lighter weight, and the communication between them isn't much slower than a regular function call.

This means you can use them to wrap a single WebAssembly module instance, or a small collection of module instances that want to share things like memory among themselves.

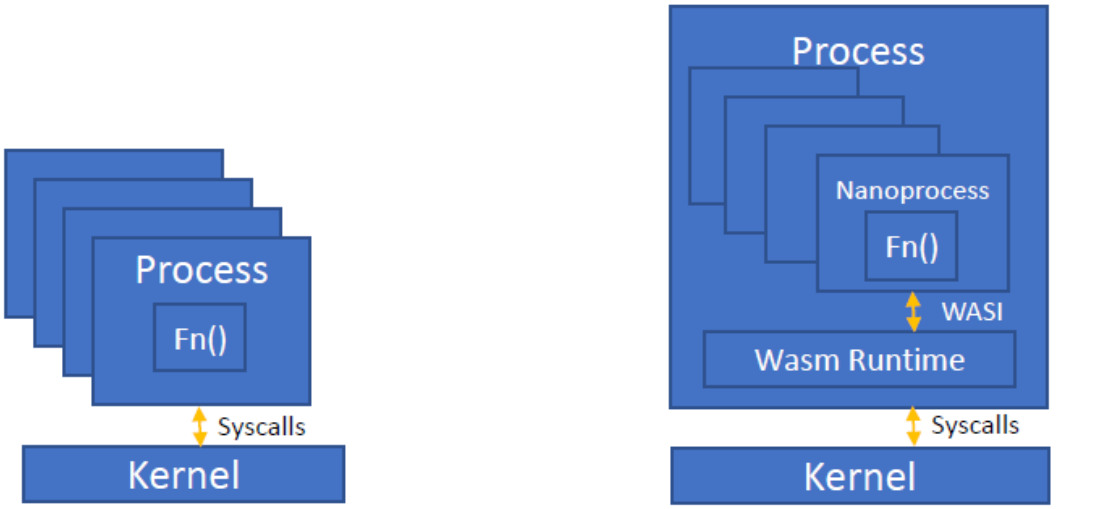
Plus, you don't have to give up the nice programming language affordances—like function signatures and static type checking.



Source: https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/

Architecture

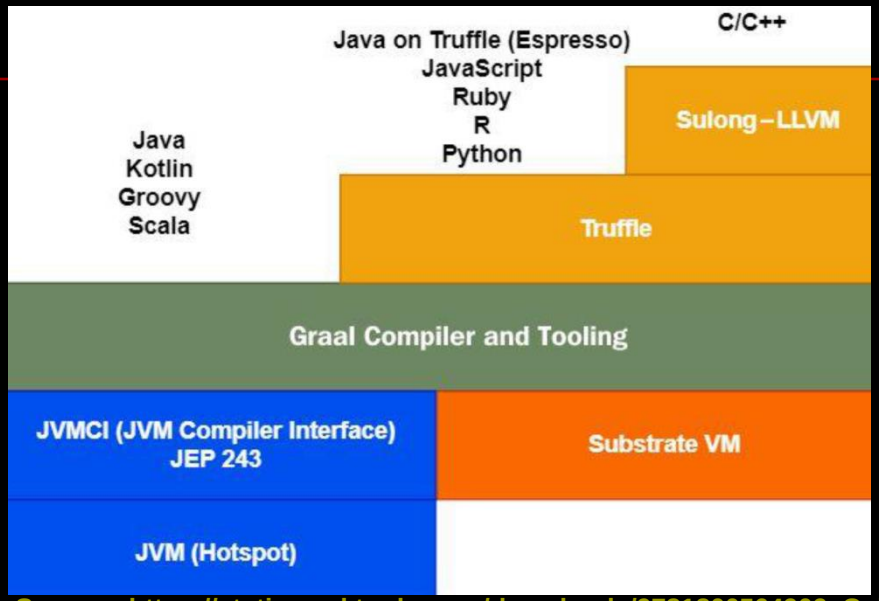
High-level reference architecture for running multiple WebAssembly sandboxes within a single native OS process. A userspace runtime schedules sandbox execution and provides system services.



Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

5.3 GraalVM

- https://en.wikipedia.org/wiki/GraalVM
- https://www.graalvm.org/
-

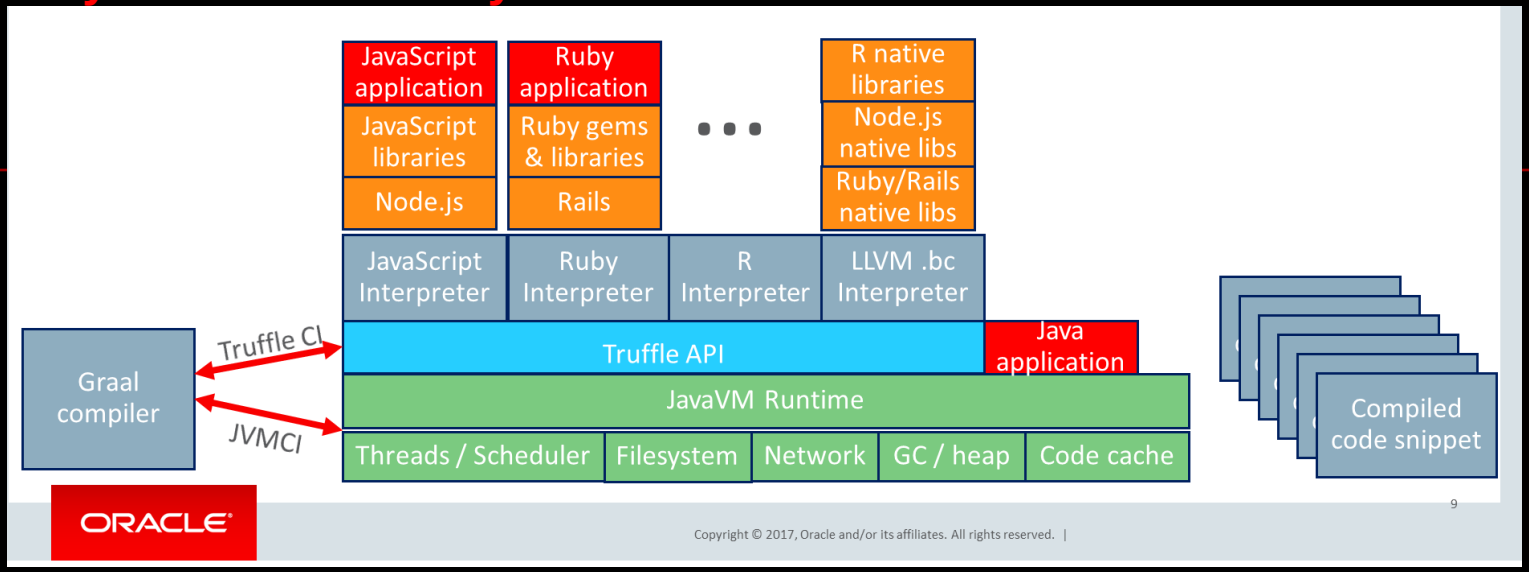


Source: https://static.packt-cdn.com/downloads/9781800564909_ColorImages.pdf

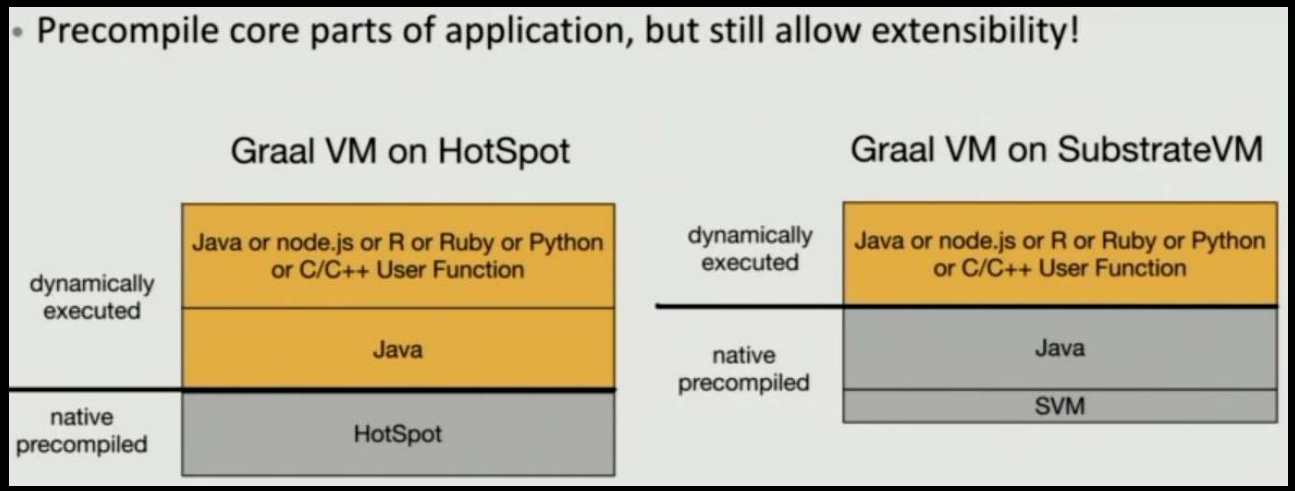
- A Universal High-Performance Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Base on OpenJDK 8, 11, 16, and 17 with JVMCI support.
- https://www.graalvm.org/docs/introduction/
- https://github.com/graalvm
- https://github.com/oracle/graal

Architecture

- A hybrid of static & dynamic runtimes

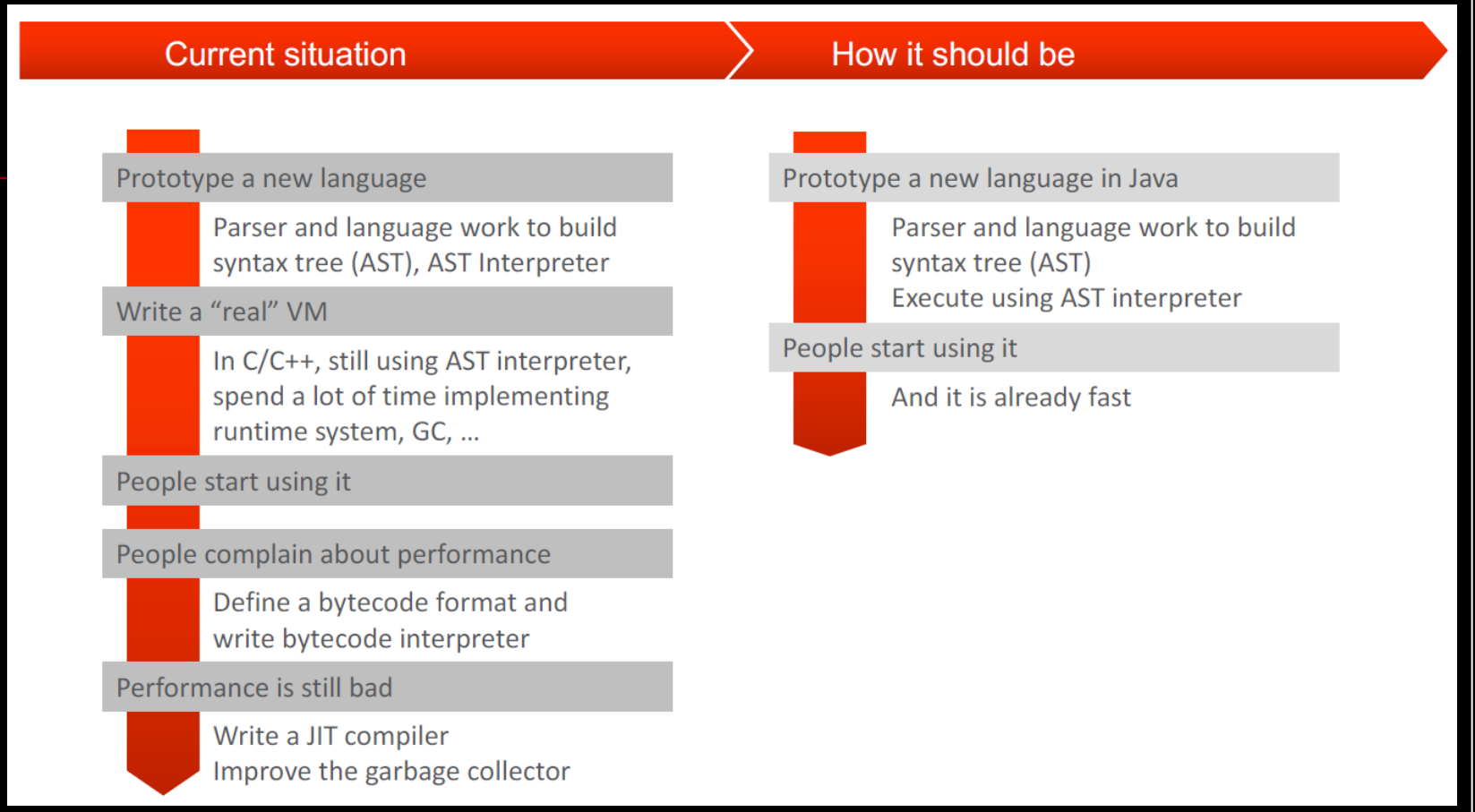


Source: https://ics.psu.edu/wp-content/uploads/2017/02/GraalVM-PSU.pptx



Source: "Adopting Java for the Serverless world", Vadym Kazulkin, JUG London 2020.

Implement a new language runtime



Source: "Turning the JVM into a Polyglot VM with Graal", Chris Seaton, Oracle Labs

Java 18

- https://openjdk.java.net/projects/jdk/18/
- Features

Features

400: UTF-8 by Default
408: Simple Web Server
413: Code Snippets in Java API Documentation
416: Reimplement Core Reflection with Method Handles
417: Vector API (Third Incubator)
418: Internet-Address Resolution SPI
419: Foreign Function & Memory API (Second Incubator)
420: Pattern Matching for switch (Second Preview)
421: Deprecate Finalization for Removal

Java 19

- https://openjdk.java.net/projects/jdk/19/
- Features

JEPs proposed to target JDK 19	*review ends*
424: Foreign Function & Memory API (Preview)	2022/05/11
427: Pattern Matching for switch (Third Preview)	2022/05/05

JEPs targeted to JDK 19, so far

422: Linux/RISC-V Port
425: Virtual Threads (Preview)
426: Vector API (Fourth Incubator)

Languages

- **https://github.com/oracle/graal/blob/master/truffle/docs/Languages.md**

Language Implementations

This page is intended to keep track of the growing number of language implementations and experiments on top of Truffle. The following language implementations exist already:

- Espresso, a meta-circular Java bytecode interpreter. *
- FastR, an implementation of GNU R. *
- Graal.js, an ECMAScript 2020 compliant JavaScript implementation. *
- Graal.Python, an early-stage implementation of Python. *
- grCUDA, a polyglot CUDA integration.
- SimpleLanguage, a toy language implementation to demonstrate Truffle features.
- SOMns, a Newspeak implementation for Concurrency Research.
- Sulong, an LLVM bitcode interpreter. *
- TRegex, a generic regular expression engine (internal, for use by other languages only). *
- TruffleRuby, an implementation of Ruby. *
- TruffleSOM, a SOM Smalltalk implementation.
- TruffleSqueak, a Squeak/Smalltalk VM implementation and polyglot programming environment.
- Yona, the reference implementation of a minimalistic, strongly and dynamically-typed, parallel and non-blocking, polyglot, strict, functional programming language.
- Enso, an open source, visual language for data science that lets you design, prototype and develop any application by connecting visual elements together.

* Shipped as part of GraalVM.

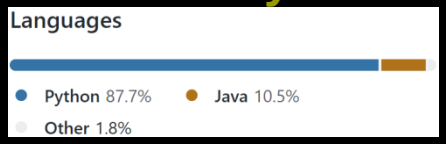
Experiments

- bf, an experimental Brainfuck programming language implementation.
- brainfuck-jvm, another Brainfuck language implementation.
- Cover, a Safe Subset of C++.
- DynSem, a DSL for declarative specification of dynamic semantics of languages.
- Heap Language, a tutorial showing the embedding of Truffle languages via interoperability.
- hextruffle, an implementation of Hex.
- LuaTruffle, an implementation of the Lua language.
- Mozart-Graal, an implementation of the Oz programming language.
- Mumbler, an experimental Lisp programming language.
- PorcE, an Orc language implementation.
- ProloGraal a Prolog language implementation supporting interoperability.
- PureScript, a small, strongly-typed programming language.
- Reactive Ruby, TruffleRuby meets Reactive Programming.
- shen-truffle, a port of the Shen programming language.
- TruffleMATE, a Smalltalk with a completely reified runtime system.
- TrufflePascal, a Pascal interpreter.
- ZipPy, a Python implementation.

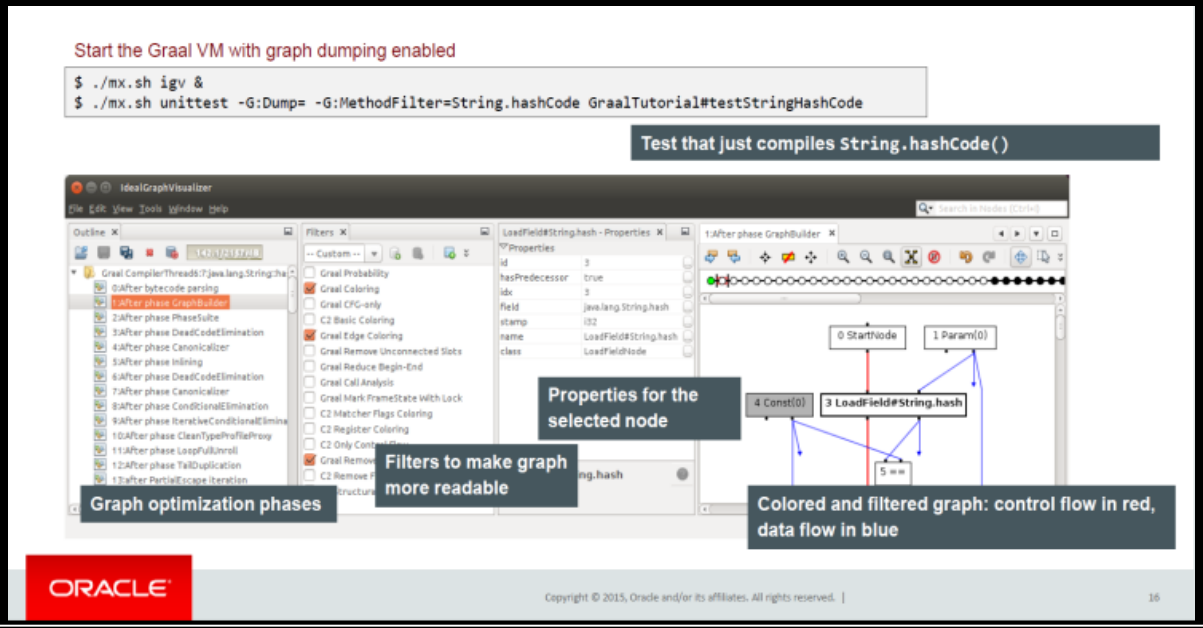
- **https://llvm.org/docs/Proposals/LLVMLibC.html**

MX

- https://github.com/graalvm/mx
- mx is a command line based tool for managing the development of (primarily) Java code. It includes a mechanism for specifying the dependencies as well as making it simple to build, test, run, update, etc the code and built artifacts
- mx is mainly written in Python and is extensible.



- mx --help
- IR Example: Ideal Graph Visualizer



5.4 .Net

- https://en.wikipedia.org/wiki/.NET
- https://dotnet.microsoft.com/
- https://docs.microsoft.com/en-us/dotnet/standard/glossary
- https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
- …
- https://en.wikipedia.org/wiki/Windows_Runtime
- https://en.wikipedia.org/wiki/.NET_Framework
- …
- https://github.com/quozd/awesome-dotnet
- https://github.com/thangchung/awesome-dotnet-core

Open Source

- https://dotnet.microsoft.com/platform/open-source
- https://github.com/dotnet
- https://github.com/aspnet
- https://github.com/microsoft
- https://en.wikipedia.org/wiki/Mono_(software)
- ...

CLI

- https://en.wikipedia.org/wiki/Common_Language_Infrastructure

The **Common Language Infrastructure** (**CLI**) is an open specification and technical standard originally developed by Microsoft and standardized by ISO (**ISO/IEC 23271**) and Ecma International (**ECMA 335**)[1][2] that describes executable code and a runtime environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures. This implies it is platform agnostic. The .NET Framework, .NET and Mono are implementations of the CLI. The metadata format is also used to specify the API definitions exposed by the Windows Runtime.[3][4]

Among other things, the CLI specification describes the following four aspects:

The Common Type System (CTS)

A set of data types and operations that are shared by all CTS-compliant programming languages.

The Metadata

Information about program structure is language-agnostic, so that it can be referenced between languages and tools, making it easy to work with code written in a language the developer is not using.

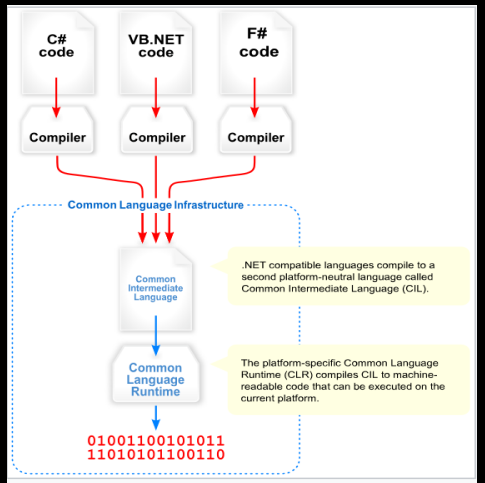
The Common Language Specification (CLS)

The CLI should conform with the set of base rules to which any language targeting, since that language should interoperate with other CLS-compliant languages. The CLS rules define a subset of the Common Type System.

The Virtual Execution System (VES)

The VES loads and executes CLI-compatible programs, using the metadata to combine separately generated pieces of code at runtime.

All compatible languages compile to Common Intermediate Language (CIL), which is an intermediate language that is abstracted from the platform hardware. When the code is executed, the platform-specific VES will compile the CIL to the machine language according to the specific hardware and operating system.



Visual overview of the Common Language Infrastructure (CLI)

- https://en.wikipedia.org/wiki/Common_Intermediate_Language
- https://en.wikipedia.org/wiki/Common_Language_Runtime

.Net 6

- https://rubikscode.net/2021/08/30/net-6-top-6-new-features-in-the-upcoming-net-6-version/

- https://devblogs.microsoft.com/dotnet/announcing-net-6/
- https://docs.microsoft.com/en-us/dotnet/core/compatibility/6.0
- …

.Net 7

- https://devblogs.microsoft.com/dotnet/announcing-net-7-preview-1/
- https://visualstudiomagazine.com/articles/2022/02/18/net-7-webassembly.aspx
- …

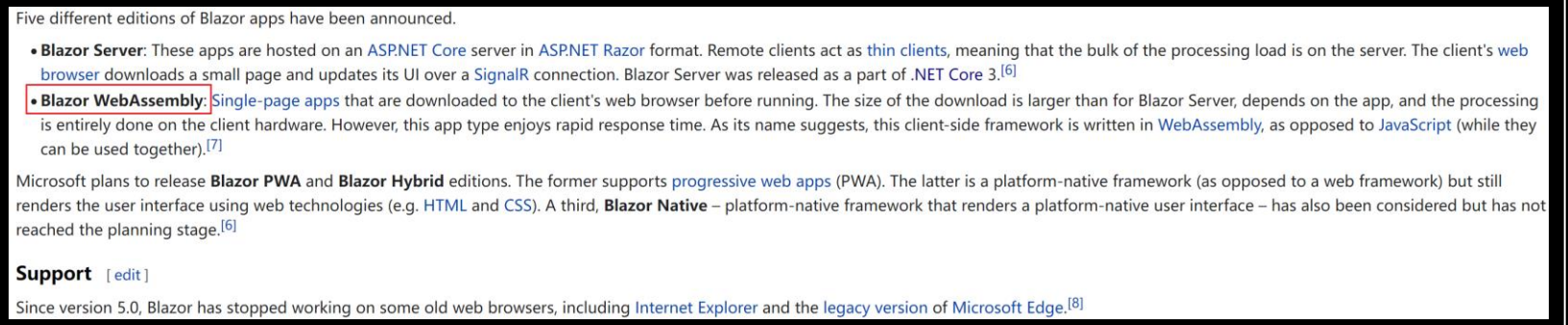
.Net on Web

- **Blazor**

  https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor
  https://github.com/dotnet/aspnetcore
  https://github.com/AdrienTorris/awesome-blazor

- https://en.wikipedia.org/wiki/Blazor

  a free and open-source web framework that enables developers to create web apps using **C# and HTML.**



Five different editions of Blazor apps have been announced.

- **Blazor Server**: These apps are hosted on an ASP.NET Core server in ASP.NET Razor format. Remote clients act as thin clients, meaning that the bulk of the processing load is on the server. The client's web browser downloads a small page and updates its UI over a SignalR connection. Blazor Server was released as a part of .NET Core 3.[6]
- **Blazor WebAssembly**: Single-page apps that are downloaded to the client's web browser before running. The size of the download is larger than for Blazor Server, depends on the app, and the processing is entirely done on the client hardware. However, this app type enjoys rapid response time. As its name suggests, this client-side framework is written in WebAssembly, as opposed to JavaScript (while they can be used together).[7]

Microsoft plans to release **Blazor PWA** and **Blazor Hybrid** editions. The former supports progressive web apps (PWA). The latter is a platform-native framework (as opposed to a web framework) but still renders the user interface using web technologies (e.g. HTML and CSS). A third, **Blazor Native** – platform-native framework that renders a platform-native user interface – has also been considered but has not reached the planning stage.[6]

Support [edit]

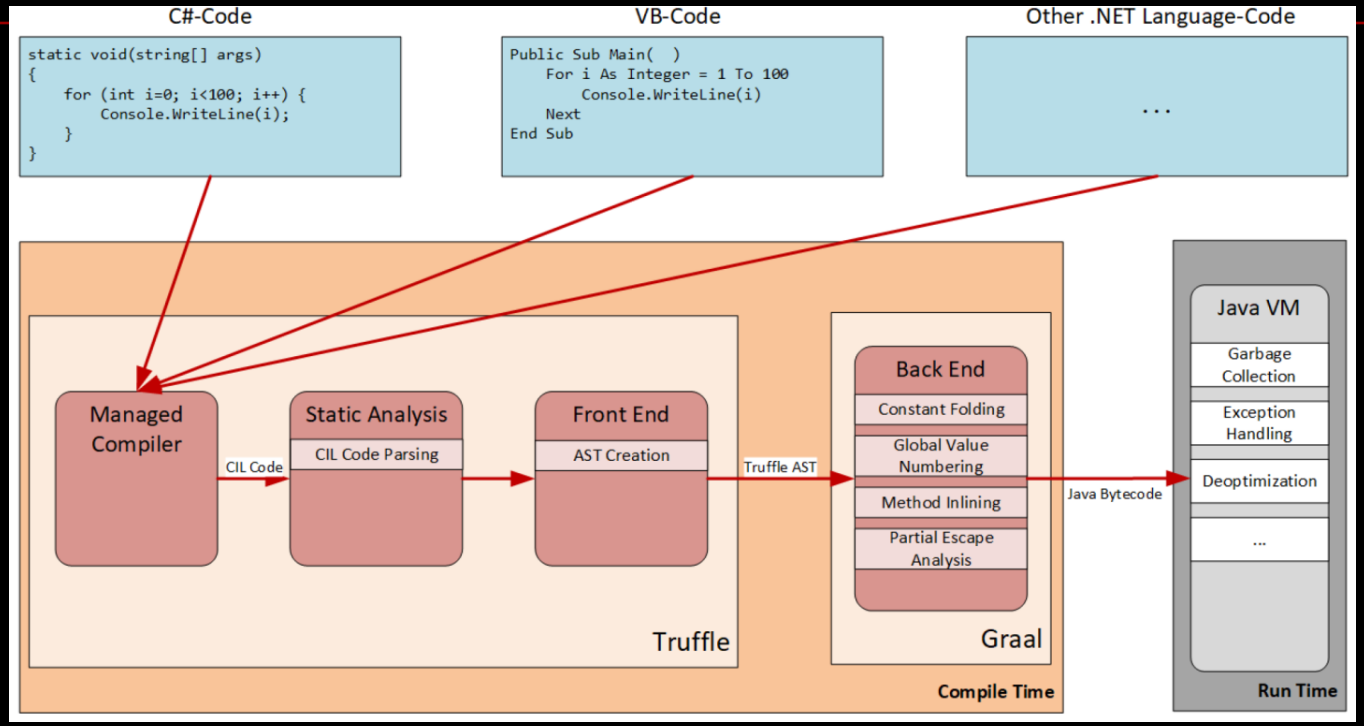
Since version 5.0, Blazor has stopped working on some old web browsers, including Internet Explorer and the legacy version of Microsoft Edge.[8]

.Net on GraalVM

- Truffle CIL Interpreter

  https://githubmemory.com/repo/jagotu/BACIL

  https://epub.jku.at/obvulihs/download/pdf/5473678



5.5 Rust

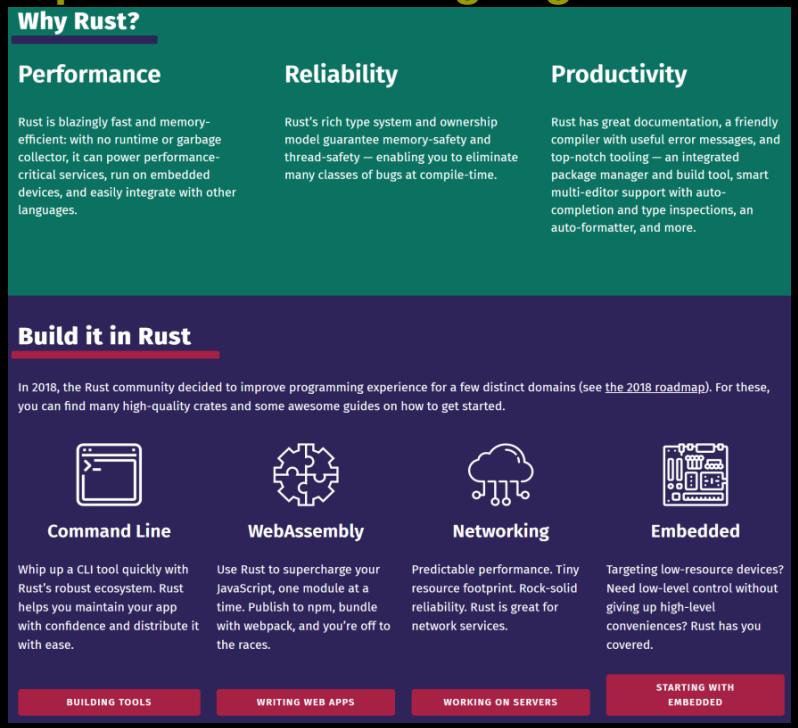
- https://en.wikipedia.org/wiki/Rust_(programming_language)

Rust is a multi-paradigm, high-level, general-purpose programming language designed for performance and safety, especially safe concurrency.[12][13] Rust is syntactically similar to C++,[14] but can guarantee memory safety by using a *borrow checker* to validate references.[15] Rust achieves memory safety without garbage collection, and reference counting is optional.[16][17]

Rust was originally designed by Graydon Hoare at Mozilla Research, with contributions from Dave Herman, Brendan Eich, and others.[18][19] The designers refined the language while writing the Servo layout or browser engine,[20] and the Rust compiler. It has gained increasing use in industry, and Microsoft has been experimenting with the language for secure and safety-critical software components.[21][22]

Rust has been voted the "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.[23]

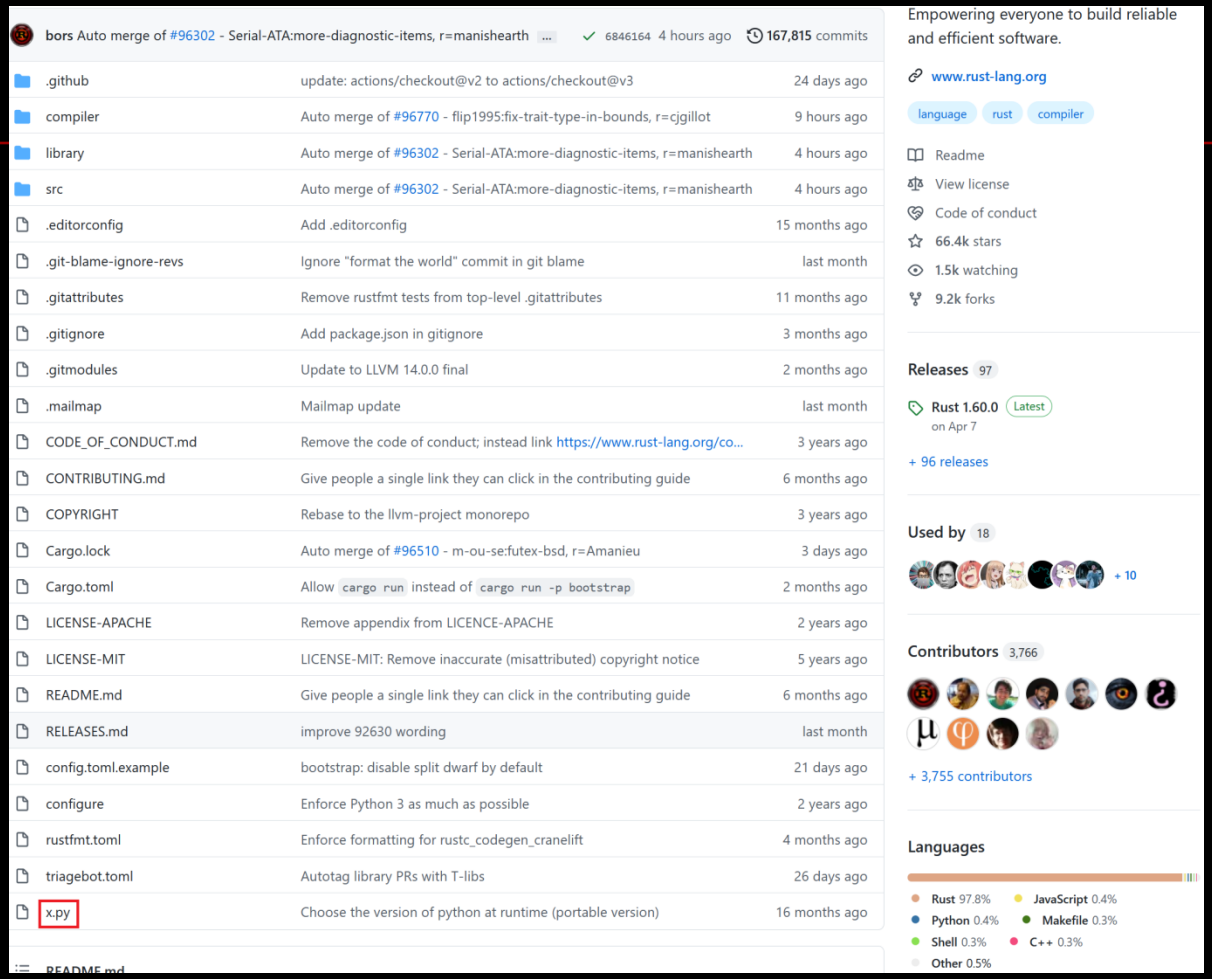
- https://www.rust-lang.org/



- https://www.memorysafety.org/
- ...

rustc & cargo

- **https://github.com/rust-lang/rust**



bors Auto merge of #96302 - Serial-ATA:more-diagnostic-items, r=manishearth ... ✓ 6846164 4 hours ago 167,815 commits

.github	update: actions/checkout@v2 to actions/checkout@v3	24 days ago
compiler	Auto merge of #96770 - flip1995:fix-trait-type-in-bounds, r=cjgillot	9 hours ago
library	Auto merge of #96302 - Serial-ATA:more-diagnostic-items, r=manishearth	4 hours ago
src	Auto merge of #96302 - Serial-ATA:more-diagnostic-items, r=manishearth	4 hours ago
.editorconfig	Add .editorconfig	15 months ago
.git-blame-ignore-revs	Ignore "format the world" commit in git blame	last month
.gitattributes	Remove rustfmt tests from top-level .gitattributes	11 months ago
.gitignore	Add package.json in gitignore	3 months ago
.gitmodules	Update to LLVM 14.0.0 final	2 months ago
.mailmap	Mailmap update	last month
CODE_OF_CONDUCT.md	Remove the code of conduct; instead link https://www.rust-lang.org/co...	3 years ago
CONTRIBUTING.md	Give people a single link they can click in the contributing guide	6 months ago
COPYRIGHT	Rebase to the llvm-project monorepo	3 years ago
Cargo.lock	Auto merge of #96510 - m-ou-se:futex-bsd, r=Amanieu	3 days ago
Cargo.toml	Allow `cargo run` instead of `cargo run -p bootstrap`	2 months ago
LICENSE-APACHE	Remove appendix from LICENCE-APACHE	2 years ago
LICENSE-MIT	LICENSE-MIT: Remove inaccurate (misattributed) copyright notice	5 years ago
README.md	Give people a single link they can click in the contributing guide	6 months ago
RELEASES.md	improve 92630 wording	last month
config.toml.example	bootstrap: disable split dwarf by default	21 days ago
configure	Enforce Python 3 as much as possible	2 years ago
rustfmt.toml	Enforce formatting for rustc_codegen_cranelift	4 months ago
triagebot.toml	Autotag library PRs with T-libs	26 days ago
x.py	Choose the version of python at runtime (portable version)	16 months ago

README.md

Empowering everyone to build reliable and efficient software.

www.rust-lang.org

language rust compiler

Readme
View license
Code of conduct
66.4k stars
1.5k watching
9.2k forks

Releases 97

Rust 1.60.0 Latest
on Apr 7

+ 96 releases

Used by 18

+ 10

Contributors 3,766

+ 3,755 contributors

Languages

Rust 97.8% JavaScript 0.4%
Python 0.4% Makefile 0.3%
Shell 0.3% C++ 0.3%
Other 0.5%

- **https://github.com/rust-lang/cargo**
- **https://crates.io/**

rustup

- **https://rustup.rs/**

  The Rust toolchain installer.

- **https://github.com/rust-lang/rustup**
- **https://rust-lang.github.io/rustup/**
- **…**

Rust for Cloud Native

- https://rust-cloud-native.github.io/

Applications and Services

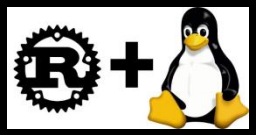
- apache/incubator-teaclave: open source universal secure computing platform, making computation on privacy-sensitive data safe and simple
- bottlerocket-os/bottlerocket: an operating system designed for hosting containers
- containers/krunvm: manage lightweight VMs created from OCI images
- containers/youki: a container runtime written in Rust
- datafuselabs/datafuse: A Modern Real-Time Data Processing & Analytics DBMS with Cloud-Native Architecture, built to make the Data Cloud easy
- firecracker-microvm/firecracker: secure and fast microVMs for serverless computing
- infinyon/fluvio: Cloud-native real-time data streaming platform with in-line computation capabilities
- krustlet/krustlet: Kubernetes Rust Kubelet
- kube-rs/controller-rs: a Kubernetes example controller
- kube-rs/version-rs: example Kubernetes reflector and web server
- kubewarden/policy-server: webhook server that evaluates WebAssembly policies to validate Kubernetes requests
- linkerd/linkerd2-proxy: a purpose-built proxy for the Linkerd service mesh
- openebs/mayastor: A cloud native declarative data plane in containers for containers
- rancher-sandbox/lockc: eBPF-based MAC security audit for container workloads
- tikv/tikv: distributed transactional key-value database
- tremor-rs/tremor-runtime: an event processing system that supporting complex workflows such as aggregation, rollups, an ETL language, and a query language
- valeriansaliou/sonic: fast, lightweight & schema-less search backend
- WasmEdge/WasmEdge: WasmEdge is a high-performance WebAssembly (Wasm) Virtual Machine (VM) runtime, which enables serverless functions to be embedded into any software platform; from cloud's edge to SaaS to automobiles

Libraries

- CNI Plugins: crate/framework to write CNI (container networking) plugins in Rust (includes a few custom plugins as well)
- containers/libkrun: a dynamic library providing Virtualization-based process isolation capabilities
- kube-rs/kube-rs: Kubernetes Rust client and async controller runtime
- qovery/engine: Qovery Engine is an open-source abstraction layer library that turns easy apps deployment on AWS, GCP, Azure, and other Cloud providers in just a few minutes
- open-telemetry/opentelemetry-rust: OpenTelemetry is a set of APIs, SDKs, tooling and integrations that are designed for the creation and management of telemetry data such as traces, metrics, and logs.

- ...

Rust heads into Linux Kernel



- **What's happening!**

https:///lwn.net/Articles/889924/ //Rustaceans at the border
https:///lwn.net/Articles/853423/ //Rust heads into the kernel?
https:///lwn.net/Articles/852704/ //Rust in the Linux kernel
https:///lwn.net/Articles/849849/ //Rust support hits linux-next
https:///lwn.net/Articles/829858/ //Supporting Linux kernel development in Rust
…
https://www.zdnet.com/article/rust-in-the-linux-kernel-why-it-matters-and-whats-happening-next/
https://blogs.gartner.com/manjunath-bhat/2021/01/03/why-2021-will-be-a-rusty-year-for-system-programmers/
https://developers.slashdot.org/story/21/04/17/009241/linus-torvalds-says-rust-closer-for-linux-kernel-development-calls-c-a-crap-language
https://www.infoq.com/news/2021/04/rust-linux-kernel-development/
https://itwire.com/open-source/rust-support-in-linux-may-be-possible-by-5-14-release-torvalds.html
https://thenewstack.io/rust-in-the-linux-kernel-good-enough/
…

- **Rust for Linux patch 7**

https://www.phoronix.com/scan.php?page=news_item&px=Rust-Linux-v7-Plus-New-Uutils

Miguel Ojeda sent out the seventh iteration of the Rust patches for the Linux kernel that add in the infrastructure for building Rust code in the kernel, adding various abstractions for use by the Rust code, and also some sample code for demonstrating how this memory-safe programming language can be used in kernel space. The Rust for Linux kernel effort continues to see much industry interest in large part for security reasons in trying to improve code safety.

With the v7 patches of Rust for Linux, various comments raised by the prior round of code review were addressed. There is also adding of SPDX license identifiers to more of the Rust code, UML x86_64 support for KUnit, documentation updates, and additional reviewed-by/acked-by tags.

https://lore.kernel.org/lkml/20220523020209.11810-1-ojeda@kernel.org/

```
182 files changed, 37614 insertions(+), 69 deletions(-)
```

Rust for Android

- https://source.android.com/setup/build/rust/building-rust-modules/overview/

The Android platform provides support for developing native OS components in Rust, a modern systems-programming language that provides memory safety guarantees with performance equivalent to C/C++. Rust uses a combination of compile-time checks that enforce object lifetime and ownership, and runtime checks that ensure valid memory accesses, thereby eliminating the need for a garbage collector.

Rust provides a range of modern language features which allow developers to be more productive and confident in their code:

- **Safe concurrent programming** - The ease with which this allows users to write efficient, thread-safe code has given rise to Rust's Fearless Concurrency slogan.
- **Expressive type system** - Rust helps prevent logical programming bugs by allowing for highly expressive types (such as Newtype wrappers, and enum variants with contents).
- **Stronger Compile-time Checks** - More bugs caught at compile-time increases developer confidence that when code compiles successfully, it works as intended.
- **Built-in Testing Framework** - Rust provides a built-in testing framework where unit tests can be placed alongside the implementation they test, making unit testing easier to include.
- **Error handling enforcement** - Functions with recoverable failures can return a Result type, which will be either a success variant or an error variant. The compiler requires callers to check for and handle the error variant of a `Result` enum returned from a function call. This reduces the potential for bugs resulting from unhandled failures.
- **Initialization** - Rust requires every variable to be initialized to a legal member of its type before use, preventing an unintentional initialization to an unsafe value.
- **Safer integer handling** - All integer-type conversions are explicit casts. Developers can't accidentally cast during a function call when assigning to a variable, or when attempting to do arithmetic with other types. Overflow checking is on by default in Android for Rust, which requires overflow operations to be explicit.

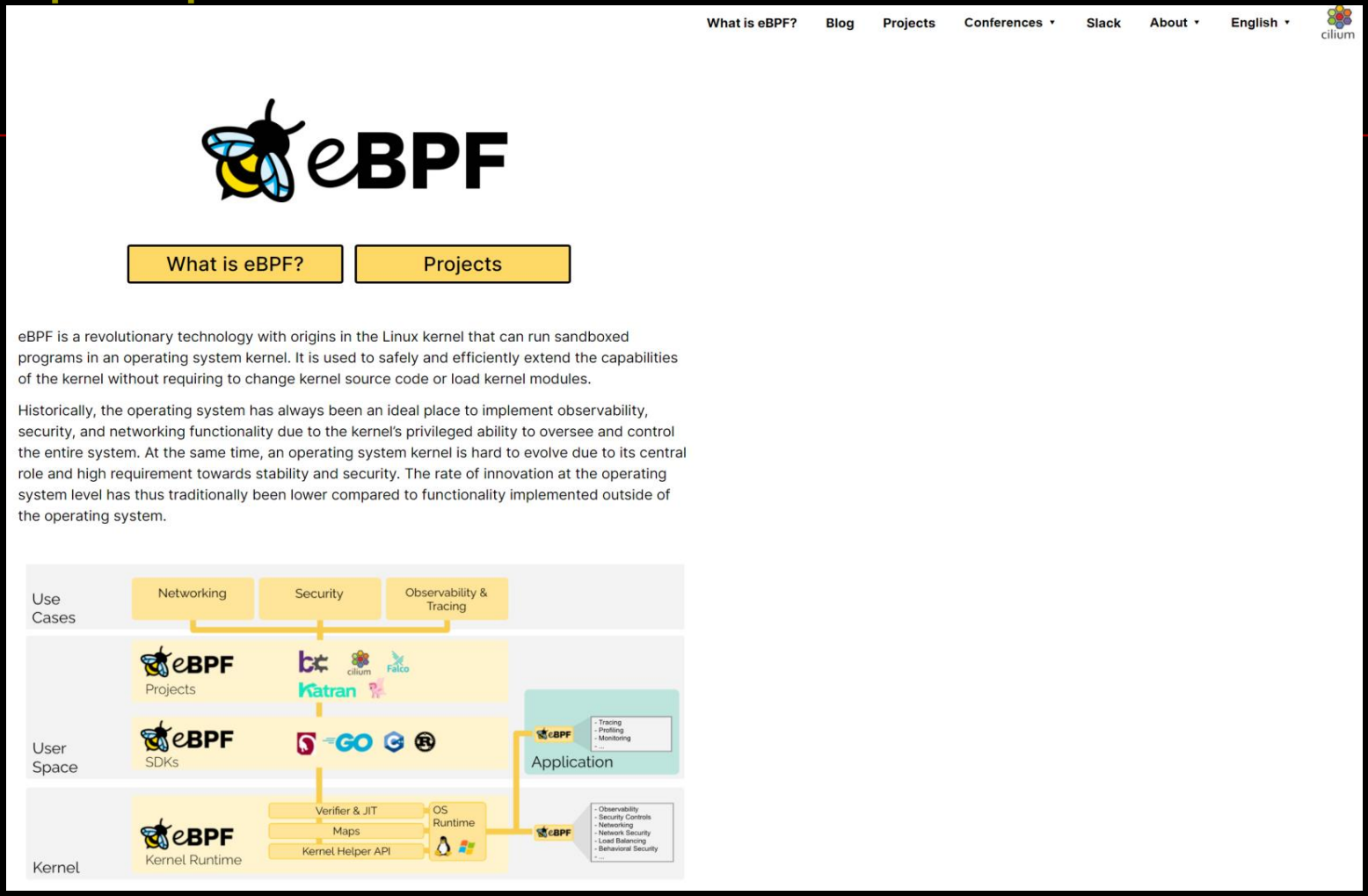
For more information, see the series of blog posts on Android Rust support:

- Rust in the Android Platform
  Provides an overview on why the Android team introduced Rust as a new platform language.
- Integrating Rust into the Android Open Source Project
  Discusses how Rust support has been introduced to the build system, and why certain design decisions were made.
- Rust/C++ interop in the Android Platform
  Discusses the approach to Rust/C++ interoperability within Android.

- ...

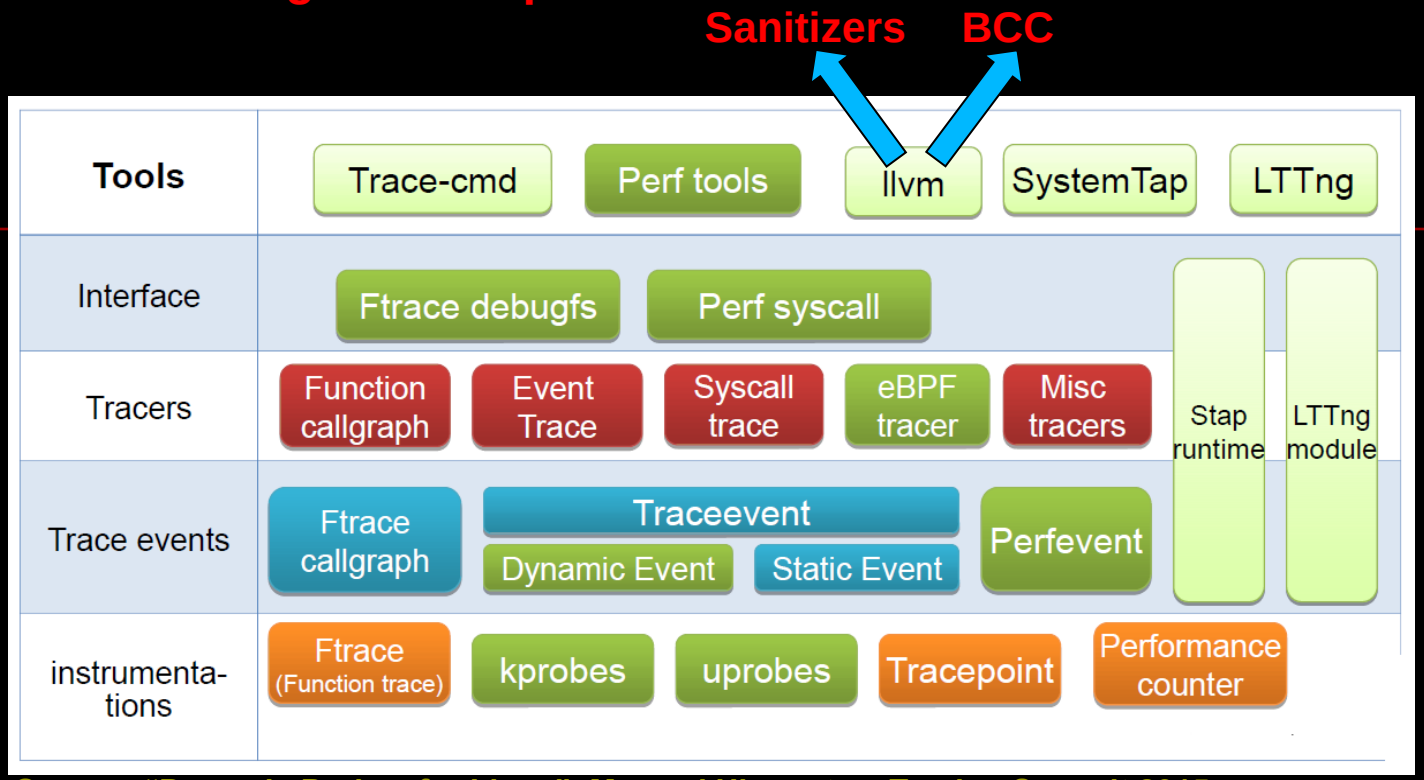
5.6 eBPF

- https://ebpf.io/



- $KERNEL_SRC/Documentation/networking/filter.txt
- $KERNEL_SRC/Documentation/trace/features/*.*

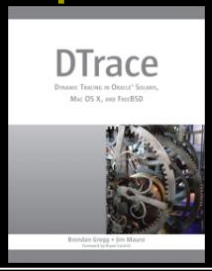
Linux Tracing Landscape

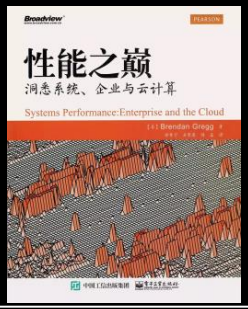


Source: "Dynamic Probes for Linux", Masami Hiramatsu, Tracing Summit 2015

DTrace

http://www.brendangregg.com/dtrace.html

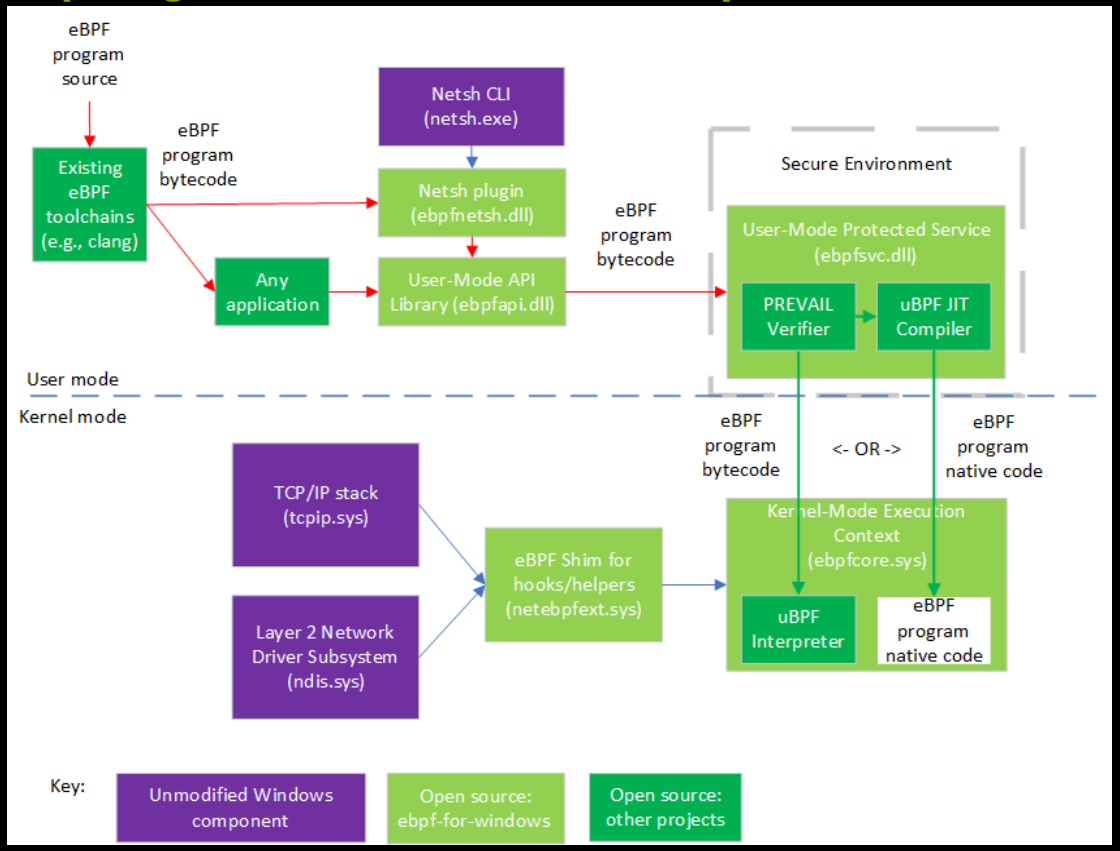


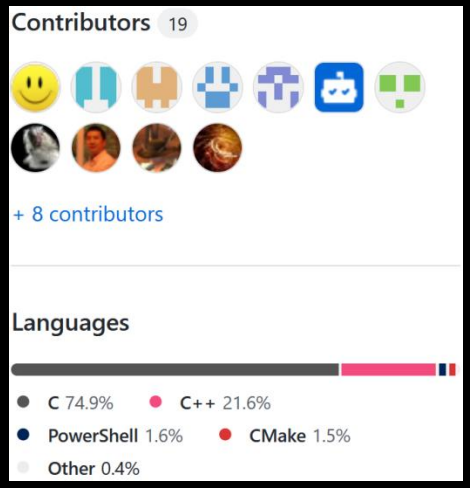


5.6.1 Cross-Platform

eBPF on Windows

- https://cloudblogs.microsoft.com/opensource/2021/05/10/making-ebpf-work-on-windows/
- https://github.com/microsoft/ebpf-for-windows





- https://lwn.net/Articles/857215/ //Implementing eBPF for Windows
- https://thenewstack.io/microsoft-brings-ebpf-to-windows/

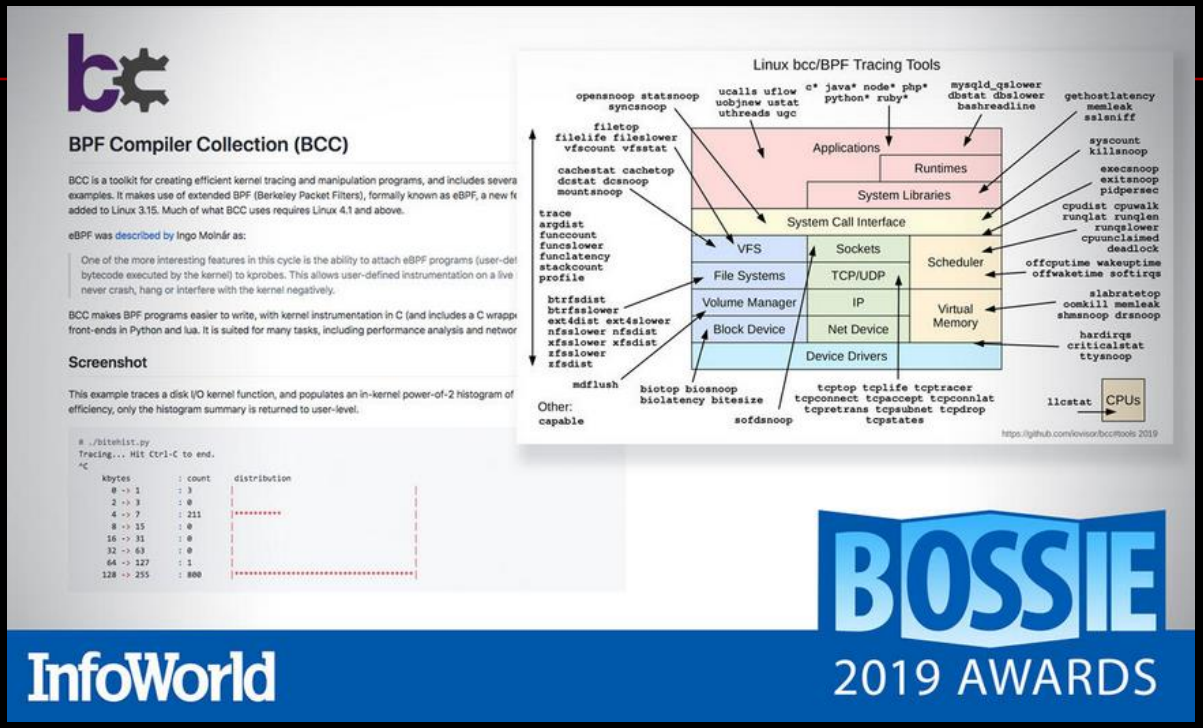
Status

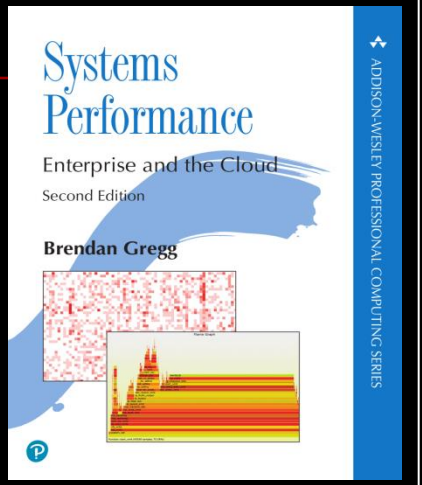
Platform →	Linux		MacOSX		Android		FreeBSD		Windows		TockOS (embedded)	
↓ Project	Kernel	User	Kernel	User	Kernel	User	Kernel	User	Kernel	User	Kernel	User
Linux	2014											
uBPF		2015										
rbpf		2017		2017						2018		
Android					2017							
Generic eBPF	2017	2017		2017			2017	2017				
eBPF for Windows									2021			
Tock											2021	

Source: "The Cross-Platform Future of eBPF", Dave Thaler(Microsoft), Cloud Native eBPF Day North America 2021

5.6.2 BCC (BPF Compiler Collection)

- https://www.infoworld.com/article/3444198/the-best-open-source-software-of-2019.html

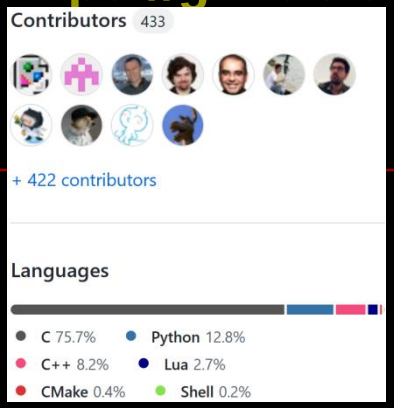




- https://www.brendangregg.com/systems-performance-2nd-edition-book.html

What is it

- **https://github.com/iovisor/bcc/**



A toolkit with Python/Lua frontend for compiling, loading, and executing BPF programs, which allows user-defined instrumentation on a live kernel image:

- **compile BPF program from C source(A modified C language for BPF backends)**
- **attach BPF program to kprobe/uprobe/tracepoint/USDT/socket…**
- **poll data from BPF program**
- **framework for building new tools or one-off scripts**
- **additional projects to support Go, Rust, and DTrace-style frontend**
- **https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md**
- **…**

Python frontend

- https://github.com/iovisor/bcc/blob/master/examples/tracing/nodejs_http_server.py

```
#!/usr/bin/python
#
# nodejs_http_server    Basic example of node.js USDT tracing.
#                       For Linux, uses BCC, BPF. Embedded C.
#
# USAGE: nodejs_http_server PID
#
# Copyright 2016 Netflix, Inc.
# Licensed under the Apache License, Version 2.0 (the "License")

from __future__ import print_function
from bcc import BPF, USDT
from bcc.utils import printb
import sys

if len(sys.argv) < 2:
    print("USAGE: nodejs_http_server PID")
    exit()
pid = sys.argv[1]
debug = 0

# load BPF program
bpf_text = """
#include <uapi/linux/ptrace.h>
int do_trace(struct pt_regs *ctx) {
    uint64_t addr;
    char path[128]={0};
    bpf_usdt_readarg(6, ctx, &addr);
    bpf_probe_read_user(&path, sizeof(path), (void *)addr);
    bpf_trace_printk("path:%s\\n", path);
    return 0;
};
"""
```

```

# enable USDT probe from given PID
u = USDT(pid=int(pid))
u.enable_probe(probe="http__server__request", fn_name="do_trace")
if debug:
    print(u.get_text())
    print(bpf_text)

# initialize BPF
b = BPF(text=bpf_text, usdt_contexts=[u])

# header
print("%-18s %-16s %-6s %s" % ("TIME(s)", "COMM", "PID", "ARGS"))

# format output
while 1:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except ValueError:
        print("value error")
        continue
    except KeyboardInterrupt:
        exit()
    printb(b"%-18.9f %-16s %-6d %s" % (ts, task, pid, msg))
```

...

- https://github.com/iovisor/bcc/blob/master/docs/tutorial_bcc_python_developer.md

5.6.3 Python to eBPF

5.6.3.1 py2bpf

- https://github.com/facebookresearch/py2bpf
  Translates functions from Python to BPF
- Languages

Example

- https://github.com/facebookresearch/py2bpf/tree/master/examples

```python
#!/usr/bin/env python3

# Copyright (c) 2017-present, Facebook, Inc.
# All rights reserved.
#
# This source code is licensed under the BSD-style license found in the
# LICENSE file in the root directory of this source tree.

import sys
import ctypes

import py2bpf.util
import py2bpf.funcs as funcs
import py2bpf.kprobe

class Call(ctypes.Structure):
    _fields_ = [
        ('pid', ctypes.c_int),
        ('comm', ctypes.c_char * 32),
        ('path', ctypes.c_char * 256),
    ]


def main(argv):
    py2bpf.util.ensure_resources()
```

```python
    call_queue = py2bpf.datastructures.BpfQueue(Call)

    @py2bpf.kprobe.probe('sys_open')
    def open_probe(pt_regs):
        call = Call()
        call.pid = funcs.get_current_pid_tgid() & 0xffffffff
        funcs.get_current_comm(call.comm)
        funcs.probe_read_str(call.path, pt_regs.rdi)

        cpuid = funcs.get_smp_processor_id()
        funcs.perf_event_output(pt_regs, call_queue, cpuid, call)

        return 0


    with open_probe():
        for call in call_queue:
            print('pid={} comm={} path={}'.format(
                call.pid,
                call.comm.decode(),
                call.path.decode()))


if __name__ == '__main__':
    main(sys.argv)
```

How it works

See: `_translation/_translate.py`

1. Un-stacking the stack-base virtual machine

First, we have to convert python's stack-based bytecode into one that is dealing with discrete variables. This means turning something like this

```
push(5)
push(10)
multiply()
```

into something that looks more like this

```
x = 5
y = 10
z = multiply(x, y)
```

Specifically, we're associating all of the instructions with their inputs and outputs. Typically, an instruction will have anywhere from zero to multiple inputs and a single output, but some instructions will have multiple outputs -- `ROT_THREE` transforms swaps the top 3 elements of the stack around, so it'll have 3 inputs and 3 outputs.

This gets a little complicated with control flow -- an if/else statement gives two potential flows. We normalize this by iterating over every possible control path. This works because bpf disallows loops, recursion, and other interesting approaches, so we know iterating over every path won't be too expensive.

See: `_translation/_vars.py`

2. Partial evaluation

Obviously most python functionality is not available within bpf, so we need to evaluate as much of the program as we can before handing it off to be switched. To do, we convert all global loads and dereferences into constants and then evaluate things as far down the line as we can. As a result, the following would be permitted:

```
    packet_short = py2bpf.funcs.load_skb_short(24)
    if packet_short == socket.htons(12345):
        return 0
```

Because `socket.htons(12345)` can be eagerly evaluated to `0x3930`. The following *will not* work, because we'd need to evaluate it in the bpf context where `socket.htons` is not available.

```
    packet_short = py2bpf.funcs.load_skb_short(24)
    # DOES NOT WORK!!!
    if socket.ntohs(packet_short) == 12345:
        return 0
```

See: `_translation/_folding.py`

3. Allocate real space for the variables

In the bpf world, we don't have a magical runtime to store our variables for us, so we need to allocate space on the stack for each. We can and should take a sophisticated approach wherein we monitor lifetimes and reuse stack space, but currently we just use a bump allocator -- we decrementing into the stack offset every time we see a new pointer. This will run us out of space in theory but hasn't been a problem in practice.

See: `_translation/_stack.py`

4. Compiling to bpf

At this point, we've got something that looks a lot more like bpf and a lot less like python's stack machine. Now we can translate it to bpf using simple templates. For example, if we see something like `return 7`, we'll translate it into a `mov` instruction which puts the immediate value `7` into the return register `R0`, and then we'll add the `ret` instruction.

See: `_bpf/_template_jit.py`

Datastructures

py2bpf supports native bpf datastructures like map. These datastructures can be transparently shared between functions compiled to bpf and the userspace processes associated with them. For example, you could implement a socket filter that simply counts packets by protocol using the following.

```
m = py2bpf.datastructures.create_map(ctypes.c_uint32, ctypes.c_uint64, 16)
def filter_fn(skb):
    m[skb.protocol] += 1
    return 0
```

In the above example, you could reference the result from other python functions with something like.

```
for proto, count in m.items():
    print('{} => {}'.format(proto, count))
```

You can also use bpf perf queues.

```
q = py2bpf.datastructures.BpfQueue(ctypes.c_int)

@py2bpf.kprobe.probe('sys_close')
def on_sys_close(pt_regs):
    pid = py2bpf.funcs.get_current_pid_tgid() & 0xffffffff
    ptr = py2bpf.funcs.addrof(pid)
    cpuid = py2bpf.funcs.get_smp_processor_id()
    py2bpf.funcs.perf_event_output(pt_regs, q, cpuid, ptr)
    return 0


with on_sys_close():
    for pid in q:
        print('pid={}'.format(pid))
```

Helpers

Limitations in the bpf bytecode mean that a lot of functionality is provided via helper functions which can be called from bpf programs. These helpers appear to you as simple python functions which can be invoked in compiled functions. For example, the following will issue a printk to `/sys/kernel/debug/tracing/trace_pipe`.

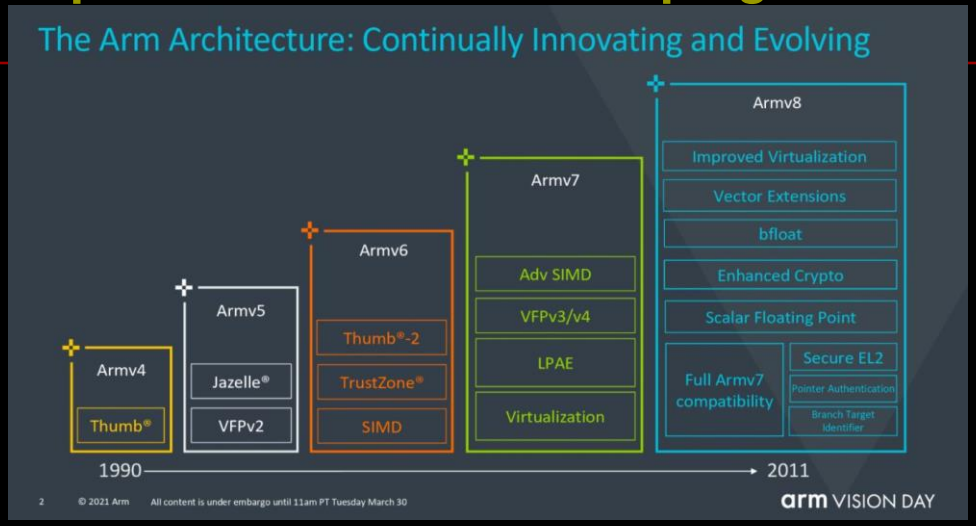
```
def fn(ctx):
    ...
    py2bpf.funcs.trace_printk("Hello World!")
    ...
```

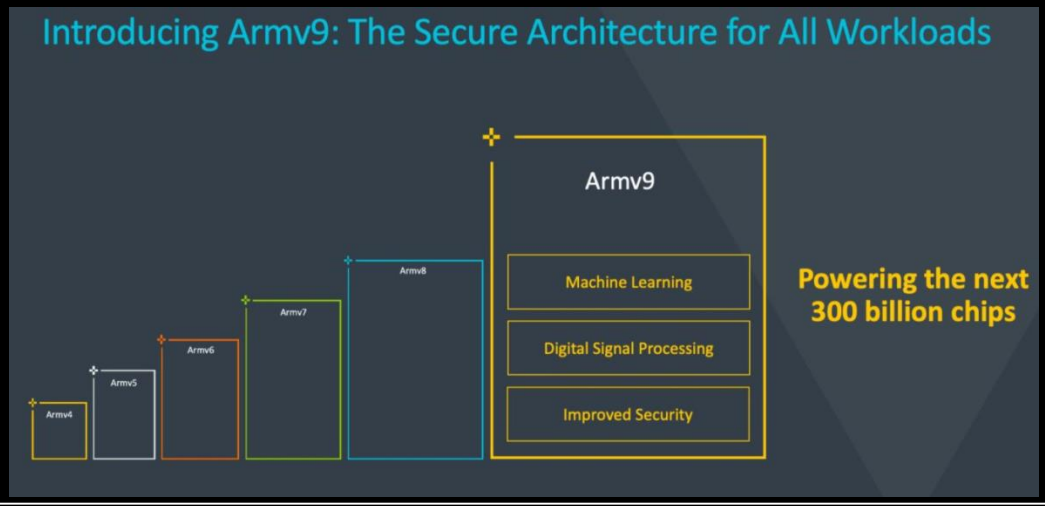
These functions are generally described in `/usr/include/linux/bpf.h`. The major difference in signature is that it's annoying in python to have to specify the array or string and then the length, so we provide the length transparently instead (see `fill_array_size_args` in `py2bpf/funcs.py:Func`).

5.7 ARM Ecosystem in 2021

5.7.1 ARM v9

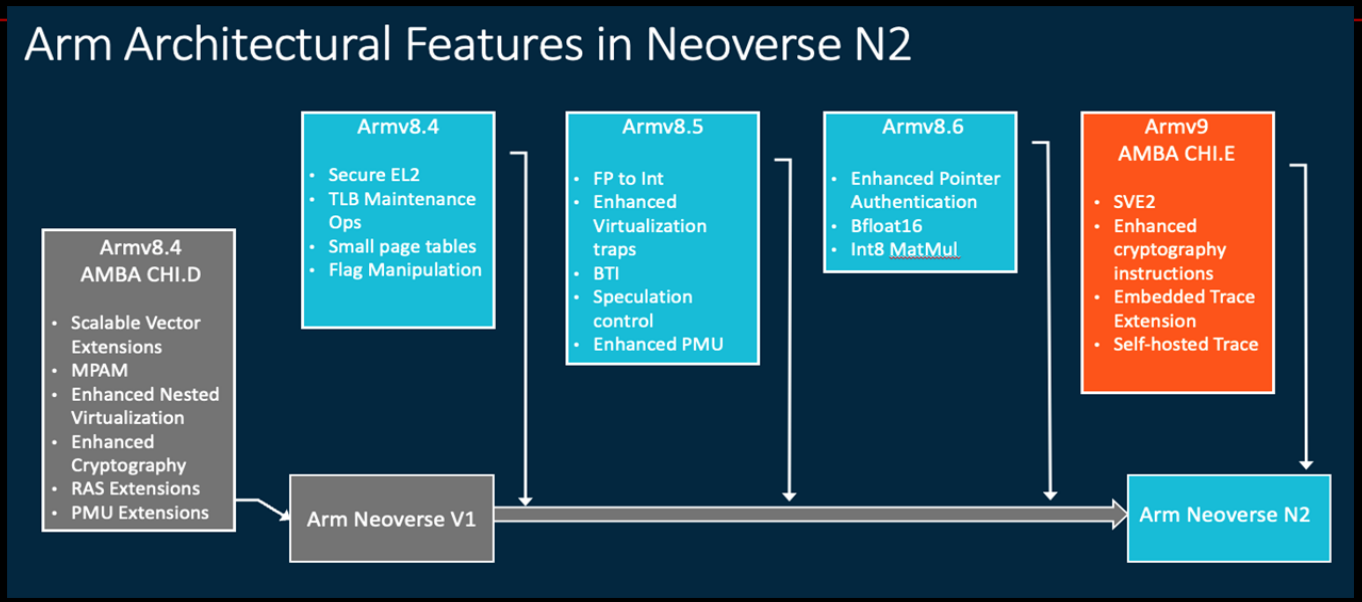
- https://www.arm.com/campaigns/arm-vision



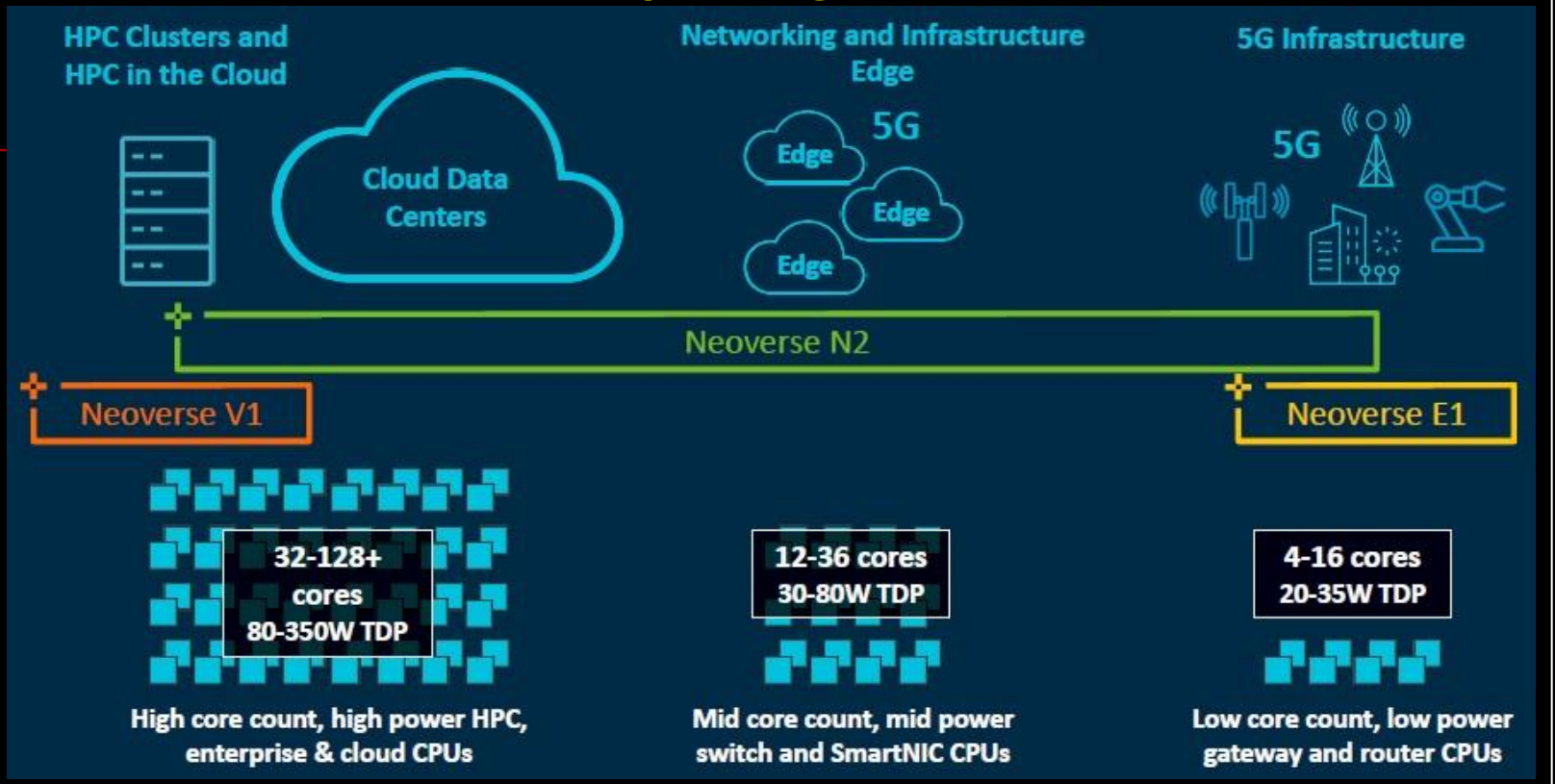


5.7.2 ARM for Edge Computing

- https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-neoverse-n2-industry-leading-performance-efficiency



- https://www.nextplatform.com/2021/04/27/arm-puts-some-muscle-into-future-neoverse-server-cpu-designs/

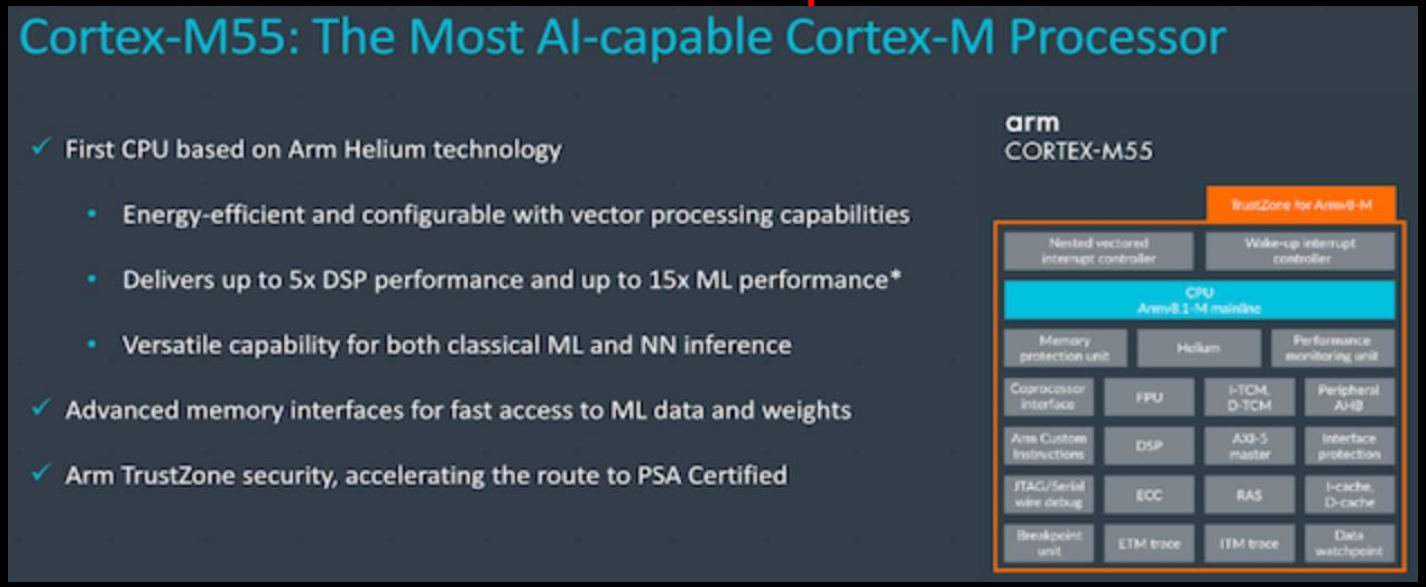


5.7.3 Cortex-M

- https://www.arm.com/blogs/blueprint/tinyml
- https://www.arm.com/why-arm/technologies/helium
- https://www.arm.com/why-arm/technologies/custom-instructions
- https://www.zhihu.com/question/371097288/answer/1010694311
- https://www.arm.com/blogs/blueprint/cambridge-consultants
- **Cortex-M55**

  https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55

First Helium and Custom Instruction capable CPU core



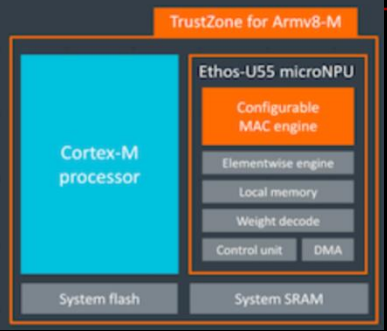
Ethos: MicroNPU for ARM

https://www.arm.com/product-filter?families=ethos%20npus&showall=true

Ethos-U55:

Ethos-U55: The First microNPU for Cortex-M

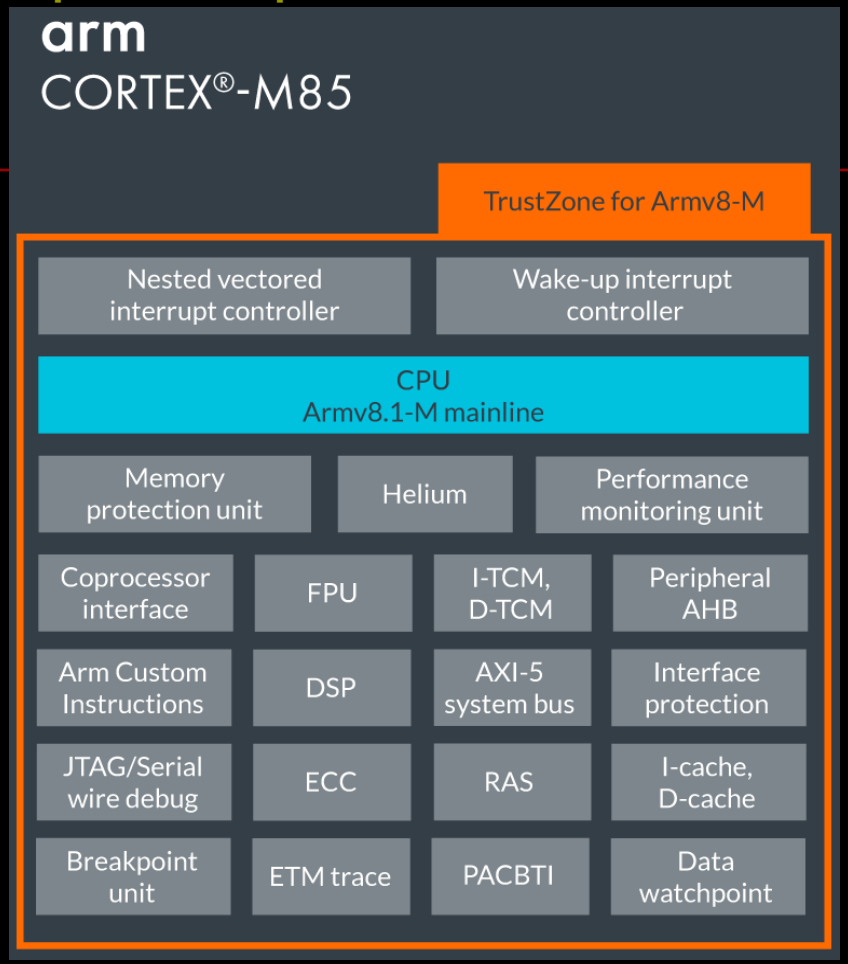
- ✓ Highest efficiency and small memory footprint
- ✓ 32, 64, 128, or 256 unit multiply-accumulate (MAC) engine
- ✓ Weight decoder and DMA for on-the-fly weight decompression
- ✓ Tooling available for offline optimization
- ✓ Works with a range of Cortex-M processors:
  - Cortex-M55
  - Cortex-M7
  - Cortex-M33
  - Cortex-M4



Key Features	Performance (At 1GHz)	64 to 512 GOP/s
	MACs (8x8)	32, 64, 128, 256
	Utilization on popular networks	Up to 85%
	Data Types	Int-8 and Int-16
	Network Support	CNN and RNN/LSTM
	Winograd Support	No
	Sparsity	Yes
Memory System	Internal SRAM	18 to 50 KB
	External On Chip SRAM	KB to Multi-MB
	Compression	Weights only
	Memory Optimizations	Extended compression, layer/operator fusion
Development Platform	Neural Frameworks	TensorFlow Lite Micro
	Operating Systems	RTOS or bare-metal
	Software Components	TensorFlow Lite Micro Runtime, CMSIS-NN, Optimizer, Driver
	Debug and Profile	Layer-by-layer visibility with PMUs
	Evaluation and Early Prototyping	Performance Model, Cycle Accurate Model, or FPGA Evaluations

- **Cortex-M85**

https://developer.arm.com/Processors/Cortex-M85



Highest Scalar, DSP, and ML Performance

Cortex-M85 integrates Arm Helium technology to deliver the highest scalar, DSP, and ML performance of the Cortex-M series of processors.

Faster Route to Security Certification

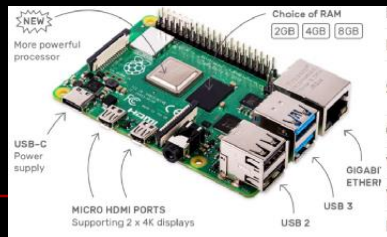
PACBTI offers enhanced software attack threat mitigation, easing the journey to achieve PSA Certified Level 2, which is becoming the baseline for IoT deployments.

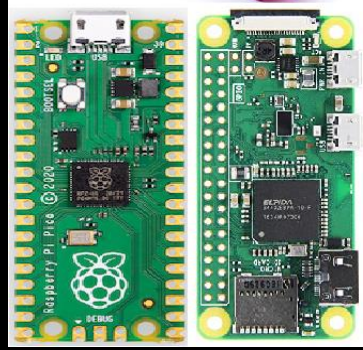
https://developer.arm.com/Processors/Corstone-310

https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m85

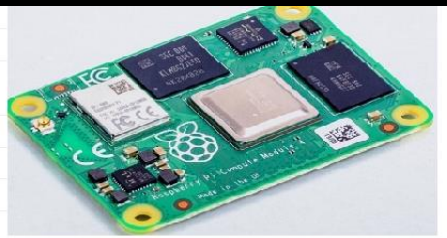
...

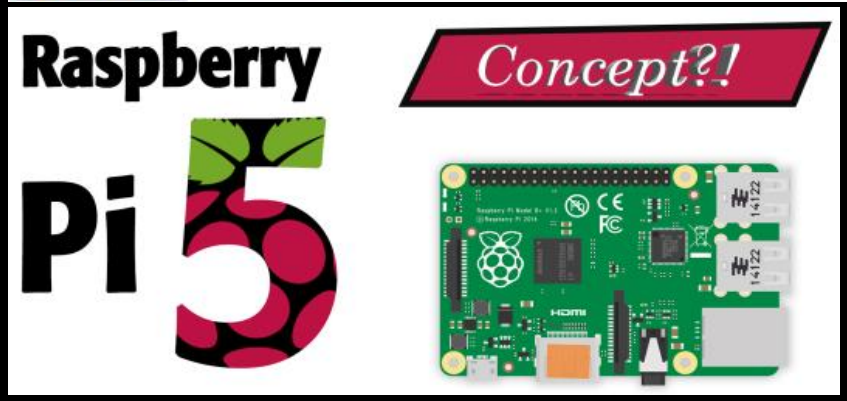
5.7.4 Raspberry Pi





Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)





5.7.4.1 Fedora

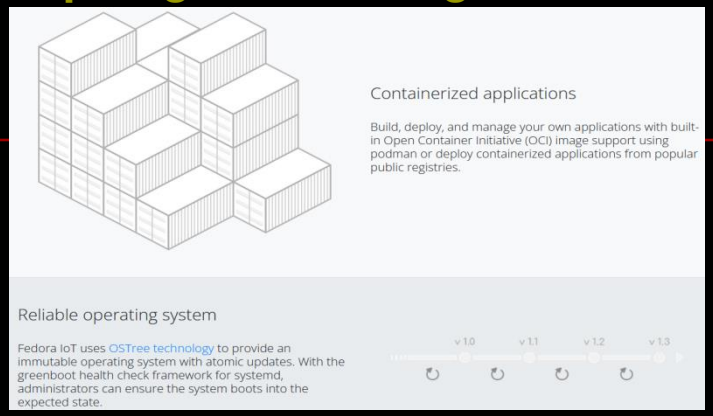
- A Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat.
- https://en.wikipedia.org/wiki/Fedora_(operating_system)
- https://getfedora.org/
- https://alt.fedoraproject.org/alt/
- https://spins.fedoraproject.org/
- https://fedoraproject.org/wiki/Architectures/ARM
- https://fedoramagazine.org/
- https://silverblue.fedoraproject.org/
- …
- Developer friendly!

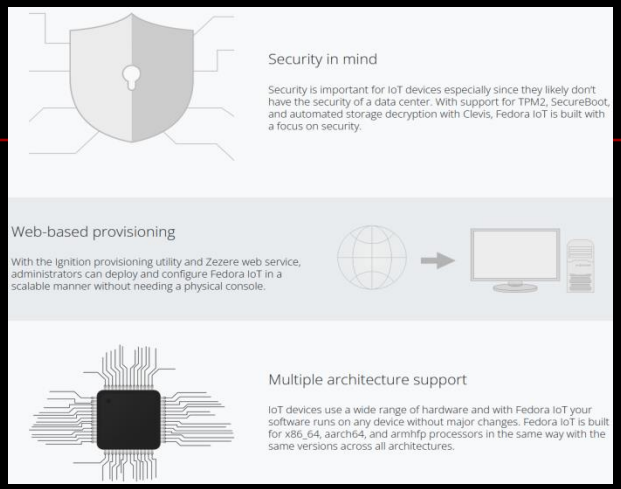
Fedora 36

- https://fedoraproject.org/wiki/Releases/36/ChangeSet
- https://www.phoronix.com/scan.php?page=news_item&px=Fedora-36

Fedora IoT

- **https://getfedora.org/iot/**





- **https://docs.fedoraproject.org/en-US/iot/**
- **FIDO**

https://www.phoronix.com/scan.php?page=news_item&px=Fedora-IoT-Better-Onboarding

For Fedora 37 later this year the Linux distribution is looking at providing support for zero touch onboarding for IoT / edge devices.

Fedora IoT 37 is looking at making use of the FIDO Alliance's FIDO Device Onboard (FDO) protocol as an open standard for simply and securely onboarding devices to cloud and on-premise management platforms. The FIDO Alliance announced the FDO Protocol last year for easily onboarding IoT devices for simplifying device setup, allowing more flexible setups, and doing so securely.

This "zero touch onboarding" just relies on a device credential and a root and chain of trust to ensure onboarding of devices without stored credentials. There is a Rust language based implementation of the FIDO device onboarding stack that Fedora is looking to use and enble by default for the Fedora IoT Edition.

https://fedoraproject.org/wiki/Changes/FIDODeviceOnboarding

https://github.com/fedora-iot/fido-device-onboard-rs

Dev Env

```
[mydev@fedora /]$ uname -a
Linux fedora 5.17.8-300.fc36.aarch64 #1 SMP Mon May 16 00:40:35 UTC 2022 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora /]$
[mydev@fedora /]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/aarch64-redhat-linux/12/lto-wrapper
Target: aarch64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,objc,obj-c++,ada,go,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --enable-libstdcxx-backtrace --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --with-isl=/builddir/build/BUILD/gcc-12.1.1-20220507/obj-aarch64-redhat-linux/isl-install --enable-gnu-indirect-function --build=aarch64-redhat-linux --with-build-config=bootstrap-lto --enable-link-serialization=1
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 12.1.1 20220507 (Red Hat 12.1.1-1) (GCC)
[mydev@fedora /]$
[mydev@fedora /]$ clang -v
clang version 14.0.0 (Fedora 14.0.0-1.fc36)
Target: aarch64-redhat-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
Found candidate GCC installation: /usr/bin/../lib/gcc/aarch64-redhat-linux/12
Selected GCC installation: /usr/bin/../lib/gcc/aarch64-redhat-linux/12
Candidate multilib: .;@m64
Selected multilib: .;@m64
```

```
[mydev@fedora /]$ java --version
openjdk 17.0.3 2022-04-19
OpenJDK Runtime Environment GraalVM CE 22.1.0 (build 17.0.3+7-jvmci-22.1-b06)
OpenJDK 64-Bit Server VM GraalVM CE 22.1.0 (build 17.0.3+7-jvmci-22.1-b06, mixed mode, sharing)
[mydev@fedora /]$ gu list
ComponentId              Version             Component name                Stability                     Origin
------------------------------------------------------------------------------------------------------------
---
graalvm                  22.1.0              GraalVM Core                  Supported
espresso                 22.1.0              Java on Truffle               Experimental
js                       22.1.0              Graal.js                      Supported
llvm-toolchain           22.1.0              LLVM.org toolchain            Supported
native-image             22.1.0              Native Image                  Early adopter
wasm                     22.1.0              GraalWasm                     Experimental
[mydev@fedora /]$
```

```
[mydev@fedora /]$ rustup -V
rustup 1.24.3 (ce5817a94 2021-05-31)
info: This is the version for the rustup toolchain manager, not the rustc compiler.
info: The currently active `rustc` version is `rustc 1.61.0 (fe5b13d68 2022-05-18)`
[mydev@fedora /]$
[mydev@fedora /]$ rustc -V
rustc 1.61.0 (fe5b13d68 2022-05-18)
[mydev@fedora /]$
[mydev@fedora /]$ go version
go version go1.18.2 linux/arm64
[mydev@fedora /]$
[mydev@fedora /]$ dotnet --version
6.0.300
```

...

5.8 Good resources for Python implementation

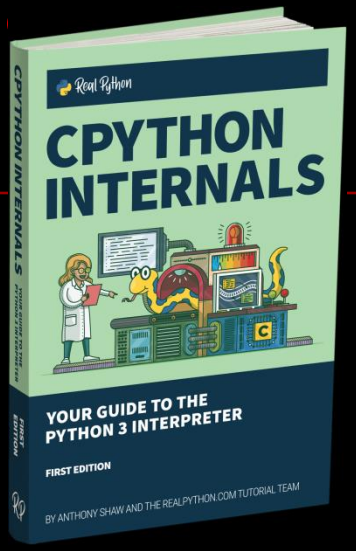
- https://wiki.python.org/moin/PythonImplementations
- https://pyfound.blogspot.com/2021/05/the-2021-python-language-summit.html
- …

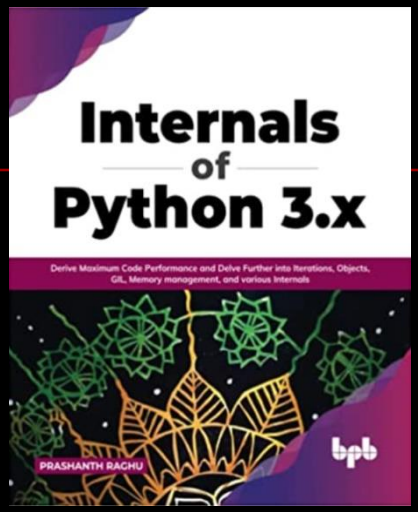
II. C-based implementation

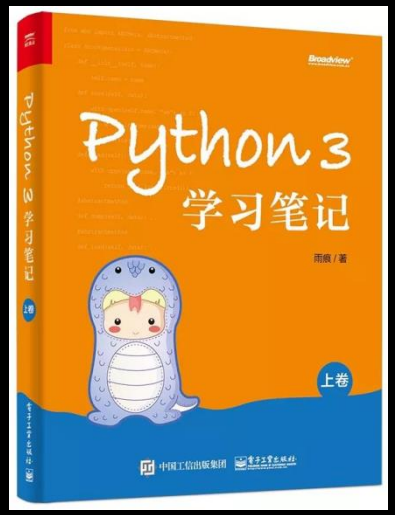
1) Overview

- …

Good Resources







- ...

2) Make CPython Faster

Bottlenecks of current CPython implementation

- No JIT
- GIL

  https://en.wikipedia.org/wiki/Global_interpreter_lock
- …

Plan from the Father of Python

- **https://thenewstack.io/guido-van-rossums-ambitious-plans-for-improving-python-performance/**

- **https://zhuanlan.zhihu.com/p/399398829**

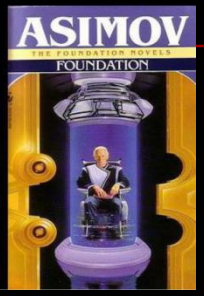
Python 之父立 Flag：明年要把 Python 速度提高 2 倍！

- **https://www.zdnet.com/article/python-programming-we-want-to-make-the-language-twice-as-fast-says-its-creator/**

> Microsoft currently has five core developers who contribute to the development of CPython, including Brett Cannon, Steve Dower, Guido van Rossum, Eric Snow, and Barry Warsaw — all veterans in the Python core developer community.

The “Shannon Plan”

- Posted to GitHub and python-dev last October
  - github.com/markshannon/faster-cpython
- Based on experience with “HotPy”, “HoyPy 2”
- Promises 5x in 4 years (1.5x per year)
- Looking for funding



https://github.com/markshannon/faster-cpython/blob/master/plan.md

The stages to high performance

Stage 1 -- Python 3.10

The key improvement for 3.10 will be an adaptive, specializing interpreter. The interpreter will adapt to types and values during execution, exploiting type stability in the program, without needing runtime code generation.

Stage 2 -- Python 3.11

This stage will make many improvements to the runtime and key objects. Stage two will be characterized by lots of "tweaks", rather than any "headline" improvement. The planned improvements include:

- Improved performance for integers of less than one machine word.
- Improved peformance for binary operators.
- Faster calls and returns, through better handling of frames.
- Better object memory layout and reduced memory management overhead.
- Zero overhead exception handling.
- Further enhancements to the interpreter
- Other small enhancements.

Stage 3 -- Python 3.12 (requires runtime code generation)

Simple "JIT" compiler for small regions. Compile small regions of specialized code, using a relatively simple, fast compiler.

Stage 4 -- Python 3.13 (requires runtime code generation)

Extend regions for compilation. Enhance compiler to generate superior machine code.

https://github.com/markshannon/faster-cpython/blob/master/tiers.md

Making CPython Faster

- https://pyfound.blogspot.com/2021/05/the-2021-python-language-summit-making.html

Seven months ago, Guido van Rossum left a brief retirement to work at Microsoft. He was given the freedom to pick a project and decided to work on making CPython faster. Microsoft will be funding a small team consisting of Guido van Rossum, Mark Shannon, Eric Snow, and possibly others.

The team will:

- Collaborate fully and openly with CPython's core developers
- Make incremental changes to CPython
- Take care of maintenance and support
- Keep all project-specific repos open
- Have all discussions in trackers on open GitHub repos

The team will need to work within some constraints. They'll need to keep code maintainable, not break stable ABI compatibility, not break limited API compatibility, and not break or slow down extreme cases, such as pushing a million cases on the eval stack. Although they won't be able to change the data model, they will be able to change:

- The byte code
- The compiler
- The internals of a lot of objects

Who Will Benefit?

You'll benefit from the speed increase if you:

- Run CPU-intensive pure Python code
- Use tools or websites built in CPython

You might not benefit from the speed increase if you:

- Rewrote your code in C, Cython, C++, or similar to increase speed already (e.g. NumPy, TensorFlow)
- Have code that is mostly waiting for I/O
- Use multithreading
- Need to make your code more algorithmically efficient first

- **Specializing Adaptive Interpreter**
  https://www.python.org/dev/peps/pep-0659/

Abstract

In order to perform well, virtual machines for dynamic languages must specialize the code that they execute to the types and values in the program being run. This specialization is often associated with "JIT" compilers, but is beneficial even without machine code generation.

A specializing, adaptive interpreter is one that speculatively specializes on the types or values it is currently operating on, and adapts to changes in those types and values.

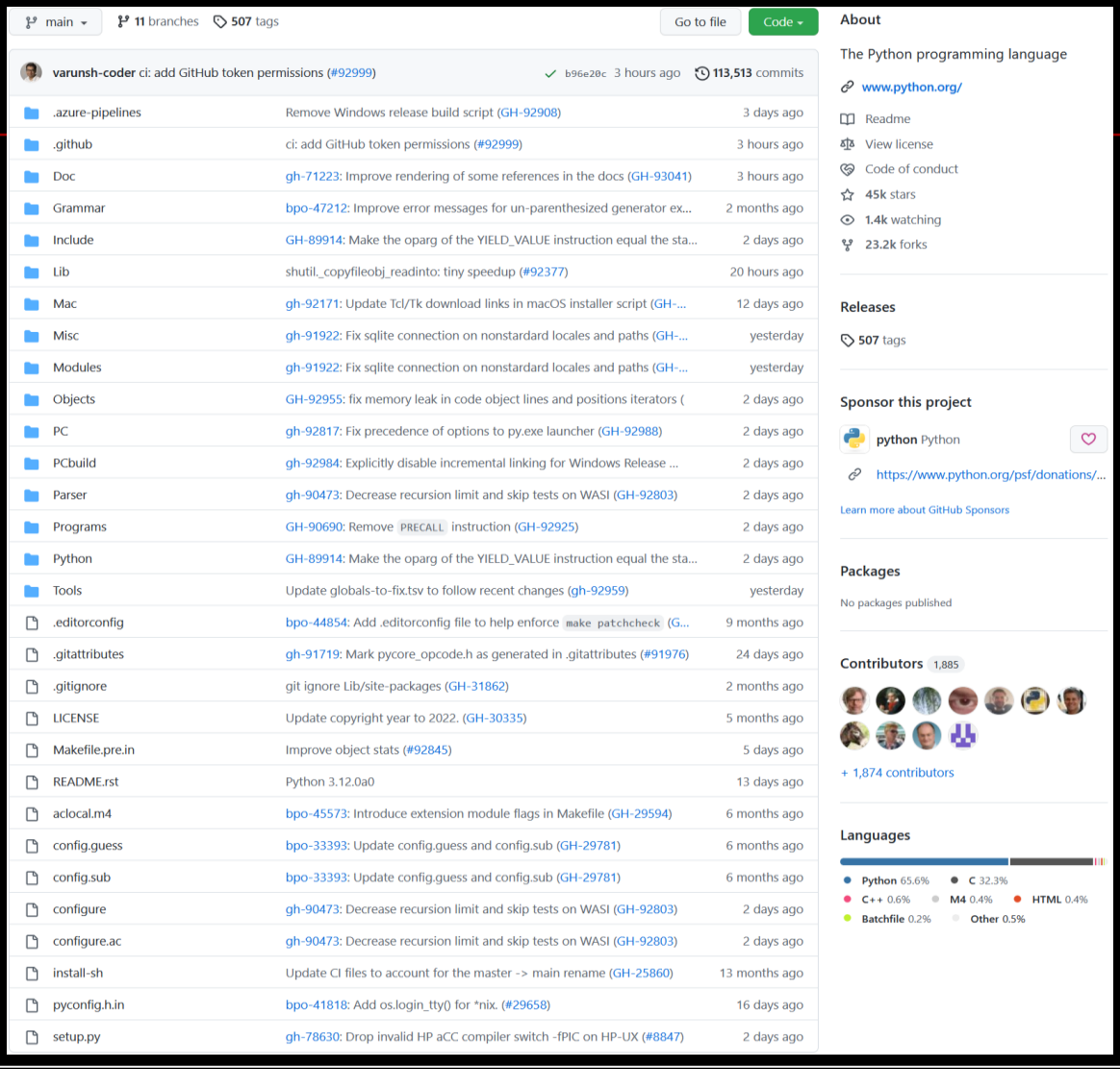
Specialization gives us improved performance, and adaptation allows the interpreter to rapidly change when the pattern of usage in a program alters, limiting the amount of additional work caused by mis-specialization.

This PEP proposes using a specializing, adaptive interpreter that specializes code aggressively, but over a very small region, and is able to adjust to mis-specialization rapidly and at low cost.

Adding a specializing, adaptive interpreter to CPython will bring significant performance improvements. It is hard to come up with meaningful numbers, as it depends very much on the benchmarks and on work that has not yet happened. Extensive experimentation suggests speedups of up to 50%. Even if the speedup were only 25%, this would still be a worthwhile enhancement.

Project

- **https://github.com/python/cpython**



main ▾ | 11 branches | 507 tags | Go to file | Code ▾

varunsh-coder ci: add GitHub token permissions (#92999) — b96e20c 3 hours ago — 113,513 commits

.azure-pipelines	Remove Windows release build script (GH-92908)	3 days ago
.github	ci: add GitHub token permissions (#92999)	3 hours ago
Doc	gh-71223: Improve rendering of some references in the docs (GH-93041)	3 hours ago
Grammar	bpo-47212: Improve error messages for un-parenthesized generator ex...	2 months ago
Include	GH-89914: Make the oparg of the YIELD_VALUE instruction equal the sta...	2 days ago
Lib	shutil._copyfileobj_readinto: tiny speedup (#92377)	20 hours ago
Mac	gh-92171: Update Tcl/Tk download links in macOS installer script (GH-...	12 days ago
Misc	gh-91922: Fix sqlite connection on nonstardard locales and paths (GH-...	yesterday
Modules	gh-91922: Fix sqlite connection on nonstardard locales and paths (GH-...	yesterday
Objects	GH-92955: fix memory leak in code object lines and positions iterators (	2 days ago
PC	gh-92817: Fix precedence of options to py.exe launcher (GH-92988)	2 days ago
PCbuild	gh-92984: Explicitly disable incremental linking for Windows Release ...	2 days ago
Parser	gh-90473: Decrease recursion limit and skip tests on WASI (GH-92803)	2 days ago
Programs	GH-90690: Remove `PRECALL` instruction (GH-92925)	2 days ago
Python	GH-89914: Make the oparg of the YIELD_VALUE instruction equal the sta...	2 days ago
Tools	Update globals-to-fix.tsv to follow recent changes (gh-92959)	yesterday
.editorconfig	bpo-44854: Add .editorconfig file to help enforce `make patchcheck` (G...	9 months ago
.gitattributes	gh-91719: Mark pycore_opcode.h as generated in .gitattributes (#91976)	24 days ago
.gitignore	git ignore Lib/site-packages (GH-31862)	2 months ago
LICENSE	Update copyright year to 2022. (GH-30335)	5 months ago
Makefile.pre.in	Improve object stats (#92845)	5 days ago
README.rst	Python 3.12.0a0	13 days ago
aclocal.m4	bpo-45573: Introduce extension module flags in Makefile (GH-29594)	6 months ago
config.guess	bpo-33393: Update config.guess and config.sub (GH-29781)	6 months ago
config.sub	bpo-33393: Update config.guess and config.sub (GH-29781)	6 months ago
configure	gh-90473: Decrease recursion limit and skip tests on WASI (GH-92803)	2 days ago
configure.ac	gh-90473: Decrease recursion limit and skip tests on WASI (GH-92803)	2 days ago
install-sh	Update CI files to account for the master -> main rename (GH-25860)	13 months ago
pyconfig.h.in	bpo-41818: Add os.login_tty() for *nix. (#29658)	16 days ago
setup.py	gh-78630: Drop invalid HP aCC compiler switch -fPIC on HP-UX (#8847)	2 days ago

About

The Python programming language

www.python.org/

Readme
View license
Code of conduct
45k stars
1.4k watching
23.2k forks

Releases

507 tags

Sponsor this project

python Python

https://www.python.org/psf/donations/...

Learn more about GitHub Sponsors

Packages

No packages published

Contributors 1,885

+ 1,874 contributors

Languages

Python 65.6% | C 32.3% | C++ 0.6% | M4 0.4% | HTML 0.4% | Batchfile 0.2% | Other 0.5%

Stats

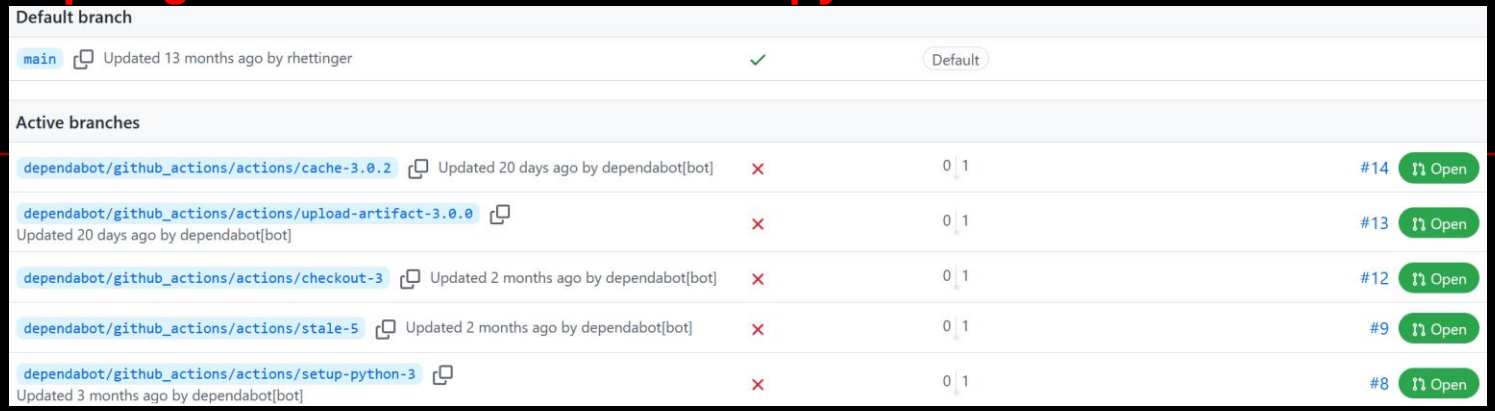
```
[mydev@fedora cpython-main]$ git branch
* main
[mydev@fedora cpython-main]$ tokei
```

Language	Files	Lines	Code	Comments	Blanks
Assembly	1	46	18	26	2
GNU Style Assembly	5	1930	1421	305	204
Autoconf	15	5537	3354	1146	1037
BASH	2	56	44	4	8
Batch	32	2270	1902	7	361
C	325	489821	381675	58125	50021
C Header	442	196378	168261	11381	16736
C Shell	1	26	12	5	9
C++	6	4316	3571	224	521
CSS	1	112	81	6	25
D	5	83	74	1	8
Fish	1	66	40	13	13
INI	2	197	121	28	48
JavaScript	2	140	121	9	10
JSON	20	16316	16313	0	3
Lisp	1	692	502	81	109
Makefile	4	455	326	45	84
Module-Definition	6	592	557	12	23
MSBuild	13	1228	1027	99	102
Objective-C	7	794	635	61	98
PowerShell	7	618	358	195	65
Prolog	1	24	24	0	0
Python	2069	903760	727160	60158	116442
ReStructuredText	654	379735	269198	0	110537
Shell	6	889	550	218	121
SVG	9	9	9	0	0
Plain Text	150	107439	0	104632	2807
TOML	64	2413	2250	92	71
VBScript	1	1	0	1	0
Visual Studio Pro|	50	6760	6700	16	44
Visual Studio Sol|	2	1535	1534	0	1
XSL	2	33	16	15	2
XML	59	31334	31258	18	58
HTML	12	2104	1991	6	107
|- CSS	4	39	39	0	0
|- JavaScript	4	245	200	13	32
(Total)		2388	2230	19	139
Markdown	5	611	0	446	165
|- C	1	3	2	1	0
|- Python	2	41	34	5	2
|- Shell	1	33	29	0	4
(Total)		688	65	452	171
Total	3982	2158681	1621407	237394	299880

```
[mydev@fedora cpython-main]$
```

Current Status

- https://github.com/markshannon/cpython



...

https://github.com/faster-cpython/ideas

- https://pyfound.blogspot.com/2022/05/the-2022-python-language-summit_2.html

Python 3.11, if you haven't heard, is fast. Over the past year, Microsoft has funded a team - led by core developers Mark Shannon and Guido van Rossum - to work full-time on making CPython faster. With additional funding from Bloomberg, and help from a wide range of other contributors from the community, the results have borne fruit. On the pyperformance benchmarks at the time of the beta release, Python 3.11 was around 1.25x faster than Python 3.10, a phenomenal achievement.

...

The gains Shannon's team has achieved are hugely impressive, and likely to benefit the community as a whole in a profound way. But various problems lie on the horizon. Sam Gross's proposal for a version of CPython without the Global Interpreter Lock (the `nogil` fork) has potential for speeding up multithreaded Python code in very different ways to the Faster CPython team's work - but it could also be problematic for some of the optimisations that have already been implemented, many of which assume that the GIL exists. Eric Snow's dream of achieving multiple subinterpreters within a single process, meanwhile, will have a smaller performance impact on single-threaded code compared to `nogil`, but could still create some minor complications for Shannon's team.

3) Cinder

- https://github.com/facebookincubator/cinder

  Meta's internal performance-oriented production version of CPython.

Cinder is Meta's internal performance-oriented production version of CPython 3.8. It contains a number of performance optimizations, including bytecode inline caching, eager evaluation of coroutines, a method-at-a-time JIT, and an experimental bytecode compiler that uses type annotations to emit type-specialized bytecode that performs better in the JIT.

Cinder is powering Instagram, where it started, and is increasingly used across more and more Python applications in Meta.

For more information on CPython, see `README.cpython.rst`.

- Docker-based Dev Env
-

Source: https://www.viehland.com/PyCon_2021.pdf

- Currently only support X64.
- https://trycinder.com/
- …

What's Special about it?

Immortal Instances

Instagram uses a multi-process webserver architecture; the parent process starts, performs initialization work (e.g. loading code), and forks tens of worker processes to handle client requests. Worker processes are restarted periodically for a number of reasons (e.g. memory leaks, code deployments) and have a relatively short lifetime. In this model, the OS must copy the entire page containing an object that was allocated in the parent process when the object's reference count is modified. In practice, the objects allocated in the parent process outlive workers; all the work related to reference counting them is unnecessary.

Instagram has a very large Python codebase and the overhead due to copy-on-write from reference counting long-lived objects turned out to be significant. We developed a solution called "immortal instances" to provide a way to opt-out objects from reference counting. See Include/object.h for details. This feature is controlled by defining Py_IMMORTAL_INSTANCES and is enabled by default in Cinder. This was a large win for us in production (~5%), but it makes straight-line code slower. Reference counting operations occur frequently and must check whether or not an object participates in reference counting when this feature is enabled.

Shadowcode

"Shadowcode" or "shadow bytecode" is our implementation of a specializing interpreter. It observes particular optimizable cases in the execution of generic Python opcodes and (for hot functions) dynamically replaces those opcodes with specialized versions. The core of shadowcode lives in `Python/shadowcode.c`, though the implementations for the specialized bytecodes are in `Python/ceval.c` with the rest of the eval loop. Shadowcode-specific tests are in `Lib/test/test_shadowcode.py`.

It is similar in spirit to the specializing adaptive interpreter (PEP-659) that will be built into CPython 3.11.

Await-aware function calls

The Instagram Server is an async-heavy workload, where each web request may trigger hundreds of thousands of async tasks, many of which can be completed without suspension (e.g. thanks to memoized values).

We extended the vectorcall protocol to pass a new flag, `_Py_AWAITED_CALL_MARKER`, indicating the caller is immediately awaiting this call.

When used with async function calls that are immediately awaited, we can immediately (eagerly) evaluate the called function, up to completion, or up to its first suspension. If the function completes without suspending, we are able to return the value immediately, with no extra heap allocations.

When used with async gather, we can immediately (eagerly) evaluate the set of passed awaitables, potentially avoiding the cost of creation and scheduling of multiple tasks for coroutines that could be completed synchronously, completed futures, memoized values, etc.

These optimizations resulted in a significant (~5%) CPU efficiency improvement.

This is mostly implemented in `Python/ceval.c`, via a new vectorcall flag `_Py_AWAITED_CALL_MARKER`, indicating the caller is immediately awaiting this call. Look for uses of the `IS_AWAITED()` macro and this vectorcall flag, as well as the `_PyEval_EvalEagerCoro` function.

Static Python

Static Python is a bytecode compiler that makes use of type annotations to emit type-specialized and type-checked Python bytecode. Used along with the Cinder JIT, it can deliver performance similar to MyPyC or Cython in many cases, while offering a pure-Python developer experience (normal Python syntax, no extra compilation step). Static Python plus Cinder JIT achieves 18x the performance of stock CPython on a typed version of the Richards benchmark. At Instagram we have successfully used Static Python in production to replace all Cython modules in our primary webserver codebase, with no performance regression.

The Static Python compiler is built on top of the Python `compiler` module that was removed from the standard library in Python 3 and has since been maintained and updated externally; this compiler is incorporated into Cinder in `Lib/compiler`. The Static Python compiler is implemented in `Lib/compiler/static/`, and its tests are in `Lib/test/test_compiler/test_static.py`.

Classes defined in Static Python modules are automatically given typed slots (based on inspection of their typed class attributes and annotated assignments in `__init__`), and attribute loads and stores against instances of these types use new `STORE_FIELD` and `LOAD_FIELD` opcodes, which in the JIT become direct loads/stores from/to a fixed memory offset in the object, with none of the indirection of a `LOAD_ATTR` or `STORE_ATTR`. Classes also gain vtables of their methods, for use by the `INVOKE_*` opcodes mentioned below. The runtime support for these features is located in `Include/classloader.h` and `Python/classloader.c`.

A static Python function begins with a new `CHECK_ARGS` opcode which checks that the supplied arguments' types match the type annotations, and raises `TypeError` if not. Calls from a static Python function to another static Python function will skip this opcode (since the types are already validated by the compiler). Static to static calls can also avoid much of the overhead of a typical Python function call. We emit an `INVOKE_FUNCTION` or `INVOKE_METHOD` opcode which carries with it metadata about the called function or method; this plus optionally immutable modules (via `StrictModule`) and types (via `cinder.freeze_type()`, which we currently apply to all types in strict and static modules in our import loader, but in future may become an inherent part of Static Python) and compile-time knowledge of the callee signature allow us to (in the JIT) turn many Python function calls into direct calls to a fixed memory address using the x64 calling convention, with little more overhead than a C function call.

Static Python is still gradually typed, and supports code that is only partially annotated or uses unknown types by falling back to normal Python dynamic behavior. In some cases (e.g. when a value of statically-unknown type is returned from a function with a return annotation), a runtime `CAST` opcode is inserted which will raise `TypeError` if the runtime type does not match the expected type.

Static Python also supports new types for machine integers, bools, doubles, and vectors/arrays. In the JIT these are handled as unboxed values, and e.g. primitive integer arithmetic avoids all Python overhead. Some operations on builtin types (e.g. list or dictionary subscript or `len()`) are also optimized.

Cinder supports gradual adoption of static modules via a strict/static module loader that can automatically detect static modules and load them as static with cross-module compilation. The loader will look for `import __static__` and `import __strict__` annotations at the top of a file, and compile modules appropriately. To enable the loader, you have one of three options:

1. Explicitly install the loader at the top level of your application via `from compiler.strict.loader import install; install()`.
2. Set `PYTHONINSTALLSTRICTLOADER=1` in your env.
3. Run `./python -X install-strict-loader application.py`.

Alternatively, you can compile all code statically by using `./python -m compiler --static some_module.py`, which will compile the module as static Python and execute it.

See `CinderDoc/static_python.rst` for more detailed documentation.

Strict Modules

Strict modules is a few things rolled into one:

1. A static analyzer capable of validating that executing a module's top-level code will not have side effects visible outside that module.

2. An immutable `StrictModule` type usable in place of Python's default module type.

3. A Python module loader capable of recognizing modules opted in to strict mode (via an `import __strict__` at the top of the module), analyzing them to validate no import side effects, and populating them in `sys.modules` as a `StrictModule` object.

JIT

The Cinder JIT

The Cinder JIT is a method-at-a-time custom JIT implemented in C++. It is enabled via the `-X jit` flag or the `PYTHONJIT=1` environment variable. It supports almost all Python opcodes, and can achieve 1.5-4x speed improvements on many Python performance benchmarks.

By default when enabled it will JIT-compile every function that is ever called, which may well make your program slower, not faster, due to overhead of JIT-compiling rarely-called functions. The option `-X jit-list-file=/path/to/jitlist.txt` or `PYTHONJITLISTFILE=/path/to/jitlist.txt` can point it to a text file containing fully qualified function names (in the form `path.to.module:funcname` or `path.to.module:ClassName.method_name`), one per line, which should be JIT-compiled. We use this option to compile only a set of hot functions derived from production profiling data. (A more typical approach for a JIT would be to dynamically compile functions as they are observed to be called frequently. It hasn't yet been worth it for us to implement this, since our production architecture is a pre-fork webserver, and for memory sharing reasons we wish to do all of our JIT compiling up front in the initial process before workers are forked, which means we can't observe the workload in-process before deciding which functions to JIT-compile.)

The JIT lives in the `Jit/` directory, and its C++ tests live in `RuntimeTests/` (run these with `make testruntime`). There are also some Python tests for it in `Lib/test/test_cinderjit.py`; these aren't meant to be exhaustive, since we run the entire CPython test suite under the JIT via `make testcinder_jit`; they cover JIT edge cases not otherwise found in the CPython test suite.

See `Jit/pyjit.cpp` for some other `-X` options and environment variables that influence the behavior of the JIT. There is also a `cinderjit` module defined in that file which exposes some JIT utilities to Python code (e.g. forcing a specific function to compile, checking if a function is compiled, disabling the JIT). Note that `cinderjit.disable()` only disables future compilation; it immediately compiles all known functions and keeps existing JIT-compiled functions.

The JIT first lowers Python bytecode to a high-level intermediate representation (HIR); this is implemented in `Jit/hir/`. HIR maps reasonably closely to Python bytecode, though it is a register machine instead of a stack machine, it is a bit lower level, it is typed, and some details that are obscured by Python bytecode but important for performance (notably reference counting) are exposed explicitly in HIR. HIR is transformed into SSA form, some optimization passes are performed on it, and then reference counting operations are automatically inserted into it according to metadata about the refcount and memory effects of HIR opcodes.

HIR is then lowered to a low-level intermediate representation (LIR), which is an abstraction over assembly, implemented in `Jit/lir/`. In LIR we do register allocation, some additional optimization passes, and then finally LIR is lowered to assembly (in `Jit/codegen/`) using the excellent asmjit library.

The JIT is in its early stages. While it can already eliminate interpreter loop overhead and offers significant performance improvements for many functions, we've only begun to scratch the surface of possible optimizations. Many common compiler optimizations are not yet implemented. Our prioritization of optimizations is largely driven by the characteristics of the Instagram production workload.

- **Asmjit**

https://asmjit.com/

AsmJit is a lightweight library for machine code generation written in C++ language. It can generate machine code for X86, X86_64, and AArch64 architectures and supports baseline instructions and all recent extensions. It has a type-safe API that allows C++ compiler to do semantic checks at compile-time even before the assembled code is generated or executed. It also provides an optional register allocator that makes it easy to start generating code without a significant development effort.

GitHub Projects

AsmJit project, as the name implies, started as a library to allow JIT code generation and execution. However, AsmJit evolved and now contains features that are far beyond the initial scope. AsmJit now consists of multiple projects:

- AsmJit - AsmJit library, the main project.
- AsmTK - AsmTK library, toolkit that implements some functionality on top of AsmJit, for example assembler parser.
- AsmDB - Assembler database in JSON format, used to generate AsmJit tables.

Online Tools

- AsmGrid - A grid view of assembler instructions (AsmDB) and their latencies (generated by CULT tool).

Highlights

- Lightweight - ~300kB compiled binary with all built-in features depending on C++ compiler and optimization level.
- Modular - Unneeded features can be disabled at compile-time to save space.
- Zero dependencies - No external libraries nor STL containers are used, easy to embed and/or link statically.
- No exceptions & RTTI - AsmJit doesn't use exceptions, but allows to attach a throwable ErrorHandler if required.

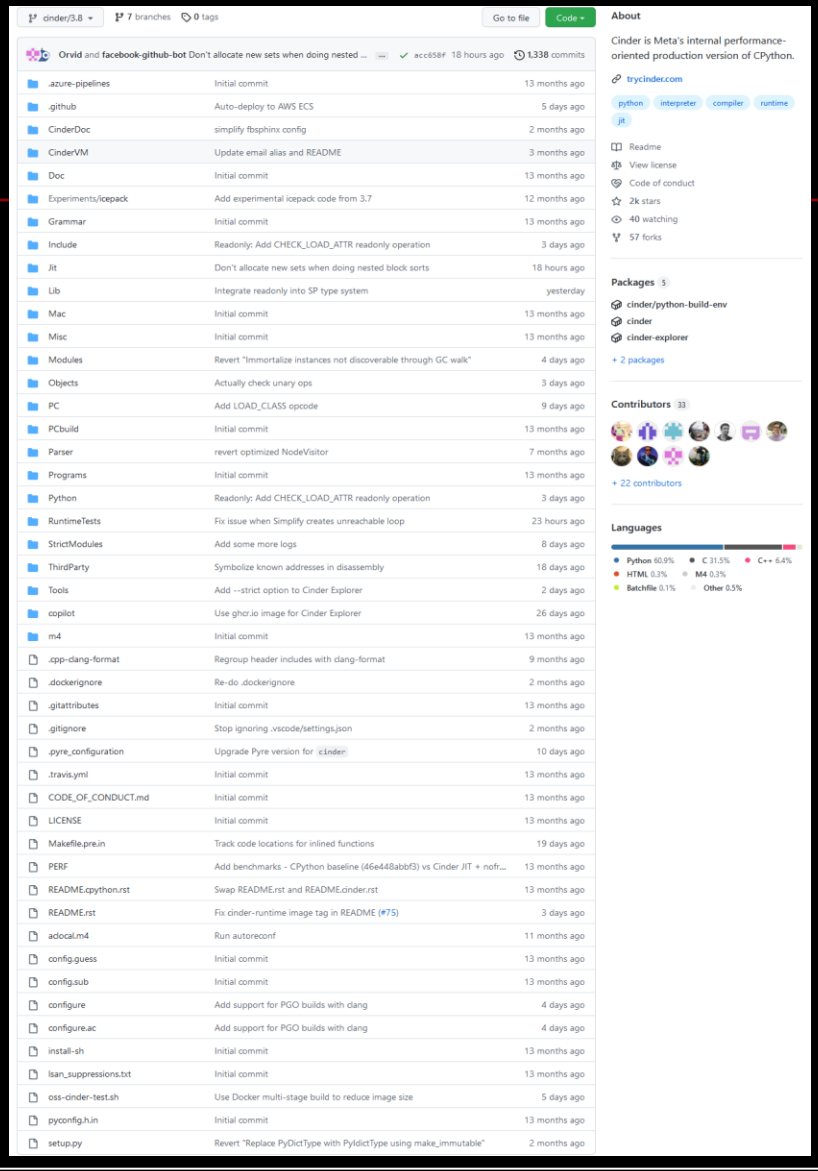
Key Features

- Complete X86/X64 instruction set - MMX, SSE+, BMI+, AVX+, FMA+, AVX-512+, AMX, privileged instructions, and other recently added ISA extensions are supported.
- Complete AArch64 instruction set - Baseline instructions and ASIMD extensions.
- Dynamic architecture that allows users to construct instructions and operands at runtime - to inspect them, to validate them, and to emit them.
- Different emitters providing various abstraction levels - Assembler, Builder, and Compiler.
- Support for sections that can be used to separate code and data or to use separate buffers during code generation.
- Built-in logging (includes formatting) and user-friendly error handling.
- JIT memory allocator with malloc-like API for JIT code generation and execution.
- Compiler can be used to emit large chunks of code with the help of a built-in register allocator. This is a unique feature that makes it almost instant to start generating code and have results.

https://github.com/asmjit

Project

-



cinder/3.8 · 7 branches · 0 tags · Go to file · Code

Orvid and facebook-github-bot Don't allocate new sets when doing nested ... … acc658f 18 hours ago 1,338 commits

.azure-pipelines	Initial commit	13 months ago
.github	Auto-deploy to AWS ECS	5 days ago
CinderDoc	simplify fbsphinx config	2 months ago
CinderVM	Update email alias and README	3 months ago
Doc	Initial commit	13 months ago
Experiments/icepack	Add experimental icepack code from 3.7	12 months ago
Grammar	Initial commit	13 months ago
Include	Readonly: Add CHECK_LOAD_ATTR readonly operation	3 days ago
Jit	Don't allocate new sets when doing nested block sorts	18 hours ago
Lib	Integrate readonly into SP type system	yesterday
Mac	Initial commit	13 months ago
Misc	Initial commit	13 months ago
Modules	Revert "Immortalize instances not discoverable through GC walk"	4 days ago
Objects	Actually check unary ops	3 days ago
PC	Add LOAD_CLASS opcode	9 days ago
PCbuild	Initial commit	13 months ago
Parser	revert optimized NodeVisitor	7 months ago
Programs	Initial commit	13 months ago
Python	Readonly: Add CHECK_LOAD_ATTR readonly operation	3 days ago
RuntimeTests	Fix issue when Simplify creates unreachable loop	23 hours ago
StrictModules	Add some more logs	8 days ago
ThirdParty	Symbolize known addresses in disassembly	18 days ago
Tools	Add --strict option to Cinder Explorer	2 days ago
copilot	Use ghcr.io image for Cinder Explorer	26 days ago
m4	Initial commit	13 months ago
.cpp-clang-format	Regroup header includes with clang-format	9 months ago
.dockerignore	Re-do .dockerignore	2 months ago
.gitattributes	Initial commit	13 months ago
.gitignore	Stop ignoring .vscode/settings.json	2 months ago
.pyre_configuration	Upgrade Pyre version for `cinder`	10 days ago
.travis.yml	Initial commit	13 months ago
CODE_OF_CONDUCT.md	Initial commit	13 months ago
LICENSE	Initial commit	13 months ago
Makefile.pre.in	Track code locations for inlined functions	19 days ago
PERF	Add benchmarks - CPython baseline (46e448abbf3) vs Cinder JIT + nofr...	13 months ago
README.cpython.rst	Swap README.rst and README.cinder.rst	13 months ago
README.rst	Fix cinder-runtime image tag in README (#75)	3 days ago
aclocal.m4	Run autoreconf	11 months ago
config.guess	Initial commit	13 months ago
config.sub	Initial commit	13 months ago
configure	Add support for PGO builds with clang	4 days ago
configure.ac	Add support for PGO builds with clang	4 days ago
install-sh	Initial commit	13 months ago
lsan_suppressions.txt	Initial commit	13 months ago
oss-cinder-test.sh	Use Docker multi-stage build to reduce image size	5 days ago
pyconfig.h.in	Initial commit	13 months ago
setup.py	Revert "Replace PyDictType with PyIdictType using make_immutable"	2 months ago

About

Cinder is Meta's internal performance-oriented production version of CPython.

trycinder.com

python · interpreter · compiler · runtime · jit

Readme · View license · Code of conduct · 2k stars · 40 watching · 57 forks

Packages 5

cinder/python-build-env
cinder
cinder-explorer
+ 2 packages

Contributors 33

+ 22 contributors

Languages

Python 60.9% · C 31.5% · C++ 6.4% · HTML 0.3% · M4 0.3% · Batchfile 0.1% · Other 0.5%

Stats

```
[mydev@fedora cinder-3.8]$ git branch
* cinder/3.8
[mydev@fedora cinder-3.8]$ tokei
```

Language	Files	Lines	Code	Comments	Blanks
Assembly	1	48	18	26	4
GNU Style Assembly	5	1930	1421	305	204
Autoconf	24	7352	5111	1075	1166
Automake	3	572	419	80	73
BASH	3	76	59	5	12
Batch	34	2146	1789	4	353
C	334	468275	361101	56429	50745
C Header	685	323660	254589	37015	32056
CMake	10	1610	1052	331	227
C Shell	1	37	21	7	9
C++	353	166384	126379	17339	22666
C++ Header	1	22109	15660	3846	2603
CSS	11	1345	899	127	319
D	7	107	90	5	12
Fish	1	75	47	15	13
Go	1	27	24	0	3
Haskell	1	65	55	1	9
INI	2	202	118	38	46
JavaScript	48	26267	19885	3070	3312
JSON	9	17245	17242	0	3
Lisp	1	692	502	81	109
Makefile	6	450	313	50	87
Module-Definition	8	1401	1364	14	23
MSBuild	12	1076	896	95	85
Objective-C	7	794	635	61	98
PowerShell	6	610	352	194	64
Prolog	1	24	24	0	0
Python	2415	920838	741422	60630	118786
ReStructuredText	627	326339	230978	0	95361
Shell	28	1632	884	560	188
SVG	13	340	336	0	4
Plain Text	194	115914	0	112306	3608
TOML	1	6	6	0	0
VBScript	1	1	0	1	0
Visual Studio Pro|	65	9162	9108	9	45
Visual Studio Sol|	7	1706	1702	0	4
XSL	1	5	5	0	0
Xcode Config	6	146	33	81	32
XML	58	417	357	7	53
YAML	3	236	154	48	34
HTML	17	2731	2062	530	139
|- CSS	7	172	171	0	1
|- JavaScript	5	206	183	3	20
(Total)		3109	2416	533	160
Markdown	39	12480	0	9383	3097
|- C++	6	2138	1412	318	408
|- JSON	1	121	121	0	0
|- Shell	1	1	1	0	0
|- XML	1	27	27	0	0
(Total)		14767	1561	9701	3505
Total	5050	2439197	1799027	304089	336081

```
[mydev@fedora cinder-3.8]$
```

Previous CPython Improvements at Instagram

- https://pyfound.blogspot.com/2021/05/the-2021-python-language-summit-cpython.html

...

Experimental Work

Instagram has tried some experimental changes as well. One big one was the JIT. They have a custom method at a time JIT. There is nearly full coverage for all of the opcodes. They do have some unsupported opcodes, but they are rare and not used in methods, such as `IMPORT_STAR`. There are a couple of intermediate representations. The front end lowers to an HIR where they do an SSA and have a ref count insertion pass as well as other optimization passes. After they go though the HIR level, they lower it to an LIR, which is closer to x64.

Another experimental idea is something they call static Python. It provides similar performance gains as MyPyC or Cython, but it works at runtime and has no extra compile steps. It starts with a new source loader that loads files marked with `import __static__`, and it supports cross module compilation across different source files. There are also new byte codes such as `INVOKE_FUNCTION` and `LOAD_FIELD` that can be bound tightly at runtime. It uses normal PEP 484 annotations.

InterOp needs to enforce types at the boundaries between untyped Python and static Python. If you call a typed function, then you might get a `TypeError`. Static Python has a whole new static compiler that uses the regular Python `ast` module and is based on the Python 2.x `compiler` package.

In addition, Pyro is an unannounced, experimental, from-scratch implementation that reuses the standard library. The main differences between Pyro and CPython are:

- Compacting garbage collection
- Tagged pointers
- Hidden classes

The C API is emulated for the PEP 384 subset for supporting C extensions.

...

Performance Results

Production improvements are difficult to measure because changes have been incremental over time, but they are estimated at between 20% and 30% overall. When Instagram was benchmarking, they used CPython 3.8 as the baseline and compared Cinder, Cinder with the JIT, and Cinder JIT noframe, which Instagram is not yet using in production but wants to move towards so they won't have to create Python frame objects for jitted code.

Cinder had good results on a large set of benchmarks and a 4x return on `richards` but did worse with others, particularly `2to3`, `python_startup`, and `python_startup_no_site`. This was probably because they JIT every single function when it's invoked the first time. They haven't yet made the changes to JIT a function when it becomes hot. They also haven't yet tested comparisons with Pypy.

III. Java-based implementation

1) Overview

- …

2) Jython

- http://www.jython.org //No new release since 2015…
-

Jython is a Java implementation of Python that combines expressive power with clarity. Jython is freely available for both commercial and non-commercial use and is distributed with source code under the PSF License v2. Jython is complementary to Java and is especially suited for the following tasks:

- Embedded scripting - Java programmers can add the Jython libraries to their system to allow end users to write simple or complicated scripts that add functionality to the application.
- Interactive experimentation - Jython provides an interactive interpreter that can be used to interact with Java packages or with running Java applications. This allows programmers to experiment and debug any Java system using Jython.
- Rapid application development - Python programs are typically 2-10x shorter than the equivalent Java program. This translates directly to increased programmer productivity. The seamless interaction between Python and Java allows developers to freely mix the two languages both during development and in shipping products.

Here is an example of running Python code inside a simple Java application

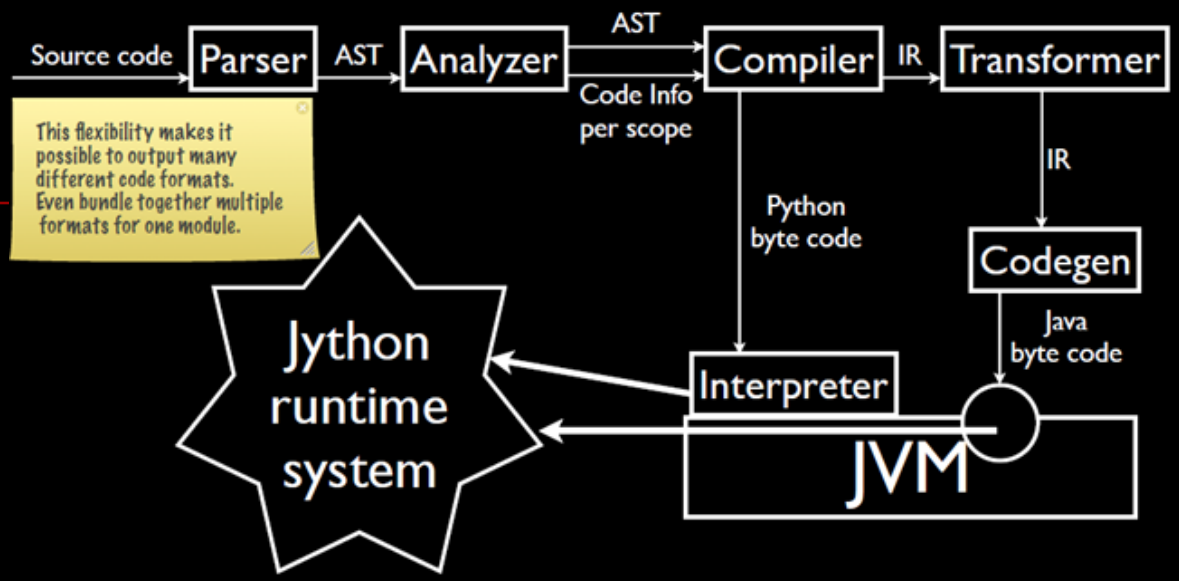
```
import org.python.util.PythonInterpreter;

public class JythonHelloWorld {
  public static void main(String[] args) {
    try(PythonInterpreter pyInterp = new PythonInterpreter()) {
      pyInterp.exec("print('Hello Python World!')");
    }
  }
}
```

Here is an example of using Java from Python code

```
from java.lang import System # Java import

print('Running on Java version: ' + System.getProperty('java.version'))
print('Unix time from Java: ' + str(System.currentTimeMillis()))
```

- …

How it works



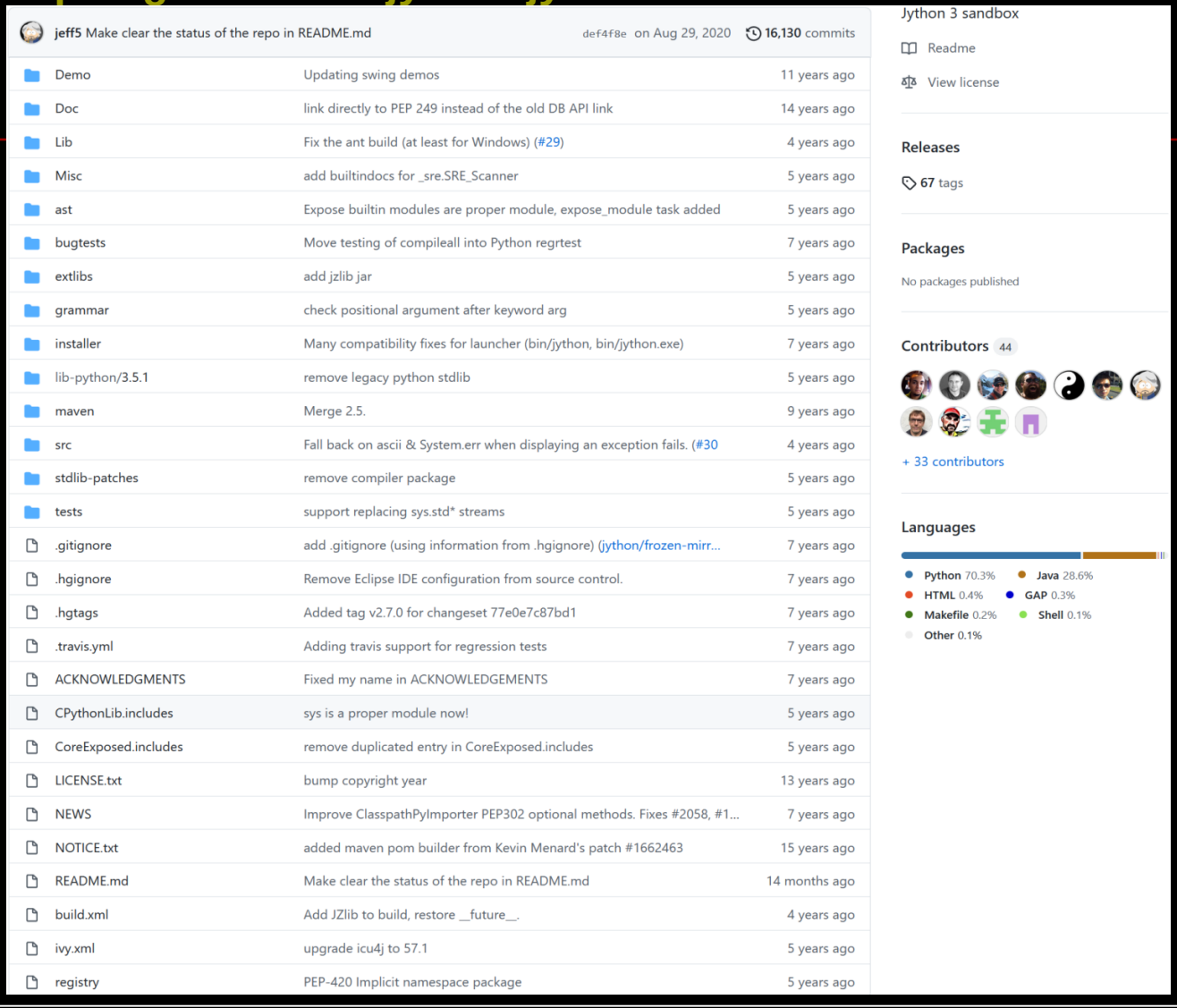
- **VOC**

  https://github.com/pybee/voc/

  A transpiler that converts Python code into Java bytecode.

Project

- https://github.com/jython/jython3



jeff5 Make clear the status of the repo in README.md — def4f8e on Aug 29, 2020 — 16,130 commits

Demo	Updating swing demos	11 years ago
Doc	link directly to PEP 249 instead of the old DB API link	14 years ago
Lib	Fix the ant build (at least for Windows) (#29)	4 years ago
Misc	add builtindocs for _sre.SRE_Scanner	5 years ago
ast	Expose builtin modules are proper module, expose_module task added	5 years ago
bugtests	Move testing of compileall into Python regrtest	7 years ago
extlibs	add jzlib jar	5 years ago
grammar	check positional argument after keyword arg	5 years ago
installer	Many compatibility fixes for launcher (bin/jython, bin/jython.exe)	7 years ago
lib-python/3.5.1	remove legacy python stdlib	5 years ago
maven	Merge 2.5.	9 years ago
src	Fall back on ascii & System.err when displaying an exception fails. (#30	4 years ago
stdlib-patches	remove compiler package	5 years ago
tests	support replacing sys.std* streams	5 years ago
.gitignore	add .gitignore (using information from .hgignore) (jython/frozen-mirr...	7 years ago
.hgignore	Remove Eclipse IDE configuration from source control.	7 years ago
.hgtags	Added tag v2.7.0 for changeset 77e0e7c87bd1	7 years ago
.travis.yml	Adding travis support for regression tests	7 years ago
ACKNOWLEDGMENTS	Fixed my name in ACKNOWLEDGEMENTS	7 years ago
CPythonLib.includes	sys is a proper module now!	5 years ago
CoreExposed.includes	remove duplicated entry in CoreExposed.includes	5 years ago
LICENSE.txt	bump copyright year	13 years ago
NEWS	Improve ClasspathPyImporter PEP302 optional methods. Fixes #2058, #1...	7 years ago
NOTICE.txt	added maven pom builder from Kevin Menard's patch #1662463	15 years ago
README.md	Make clear the status of the repo in README.md	14 months ago
build.xml	Add JZlib to build, restore __future__.	4 years ago
ivy.xml	upgrade icu4j to 57.1	5 years ago
registry	PEP-420 Implicit namespace package	5 years ago

Jython 3 sandbox

Readme

View license

Releases

67 tags

Packages

No packages published

Contributors 44

+ 33 contributors

Languages

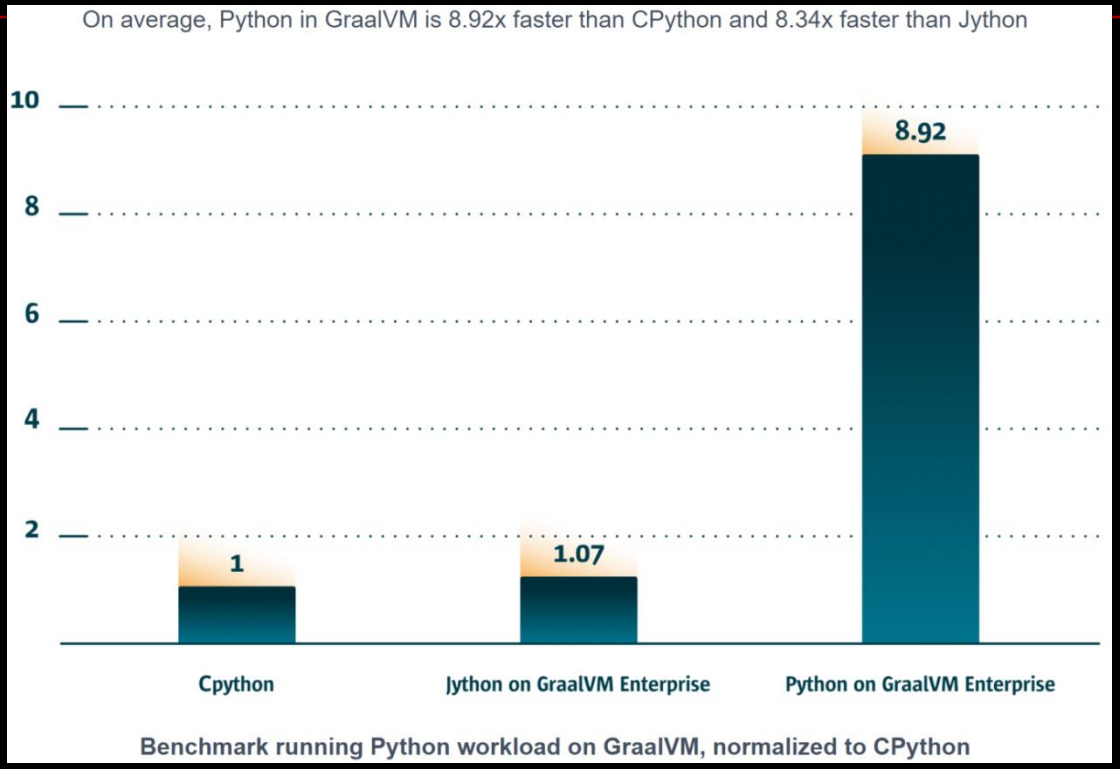
Python 70.3% Java 28.6%
HTML 0.4% GAP 0.3%
Makefile 0.2% Shell 0.1%
Other 0.1%

Stats

```
[mydev@fedora jython3]$ git branch
* master
[mydev@fedora jython3]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Autoconf                1           67           41            5           21
 BASH                    1          287          215           42           30
 Batch                   2            5            5            0            0
 C                       3          640          476           67           97
 C Header                1           24           24            0            0
 C Shell                 1           37           21            7            9
 CSS                     1            6            0            6            0
 Fish                    1           74           50           13           11
 Java                 1091       356230       276916        38176        41138
 Makefile                1         1746         1315          212          219
 Markdown                1           23            0           20            3
 Module-Definition       4          484          460            0           24
 Python               2220       817566       652056        59920       105590
 ReStructuredText        1          216          165            0           51
 Shell                   4          364          206           85           73
 Plain Text            136       169438            0       167640         1798
 VBScript                2           14           12            1            1
 XML                    13        33148        32584          324          240
-------------------------------------------------------------------------------
 HTML                    4         1437         1384            6           47
 |- CSS                  1            7            7            0            0
 |- JavaScript           2           17           16            1            0
 (Total)                            1461         1407            7           47
===============================================================================
 Total                3488      1381830       965953       266525       149352
===============================================================================
[mydev@fedora jython3]$
```

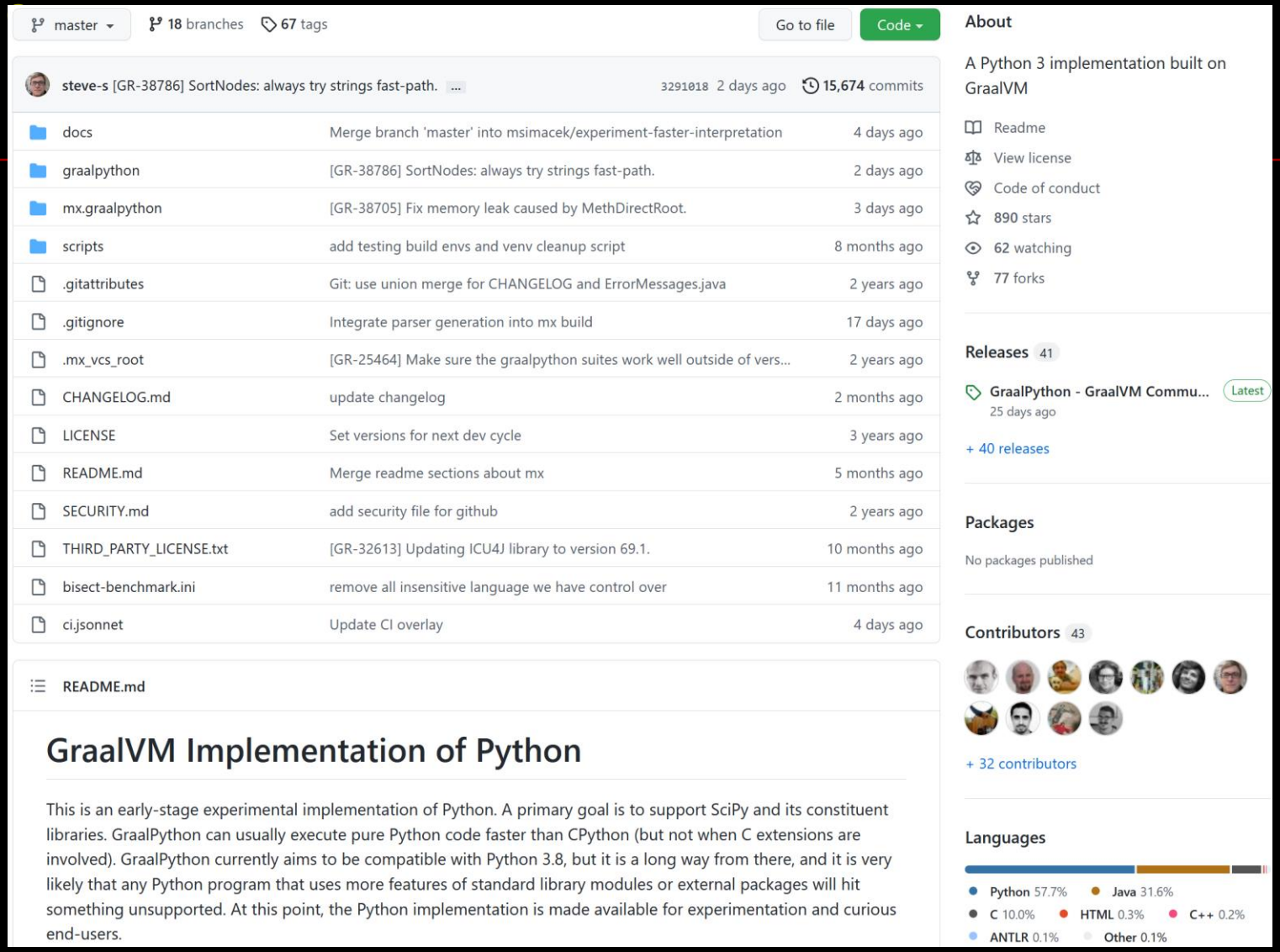
3) GraalPython

- https://github.com/oracle/graalpython
  A Python 3 implementation built on GraalVM.
- https://www.graalvm.org/python/



- Currently only support X64
- …

Project



master · 18 branches · 67 tags · Go to file · Code

steve-s [GR-38786] SortNodes: always try strings fast-path. ... 3291018 2 days ago 15,674 commits

docs	Merge branch 'master' into msimacek/experiment-faster-interpretation	4 days ago
graalpython	[GR-38786] SortNodes: always try strings fast-path.	2 days ago
mx.graalpython	[GR-38705] Fix memory leak caused by MethDirectRoot.	3 days ago
scripts	add testing build envs and venv cleanup script	8 months ago
.gitattributes	Git: use union merge for CHANGELOG and ErrorMessages.java	2 years ago
.gitignore	Integrate parser generation into mx build	17 days ago
.mx_vcs_root	[GR-25464] Make sure the graalpython suites work well outside of vers...	2 years ago
CHANGELOG.md	update changelog	2 months ago
LICENSE	Set versions for next dev cycle	3 years ago
README.md	Merge readme sections about mx	5 months ago
SECURITY.md	add security file for github	2 years ago
THIRD_PARTY_LICENSE.txt	[GR-32613] Updating ICU4J library to version 69.1.	10 months ago
bisect-benchmark.ini	remove all insensitive language we have control over	11 months ago
ci.jsonnet	Update CI overlay	4 days ago

README.md

GraalVM Implementation of Python

This is an early-stage experimental implementation of Python. A primary goal is to support SciPy and its constituent libraries. GraalPython can usually execute pure Python code faster than CPython (but not when C extensions are involved). GraalPython currently aims to be compatible with Python 3.8, but it is a long way from there, and it is very likely that any Python program that uses more features of standard library modules or external packages will hit something unsupported. At this point, the Python implementation is made available for experimentation and curious end-users.

About

A Python 3 implementation built on GraalVM

Readme
View license
Code of conduct
890 stars
62 watching
77 forks

Releases 41

GraalPython - GraalVM Commu... Latest
25 days ago

+ 40 releases

Packages

No packages published

Contributors 43

+ 32 contributors

Languages

Python 57.7% Java 31.6% C 10.0% HTML 0.3% C++ 0.2% ANTLR 0.1% Other 0.1%

Stats

```
[mydev@fedora graalpython-master]$ git branch
* master
[mydev@fedora graalpython-master]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    2           56           44            4            8
 Batch                   4           59           44            0           15
 C                      98        52737        39503         7596         5638
 C Header              140        70466        58825         7875         3766
 C Shell                 1           37           21            7            9
 CSS                     1            6            0            6            0
 D                       4           61           55            0            6
 Fish                    1           75           47           15           13
 INI                     1           42            7           28            7
 Java                 1340       400790       293975        68471        38344
 JSON                    3          475          475            0            0
 Makefile                4          333          182          113           38
 Module-Definition       4          569          547            0           22
 PowerShell              1          241          104          105           32
 Python               2187       871538       688801        71831       110906
 R                       1           78           20           49            9
 ReStructuredText        2          217          166            0           51
 Shell                  17          761          121          596           44
 Plain Text            471       120481            0       118327         2154
 VBScript                1            1            0            1            0
 XSL                     1            5            5            0            0
 XML                    56          412          352            7           53
-------------------------------------------------------------------------------
 HTML                    3         1707         1631            6           70
 |- CSS                 2           12           12            0            0
 |- JavaScript          2           11           10            1            0
 (Total)                           1730         1653            7           70
-------------------------------------------------------------------------------
 Markdown               17         2103            0         1635          468
 |- Java                3           38           31            0            7
 |- Python              3          139          104           14           21
 |- Shell               5           36           36            0            0
 (Total)                           2316          171         1649          496
===============================================================================
 Total                4360      1523486      1085118       276687       161681
===============================================================================
[mydev@fedora graalpython-master]$
```

IV. LLVM-based implementation

1) Overview

- …

VMKit

- https://vmkit.llvm.org/

The VMKit project is retired

You can still play with the last VMKit release, but the project is not more maintained. Moreover, the information on these pages may be out of date.

If you are interested in restarting the project, please contact Gaël Thomas

VMKit: a substrate for virtual machines

Current MREs are monolithic. Extending them to propose new features or reusing them to execute new languages is difficult. VMKit is a library that eases the development of new MREs and the process of experimenting with new mechanisms inside MREs. VMKit provides the basic components of MREs: a JIT compiler, a GC, and a thread manager.

VMKit relies on LLVM for compilation and MMTk to manage memory. Currently, a full Java virtual machine called J3 is distributed with VMKit.

Features

For the end user, VMKit provides:

- Precise garbage collection.
- Just-in-Time and Ahead-of-Time compilation.
- Portable on many architectures (x86, x64, ppc32, ppc64, arm).

For the MRE developer, VMKit provides:

- Relatively small code base (~ 20k loc per VM)
- Infrastructure for virtual machine research and development

Current Status

VMKit currently has a decent implementation of a JVM called J3. It executes large projects (e.g. OSGi Felix, Tomcat, Eclipse) and the DaCapo benchmarks. A R virtual machine is currently under heavy development.

J3 has been tested on Linux/x64, Linux/x86, Linux/ppc32, MacOSX/x64, MacOSX/x86, MacOSX/ppc32. The JVM may work on ppc64. Support for Windows has not been investigated.

While this work aims to provide a fully functional JVM, it is still early work and is under heavy development. Some of the common missing pieces in vmkit/llvm are:

- Mixed interpretation/compilation.
- Adaptive optimization.

- ...

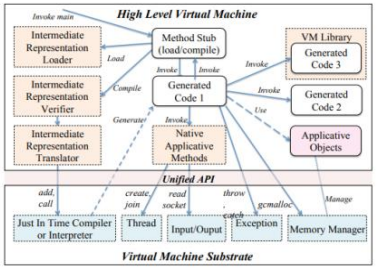
VMKit : a Substrate for Virtual Machines

Nicolas Geoffray, Gaël Thomas, Charles Clément, Bertil Folliot and Gilles Muller

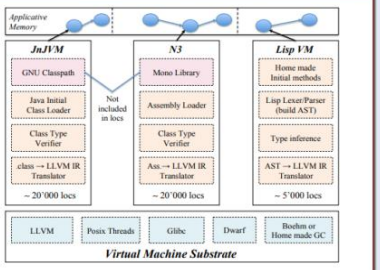
Virtual Machines optimizations require a huge amount of time. Although they share some common principles, such as a Just In Time Compiler or a Garbage Collector, this opportunity for sharing has not been exploited in current VMs.

VMKit is a first attempt to build a common substrate that eases the development of high-level VMs by reusing existing projects: LLVM, GNU Classpath, Mono Library, Boehm GC, GCC, Posix Threads.

Design of VMKit



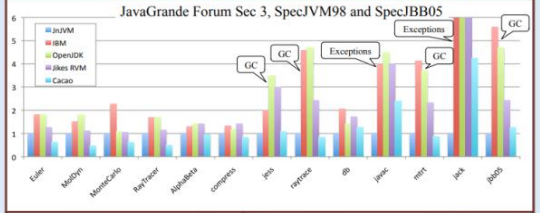
Implementation of three VMs



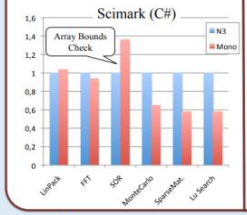
VMKit has been successfully used to build three VMs: a Java Virtual Machine (JnJVM), a Common Language Runtime (N3) and a Lisp-like runtime with type inference. Our high level VMs are only 20,000 lines of code, it took one of the author a month to develop a Common Language Runtime and implementing new ideas in the VMs was remarkably easy [IJVM - DSN09].

Performance (Linux 2.6/1.5GHz/512Mo)

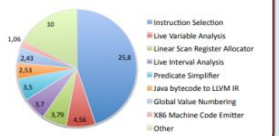
JavaGrande Forum Sec 3, SpecJVM98 and SpecJBB05



Scimark (C#)



Compilation overhead (Tomcat 6.0.16) : OpenJDK 8s, JnJVM 75s



Results

+ LLVM excellent for CPU intensive Java and .Net applications
+ Fast VM development
+ Ahead of Time Compiler

− Ongoing work for a baseline JIT
− Ongoing work to use MMTk
− Ongoing work to optimize exceptions

Homepage

http://vmkit.llvm.org

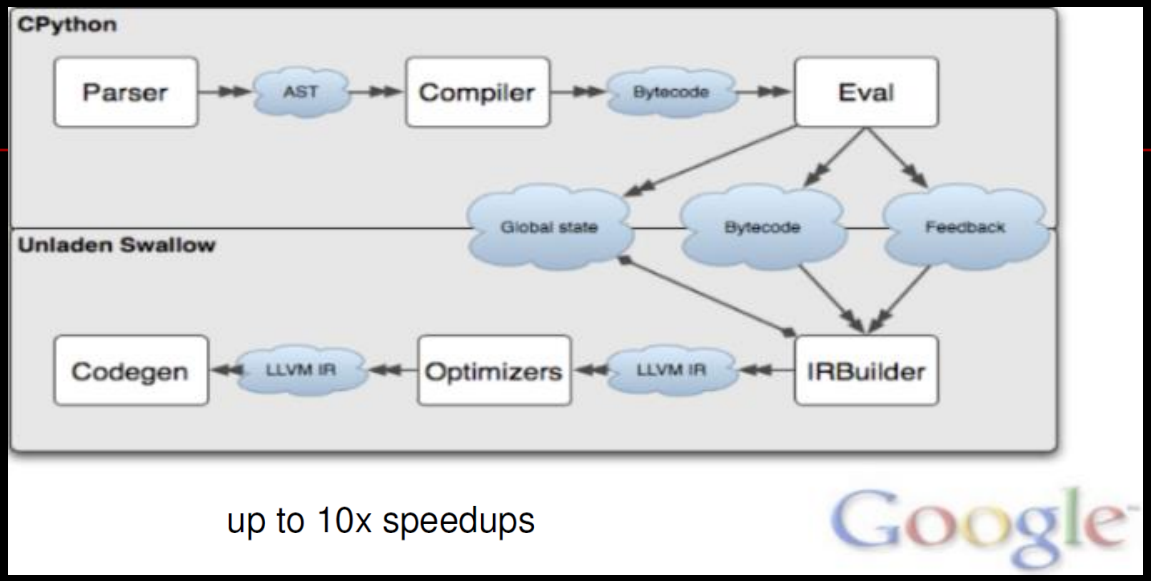
Contact

nicolas.geoffray@lip6.fr
gael.thomas@lip6.fr

LIP6 — UPMC PARIS UNIVERSITAS — INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE — INRIA centre de recherche PARIS - ROCQUENCOURT

Source: http://www-public.it-sudparis.eu/~thomas_g/research/biblio/2009/geoffray09poster-asplos-vmkit.pdf

Unladen Swallow

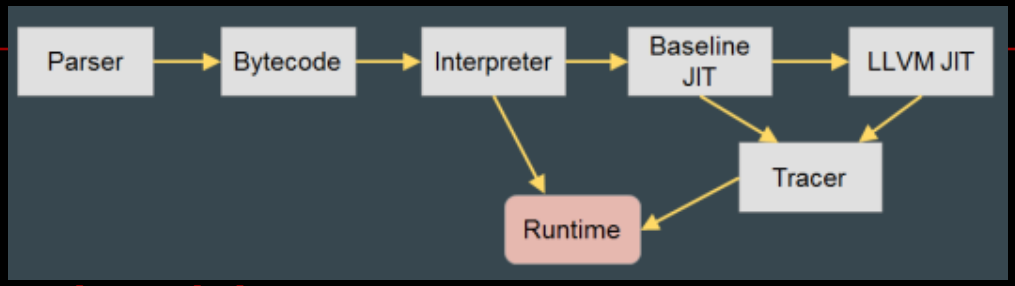
- https://code.google.com/p/unladen-swallow/
- 

- There was a branch for CPython…

2) Pyston

- https://www.pyston.org/

Pyston is a fork of CPython 3.8.12 with additional optimizations for performance. It is targeted at large real-world applications such as web serving, delivering up to a 30% speedup with no development work required.



- Submodules

```
[mydev@fedora pyston-main]$ find . -name "*.git"
./.git
./pyston/LuaJIT/.git
./pyston/bolt/.git
./pyston/llvm/.git
./pyston/macrobenchmarks/.git
./pyston/test/external/django/.git
./pyston/test/external/numpy/.git
./pyston/test/external/numpy/doc/source/_static/scipy-mathjax/.git
./pyston/test/external/pandas/.git
./pyston/test/external/requests/.git
./pyston/test/external/setuptools/.git
./pyston/test/external/six/.git
./pyston/test/external/sqlalchemy/.git
./pyston/test/external/urllib3/.git
./pyston/tools/FlameGraph/.git
[mydev@fedora pyston-main]$
```

- https://blog.pyston.org/
  https://blog.pyston.org/2021/08/30/pyston-team-joins-anaconda/
- …

Techniques

We plan on explaining our techniques in more detail in future blog posts, but the main ones we use are:

- A very-low-overhead JIT using DynASM
- Quickening
- Aggressive attribute caching
- General CPython optimizations
- Build process improvements

- **DynASM**
  https://luajit.org/dynasm.html
  https://github.com/LuaJIT/LuaJIT
  https://luajit.org/dynasm_features.html

DynASM Toolchain Features

- DynASM is a pre-processing assembler.
- DynASM converts mixed C/Assembler source to plain C code.
- The primary knowledge about instruction names, operand modes, registers, opcodes and how to encode them is *only* needed in the pre-processor.
- The generated C code is extremely small and fast.
- A tiny embeddable C library helps with the process of dynamically assembling, relocating and linking machine code.
- There are no outside dependencies on other tools (such as stand-alone assemblers or linkers).
- Internal consistency checks catch runtime errors (e.g. undefined labels).
- The toolchain is split into a portable subset and CPU-specific modules.
- DynASM itself (the pre-processor) is written in Lua.
- There is no machine-dependency for the pre-processor itself. It should work everywhere you can get Lua 5.1 and Lua BitOp up and running (i.e. Linux, *BSD, Windows, ... you name it).

DynASM Assembler Features

- C code and assembler code can be freely mixed. *Readable*, too.
- All the usual syntax for instructions and operand modes you come to expect from a standard assembler.
- Access to C variables and CPP defines in assembler statements.
- Access to C structures and unions via type mapping.
- Convenient shortcuts for accessing C structures.
- Local and global labels.
- Numbered labels (e.g. for mapping bytecode instruction numbers).
- Multiple code sections (e.g. for tailcode).
- Defines/substitutions (inline and from command line).
- Conditionals (translation time) with proper nesting.
- Macros with parameters.
- Macros can mix assembler statements and C code.
- Captures (output diversion for code reordering).
- Simple and extensible template system for instruction definitions.

Restrictions

Currently the x86, x64, ARM, ARM64, PowerPC and MIPS instruction sets are supported. This includes most user-mode instructions available on modern CPUs. For x86/x64 this includes SSE, SSE2, SSE3, SSSE3, SSE4a, SSE4.1, SSE4.2, AVX, AVX2, BMI, ADX, AES-NI and FMA3. For PPC this also includes the e500 instruction set extension.

The whole toolchain has been designed to support multiple CPU architectures. As LuaJIT gets support for more architectures, DynASM will be extended with new CPU-specific modules.

Note that runtime conditionals are not really needed, since you can just use plain C code for that (and LuaJIT does this *a lot*). It's not going to be more (time-) efficient if conditionals are done by the embedded C library (maybe a bit more space-efficient).

DynASM Dependencies

Please don't be shied away because DynASM itself is written in Lua. This *only* applies to the pre-processor. This is pure text-processing and writing such stuff in C would be a waste of time. Pre-processing is done only *once* while your code generator itself is compiled. There are no dependencies on Lua during runtime, i.e. when your code generator is in action.

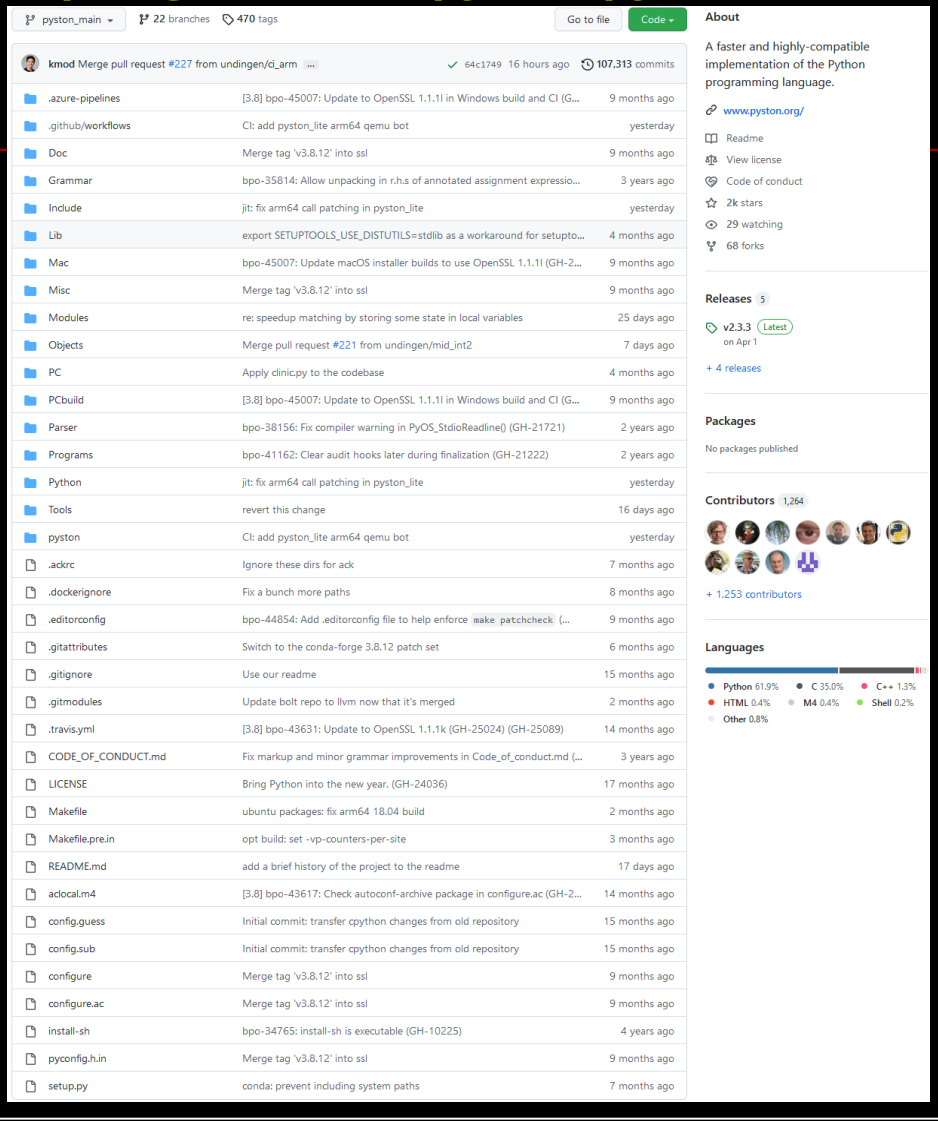
Apart from that, a full Lua distribution is around 200K and can be compiled in a few seconds. Consider it a part of the toolchain, if you want.

Or bundle `src/host/minilua.c` from the LuaJIT source tree, which is a minified Lua 5.1 + BitOp in a single file (45K compressed).

...

Project

- https://github.com/pyston/pyston



pyston_main · 22 branches · 470 tags · Go to file · Code

kmod Merge pull request #227 from undingen/ci_arm ... 64c1749 16 hours ago 107,313 commits

.azure-pipelines	[3.8] bpo-45007: Update to OpenSSL 1.1.1l in Windows build and CI (G...	9 months ago
.github/workflows	CI: add pyston_lite arm64 qemu bot	yesterday
Doc	Merge tag 'v3.8.12' into ssl	9 months ago
Grammar	bpo-35814: Allow unpacking in r.h.s of annotated assignment expressio...	3 years ago
Include	jit: fix arm64 call patching in pyston_lite	yesterday
Lib	export SETUPTOOLS_USE_DISTUTILS=stdlib as a workaround for setupto...	4 months ago
Mac	bpo-45007: Update macOS installer builds to use OpenSSL 1.1.1l (GH-2...	9 months ago
Misc	Merge tag 'v3.8.12' into ssl	9 months ago
Modules	re: speedup matching by storing some state in local variables	25 days ago
Objects	Merge pull request #221 from undingen/mid_int2	7 days ago
PC	Apply clinic.py to the codebase	4 months ago
PCbuild	[3.8] bpo-45007: Update to OpenSSL 1.1.1l in Windows build and CI (G...	9 months ago
Parser	bpo-38156: Fix compiler warning in PyOS_StdioReadline() (GH-21721)	2 years ago
Programs	bpo-41162: Clear audit hooks later during finalization (GH-21222)	2 years ago
Python	jit: fix arm64 call patching in pyston_lite	yesterday
Tools	revert this change	16 days ago
pyston	CI: add pyston_lite arm64 qemu bot	yesterday
.ackrc	Ignore these dirs for ack	7 months ago
.dockerignore	Fix a bunch more paths	8 months ago
.editorconfig	bpo-44854: Add .editorconfig file to help enforce `make patchcheck` (...	9 months ago
.gitattributes	Switch to the conda-forge 3.8.12 patch set	6 months ago
.gitignore	Use our readme	15 months ago
.gitmodules	Update bolt repo to llvm now that it's merged	2 months ago
.travis.yml	[3.8] bpo-43631: Update to OpenSSL 1.1.1k (GH-25024) (GH-25089)	14 months ago
CODE_OF_CONDUCT.md	Fix markup and minor grammar improvements in Code_of_conduct.md (...	3 years ago
LICENSE	Bring Python into the new year. (GH-24036)	17 months ago
Makefile	ubuntu packages: fix arm64 18.04 build	2 months ago
Makefile.pre.in	opt build: set -vp-counters-per-site	3 months ago
README.md	add a brief history of the project to the readme	17 days ago
aclocal.m4	[3.8] bpo-43617: Check autoconf-archive package in configure.ac (GH-2...	14 months ago
config.guess	Initial commit: transfer cpython changes from old repository	15 months ago
config.sub	Initial commit: transfer cpython changes from old repository	15 months ago
configure	Merge tag 'v3.8.12' into ssl	9 months ago
configure.ac	Merge tag 'v3.8.12' into ssl	9 months ago
install-sh	bpo-34765: install-sh is executable (GH-10225)	4 years ago
pyconfig.h.in	Merge tag 'v3.8.12' into ssl	9 months ago
setup.py	conda: prevent including system paths	7 months ago

About

A faster and highly-compatible implementation of the Python programming language.

www.pyston.org/

Readme
View license
Code of conduct
2k stars
29 watching
68 forks

Releases 5

v2.3.3 Latest
on Apr 1

+ 4 releases

Packages

No packages published

Contributors 1,264

+ 1,253 contributors

Languages

Python 61.9% · C 35.0% · C++ 1.3% · HTML 0.4% · M4 0.4% · Shell 0.2% · Other 0.8%

Stats (include submodules)

```
[mydev@fedora pyston-main]$ git branch
* pyston_main
[mydev@fedora pyston-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Alex                    4           12            4            0            8
 Assembly              101        16765        12390         2850         1525
 GNU Style Assembly  16797      3622610      1830410      1038539       753661
 Autoconf              765        64726        23429        28779        12518
 Automake                4          803          732            6           65
 BASH                   22         3362         2469          567          326
 Batch                 114         6172         5179          140          853
 C                   16623      4063036      1991457      1672612       398967
 C Header            20061      3895343      2481967       856063       557313
 CMake                3444       173364       139202        14313        19849
 C#                     16         1532         1140          214          178
 C Shell                 1           37           21            7            9
 C++                 50063     14929229      9312211      4025486      1591532
 C++ Header            110        88566        82946         2627         2993
 CSS                   104        15086        12260          637         2189
 D                       9          183          160            1           22
 Dockerfile             20         1003          508          343          152
 .NET Resource           4          538          264          274            0
 Emacs Lisp             21         2735         1882          541          312
 Elixir                  4          149           46           68           35
 Fish                    1           75           47           15           13
 FORTRAN Legacy        124         1884         1308          546           30
 FORTRAN Modern       1734       121979        84003        25536        12440
 GDB Script              4          532           95          348           89
 Go                     44         8844         6686         1242          916
 HEX                     2           30           30            0            0
 HLSL                    1          110           58           36           16
 INI                    32          982          659          149          174
 JavaScript            328        83037        57585        16563         8889
 JSON                  293       838104       838030            0           74
 Julia                   2         2670         1532          902          236
 LLVM                59572     13288011      4957845      7269857      1060309
 LD Script               2            8            8            0            0
 Lisp                    1          692          502           81          109
 Lua                    37        16373        13461         1611         1301
 Makefile             1433        11394         7907          717         2770
 Module-Definition     315        65330        58695          254         6381
 MSBuild                26         2232         1828          275          129
 OCaml                  94        19284        10870         5304         3110
 Objective-C          3662       239853       135020        64993        39840
 Objective-C++        1043        79736        52346        13500        13890
 Pan                    10          352          333            1           18
 Perl                   64        20925        13773         5140         2012
 PHP                     1          197          159           25           13
 PowerShell              7          612          354          194           64
 Prolog                  1           24           24            0            0
 Protocol Buffers       12         1018          706          144          168
 Python              11730      3101678      2461125       199476       441077
 ReStructuredText     4473      1126957       836609            0       290348
 Rust                    6           60           26           22           12
 Scala                  28          296          296            0            0
 Scheme                  1          172          135           24           13
 Shell                 281        49511        36264         7833         5414
 SVG                   177       109804       109667           94           43
 Swift                   4           46           34            0           12
 SWIG                  150        27268        20526         1602         5140
 TeX                     9         4980         4523          101          356
 Plain Text           3204      1431144            0      1089698       341446
 Thrift                  1           23           17            1            5
 TOML                   26          477          405           20           52
 TypeScript             12          713          520          116           77
 VBScript                1            1            0            1            0
 Vim script             32         1490         1305          106           79
 Visual Studio Pro|     48         6265         6211            9           45
 Visual Studio Sol|      6         1566         1561            0            5
 XSL                     3           42           37            0            5
 XML                   174         2726         2589           24          113
 YAML                 1782       219430       164241        45423         9766
-------------------------------------------------------------------------------
 HTML                  642       113009       106034         1002         5973
 |- CSS                66         1430         1126           14          290
 |- JavaScript         40        15006        13355          550         1101
 (Total)                      129445       120515         1566         7364
-------------------------------------------------------------------------------
 Jupyter Notebooks       1            0            0            0            0
 |- Markdown             1          400            1          281          118
 |- Python               1          376          344           22           10
 (Total)                             776          345          303          128
-------------------------------------------------------------------------------
 Markdown              370        92785            0        73667        19118
 |- BASH                20          433          383           41            9
 |- C                    8          138          111           20            7
 |- CMake                6           70           62            0            8
 |- C++                 71         7327         4884         1788          655
 |- JavaScript           2           70           70            0            0
 |- JSON                 3          105          105            0            0
 |- LLVM                 5          113          105            0            8
 |- Python              10          466          349           58           59
 |- Shell               25          475          379           47           49
 (Total)                          101982         6448        75621        19913
===============================================================================
 Total              200293     48006391     25915940     16473540      5616911
===============================================================================
[mydev@fedora pyston-main]$
[mydev@fedora pyston-main]$ du -sh
8.6G    .
[mydev@fedora pyston-main]$
```

V. DotNet-based implementation

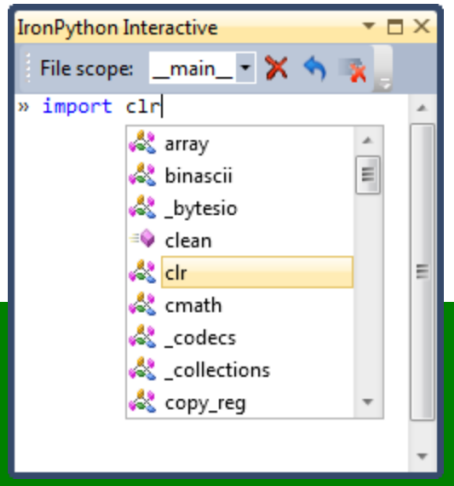
1) Overview

- …

2) IronPython

- https://ironpython.net/
-

IronPython is an open-source implementation of the Python programming language which is tightly integrated with .NET. IronPython can use .NET and Python libraries, and other .NET languages can use Python code just as easily.



Download IronPython 2.7

2.7.11 released on 2020-11-17
release notes | source

Download IronPython 3.4

3.4.0-alpha1 released on 2021-04-19
release notes | source

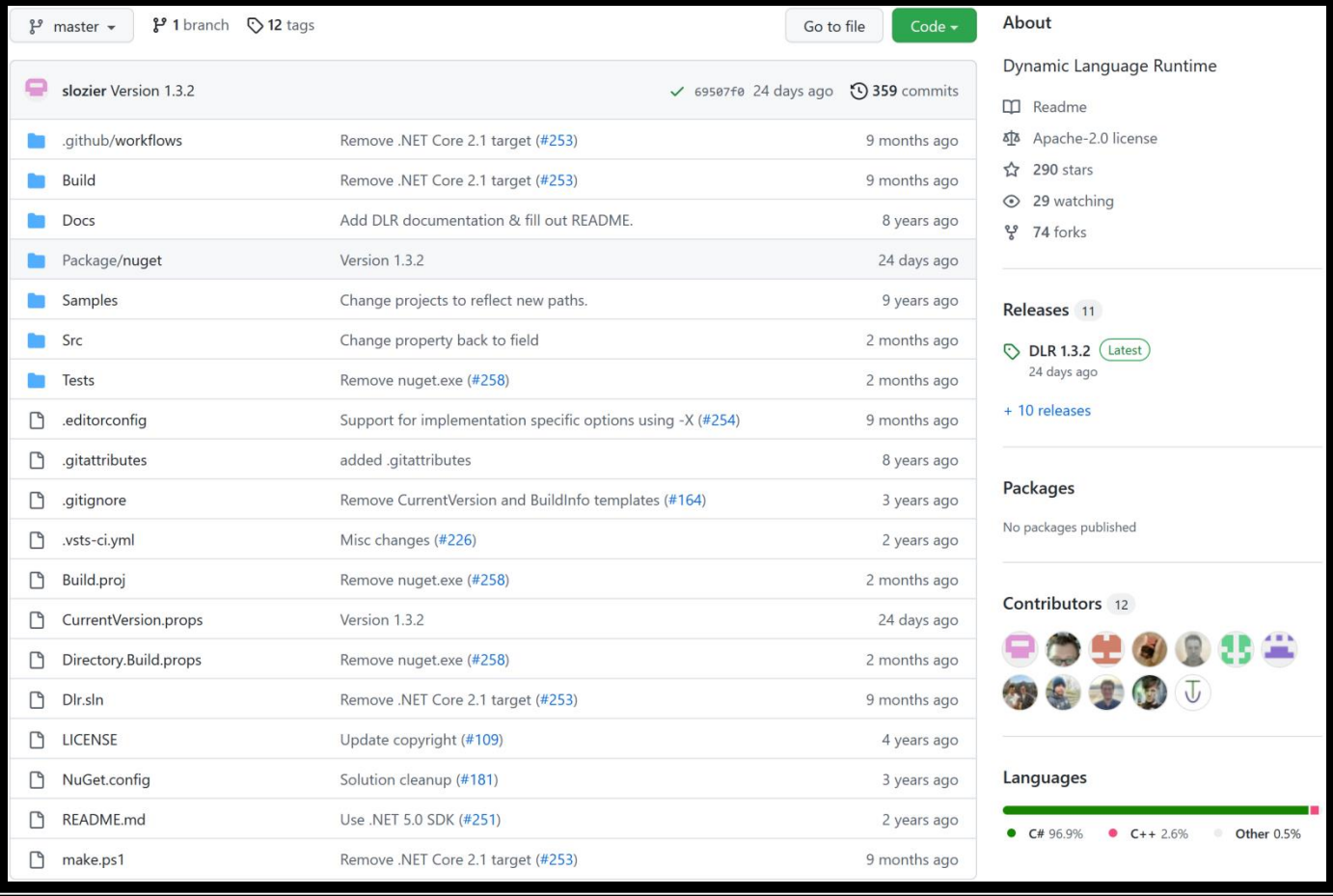
Experience a more interactive .NET and Python development experience with Python Tools for Visual Studio.

- ...

DLR (Dynamic Language Runtime)

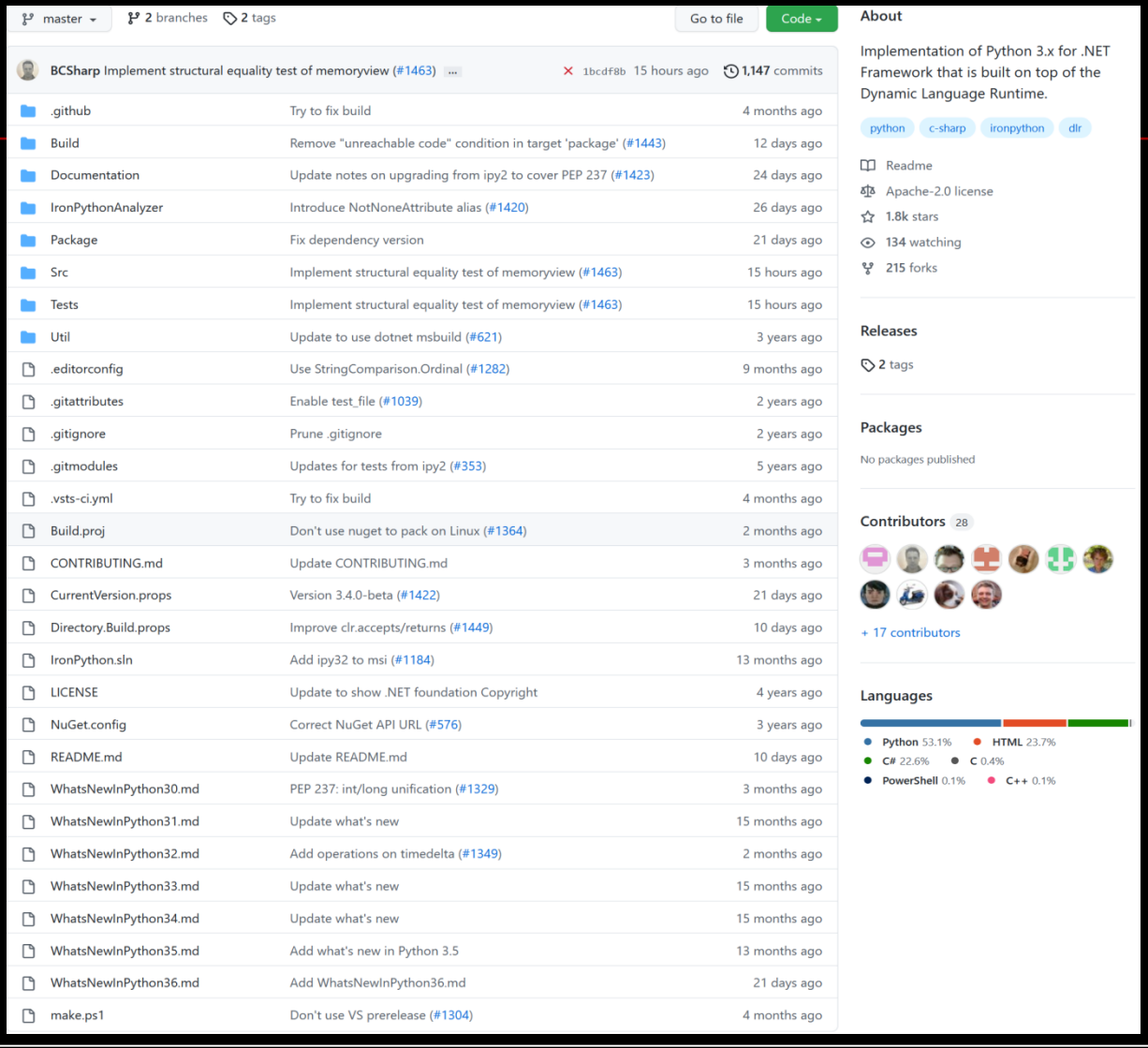
- https://github.com/IronLanguages/dlr

The Dynamic Language Runtime enables language developers to more easily create dynamic languages for the .NET platform. In addition to being a pluggable back-end for dynamic language compilers, the DLR provides language interop for dynamic operations on objects. The DLR has common hosting APIs for using dynamic languages as libraries or for scripting in your .NET applications.



Project

- https://github.com/IronLanguages/ironpython3



master · 2 branches · 2 tags · Go to file · Code

BCSharp Implement structural equality test of memoryview (#1463) ... 1bcdf8b 15 hours ago 1,147 commits

.github	Try to fix build	4 months ago
Build	Remove "unreachable code" condition in target 'package' (#1443)	12 days ago
Documentation	Update notes on upgrading from ipy2 to cover PEP 237 (#1423)	24 days ago
IronPythonAnalyzer	Introduce NotNoneAttribute alias (#1420)	26 days ago
Package	Fix dependency version	21 days ago
Src	Implement structural equality test of memoryview (#1463)	15 hours ago
Tests	Implement structural equality test of memoryview (#1463)	15 hours ago
Util	Update to use dotnet msbuild (#621)	3 years ago
.editorconfig	Use StringComparison.Ordinal (#1282)	9 months ago
.gitattributes	Enable test_file (#1039)	2 years ago
.gitignore	Prune .gitignore	2 years ago
.gitmodules	Updates for tests from ipy2 (#353)	5 years ago
.vsts-ci.yml	Try to fix build	4 months ago
Build.proj	Don't use nuget to pack on Linux (#1364)	2 months ago
CONTRIBUTING.md	Update CONTRIBUTING.md	3 months ago
CurrentVersion.props	Version 3.4.0-beta (#1422)	21 days ago
Directory.Build.props	Improve clr.accepts/returns (#1449)	10 days ago
IronPython.sln	Add ipy32 to msi (#1184)	13 months ago
LICENSE	Update to show .NET foundation Copyright	4 years ago
NuGet.config	Correct NuGet API URL (#576)	3 years ago
README.md	Update README.md	10 days ago
WhatsNewInPython30.md	PEP 237: int/long unification (#1329)	3 months ago
WhatsNewInPython31.md	Update what's new	15 months ago
WhatsNewInPython32.md	Add operations on timedelta (#1349)	2 months ago
WhatsNewInPython33.md	Update what's new	15 months ago
WhatsNewInPython34.md	Update what's new	15 months ago
WhatsNewInPython35.md	Add what's new in Python 3.5	13 months ago
WhatsNewInPython36.md	Add WhatsNewInPython36.md	21 days ago
make.ps1	Don't use VS prerelease (#1304)	4 months ago

About

Implementation of Python 3.x for .NET Framework that is built on top of the Dynamic Language Runtime.

python · c-sharp · ironpython · dlr

Readme
Apache-2.0 license
1.8k stars
134 watching
215 forks

Releases

2 tags

Packages

No packages published

Contributors 28

+ 17 contributors

Languages

Python 53.1% · HTML 23.7% · C# 22.6% · C 0.4% · PowerShell 0.1% · C++ 0.1%

Stats (include submodules)

```
[mydev@fedora ironpython3-master]$ git branch
* master
[mydev@fedora ironpython3-master]$ tokei
```

Language	Files	Lines	Code	Comments	Blanks
Batch	12	303	214	32	57
C	1	662	535	27	100
C Header	87	10243	7649	1051	1543
C#	1287	424381	294609	74775	54997
C Shell	1	37	21	7	9
C++	20	3279	2412	89	778
CSS	1	6	0	6	0
.NET Resource	1	244	185	59	0
Fish	1	74	50	13	11
INI	2	1279	915	2	362
JSON	1	5	5	0	0
Makefile	1	19	15	0	4
Module-Definition	6	503	474	1	28
MSBuild	55	8592	5291	2326	975
PowerShell	10	2661	2020	343	298
Python	1926	738468	592453	51500	94515
ReStructuredText	1	216	165	0	51
Ruby	1	7	6	0	1
Shell	5	47	34	5	8
SVG	1	93	86	1	6
Plain Text	119	54084	0	52450	1634
VBScript	2	14	12	1	1
Visual Basic	3	294	223	28	43
Visual Studio Pro|	3	850	850	0	0
Visual Studio Sol|	4	381	377	0	4
XML	11	15057	15017	2	38
YAML	2	284	226	15	43
HTML	42	255539	227974	6	27559
|- CSS	1	7	7	0	0
|- JavaScript	2	17	16	1	0
(Total)		255563	227997	7	27559
Markdown	18	1947	0	1589	358
|- Python	1	16	12	4	0
(Total)		1963	12	1593	358
Total	3624	1519609	1151853	184333	183423

```
[mydev@fedora ironpython3-master]$
```

3) Pyjion

- https://pyjion.readthedocs.io/en/latest/

  A JIT extension(not a standalone Python implementation) for CPython that compiles your Python code into native CIL and executes it using the .NET CLR.

- Build from source

  …

- https://www.trypyjion.com/
- Originally from https://github.com/microsoft/Pyjion
- https://thautwarm.github.io/Site-32/Design/Research-Restrain-JIT.html
- Support both X64 and AArch64!
- Maybe a good fit for Serverless functions in Python…
- …

Usage

To get started, you need to have .NET installed, with Python 3.10 and the Pyjion package (I also recommend using a virtual environment).

After importing pyjion, enable it by calling `pyjion.enable()` which sets a compilation threshold to 0 (the code only needs to be run once to be compiled by the JIT):

```
>>> import pyjion
>>> pyjion.enable()
```

Any Python code you define or import after enabling pyjion will be JIT compiled. You don't need to execute functions in any special API, its completely transparent:

```
>>> def half(x):
...     return x/2
>>> half(2)
1.0
```

Pyjion will have compiled the `half` function into machine code on-the-fly and stored a cached version of that compiled function inside the function object. You can see some basic stats by running `pyjion.info(f)`, where `f` is the function object:

```
>>> pyjion.info(half)
JitInfo(failed=False, compile_result=<CompilationResult.Success: 1>, compiled=True, optimizations=<Optimizatio
```

You can also execute Pyjion against any script or module:

```
pyjion my_script.py
```

Or, for an existing Python module:

```
pyjion -m calendar
```

Disassembly

You can see the machine code for the compiled function by disassembling it in the Python REPL. Pyjion has essentially compiled your small Python function into a small, standalone application. Install `distorm3` and `rich` first to disassemble x86-64 assembly and run `pyjion.dis.dis_native(f)`:

```
>>> import pyjion.dis
>>> pyjion.dis.dis_native(half)
00000000: PUSH RBP
00000001: MOV RBP, RSP
00000004: PUSH R14
00000006: PUSH RBX
00000007: MOV RBX, RSI
0000000a: MOV R14, [RDI+0x40]
0000000e: CALL 0x1b34
00000013: CMP DWORD [RAX+0x30], 0x0
00000017: JZ 0x31
00000019: CMP QWORD [RAX+0x40], 0x0
0000001e: JZ 0x31
00000020: MOV RDI, RAX
00000023: MOV RSI, RBX
00000026: XOR EDX, EDX
00000028: POP RBX
00000029: POP R14
...
```

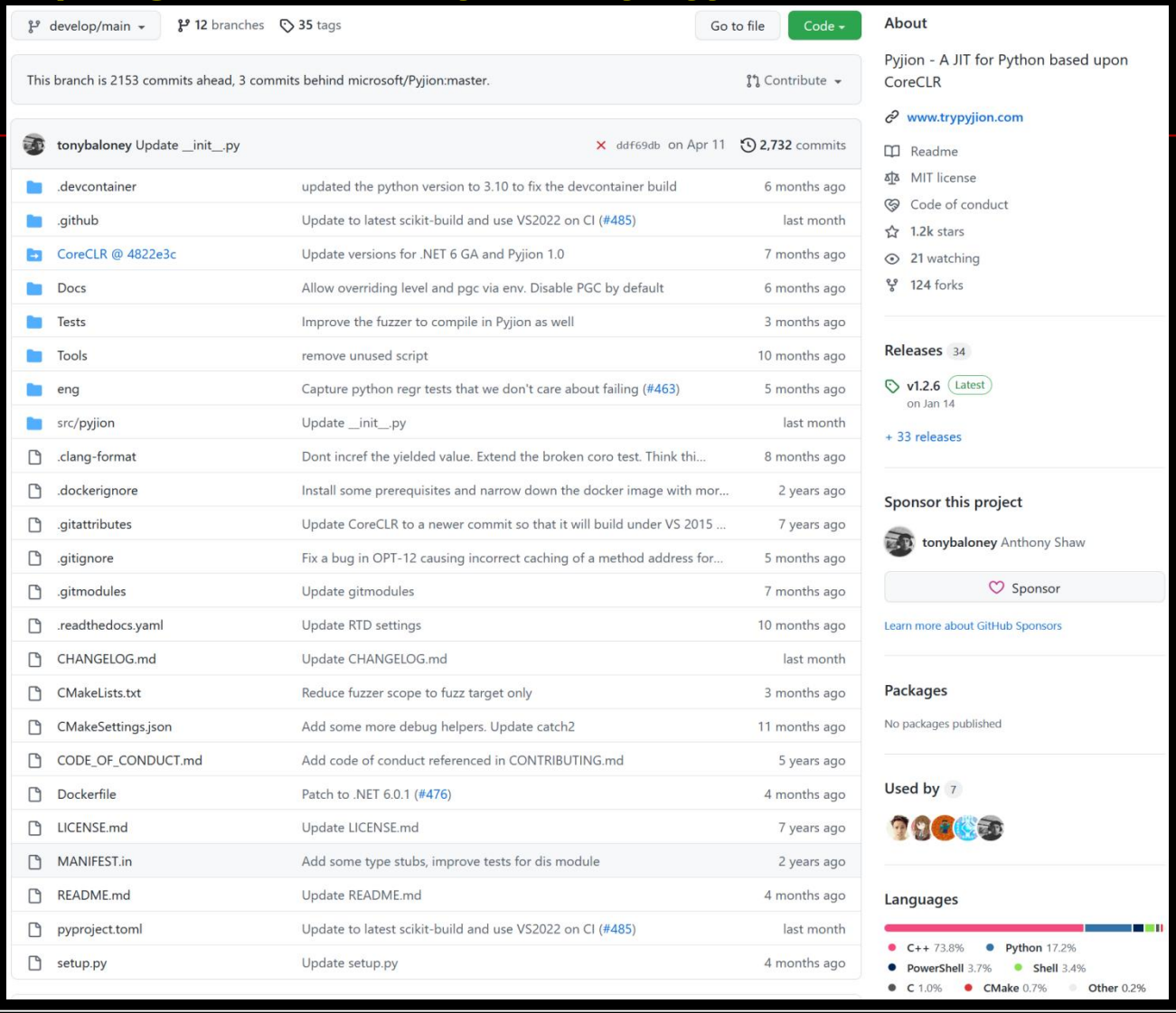
The complex logic of converting a portable instruction set into low-level machine instructions is done by .NET's CLR JIT compiler.

All Python code executed after the JIT is enabled will be compiled into native machine code at runtime and cached on disk. For example, to enable the JIT on a simple `app.py` for a Flask web app:

```
from src import pyjion
pyjion.enable()

from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

app.run()
```

Project

- https://github.com/tonybaloney/Pyjion



develop/main · 12 branches · 35 tags · Go to file · Code

This branch is 2153 commits ahead, 3 commits behind microsoft/Pyjion:master. · Contribute

tonybaloney Update __init__.py — ddf69db on Apr 11 — 2,732 commits

.devcontainer	updated the python version to 3.10 to fix the devcontainer build	6 months ago
.github	Update to latest scikit-build and use VS2022 on CI (#485)	last month
CoreCLR @ 4822e3c	Update versions for .NET 6 GA and Pyjion 1.0	7 months ago
Docs	Allow overriding level and pgc via env. Disable PGC by default	6 months ago
Tests	Improve the fuzzer to compile in Pyjion as well	3 months ago
Tools	remove unused script	10 months ago
eng	Capture python regr tests that we don't care about failing (#463)	5 months ago
src/pyjion	Update __init__.py	last month
.clang-format	Dont incref the yielded value. Extend the broken coro test. Think thi...	8 months ago
.dockerignore	Install some prerequisites and narrow down the docker image with mor...	2 years ago
.gitattributes	Update CoreCLR to a newer commit so that it will build under VS 2015 ...	7 years ago
.gitignore	Fix a bug in OPT-12 causing incorrect caching of a method address for...	5 months ago
.gitmodules	Update gitmodules	7 months ago
.readthedocs.yaml	Update RTD settings	10 months ago
CHANGELOG.md	Update CHANGELOG.md	last month
CMakeLists.txt	Reduce fuzzer scope to fuzz target only	3 months ago
CMakeSettings.json	Add some more debug helpers. Update catch2	11 months ago
CODE_OF_CONDUCT.md	Add code of conduct referenced in CONTRIBUTING.md	5 years ago
Dockerfile	Patch to .NET 6.0.1 (#476)	4 months ago
LICENSE.md	Update LICENSE.md	7 years ago
MANIFEST.in	Add some type stubs, improve tests for dis module	2 years ago
README.md	Update README.md	4 months ago
pyproject.toml	Update to latest scikit-build and use VS2022 on CI (#485)	last month
setup.py	Update setup.py	4 months ago

About

Pyjion - A JIT for Python based upon CoreCLR

www.trypyjion.com

Readme · MIT license · Code of conduct · 1.2k stars · 21 watching · 124 forks

Releases 34

v1.2.6 Latest on Jan 14

+ 33 releases

Sponsor this project

tonybaloney Anthony Shaw

Sponsor

Learn more about GitHub Sponsors

Packages

No packages published

Used by 7

Languages

C++ 73.8% · Python 17.2% · PowerShell 3.7% · Shell 3.4% · C 1.0% · CMake 0.7% · Other 0.2%

Stats (include submodules)

```
[mydev@fedora pyjion-develop-main]$ git branch
* develop/main
[mydev@fedora pyjion-develop-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Assembly               46        12992         6364         3728         2900
 GNU Style Assembly    108        13616         7612         3576         2428
 Autoconf               39         4010         1940         1396          674
 Automake                4         1237         1000           87          150
 BASH                    2          111           75           15           21
 Batch                  51         4251         3106          443          702
 C                    1218       591031       438881        74262        77888
 C Header             2126       679983       453331       111211       115441
 CMake                 297        23255        18383         1797         3075
 C#                  28375      8258942      6475679       666658      1116605
 C++                  2331      1481951      1030174       235325       216452
 C++ Header            230        97810        69662        12835        15313
 CSS                     2          132          108            1           23
 Dockerfile             12          313          222           25           66
 .NET Resource         213        70153        63468         6680            5
 F#                     10         1521         1196           94          231
 Happy                   1         2063         1847            0          216
 INI                     1            6            6            0            0
 Java                    2          204          168            7           29
 JavaScript             14         4788         3572          524          692
 JSON                   64        12821        12811            0           10
 Makefile               16          812          563           80          169
 Module-Definition      31         3682         3343           16          323
 MSBuild              8243       162589       156372         3350         2867
 Objective-C            11          972          746           63          163
 Objective-C++           5          219          156           26           37
 Perl                   10         1888         1475          204          209
 PowerShell             59         8135         6428          742          965
 Prolog                  6         2547         2296            0          251
 Python                119        32262        23853         2740         5669
 ReStructuredText       23         1834         1251            0          583
 Shell                 184        28096        21618         1839         4639
 SVG                     3          800          789            3            8
 TeX                    32         3253         2511            0          742
 Plain Text            445       178202            0       174015         4187
 TOML                    1           24           20            0            4
 Visual Basic           74        50623        36496         5027         9100
 Visual Studio Pro|     11         2398         2391            6            1
 Visual Studio Sol|    252        30703        30680            0           23
 XSL                   640        14345        11693          387         2265
 XML                   845        67465        63380         2318         1767
 YAML                  140        16212        13375         1363         1474
-------------------------------------------------------------------------------
 HTML                   30          637          593           12           32
 |- JavaScript           7          245          214            4           27
 (Total)                            882          807           16           59
-------------------------------------------------------------------------------
 Markdown              376        61885            0        45354        16531
 |- Assembly             1          109           81           28            0
 |- BASH                13          195          148           29           18
 |- C                    3           40           36            0            4
 |- CMake                2           56           46            0           10
 |- C#                  20         1076          851           90          135
 |- C++                 17         1513         1070          262          181
 |- Java                 1            7            7            0            0
 |- JavaScript           1           33           33            0            0
 |- JSON                 5          150          150            0            0
 |- Markdown             1            9            0            5            4
 |- PowerShell           4           17           17            0            0
 |- Python               2           22           19            0            3
 |- Shell                1           15            7            5            3
 |- XML                 21          232          211           12            9
 (Total)                           65359         2676        45785        16898
===============================================================================
 Total                46702     11934492      8972524      1356644      1605324
===============================================================================
[mydev@fedora pyjion-develop-main]$
[mydev@fedora pyjion-develop-main]$ du -sh
1.8G    .
[mydev@fedora pyjion-develop-main]$
```

VI. Wasm-based implementation

1) Overview

- …

2) Pyodide

- **https://pyodide.org**

Pyodide is a port of CPython to WebAssembly/Emscripten.

Pyodide makes it possible to install and run Python packages in the browser with micropip. Any pure Python package with a wheel available on PyPi is supported. Many packages with C extensions have also been ported for use with Pyodide. These include many general-purpose packages such as regex, PyYAML, lxml and scientific Python packages including NumPy, pandas, SciPy, Matplotlib, and scikit-learn.

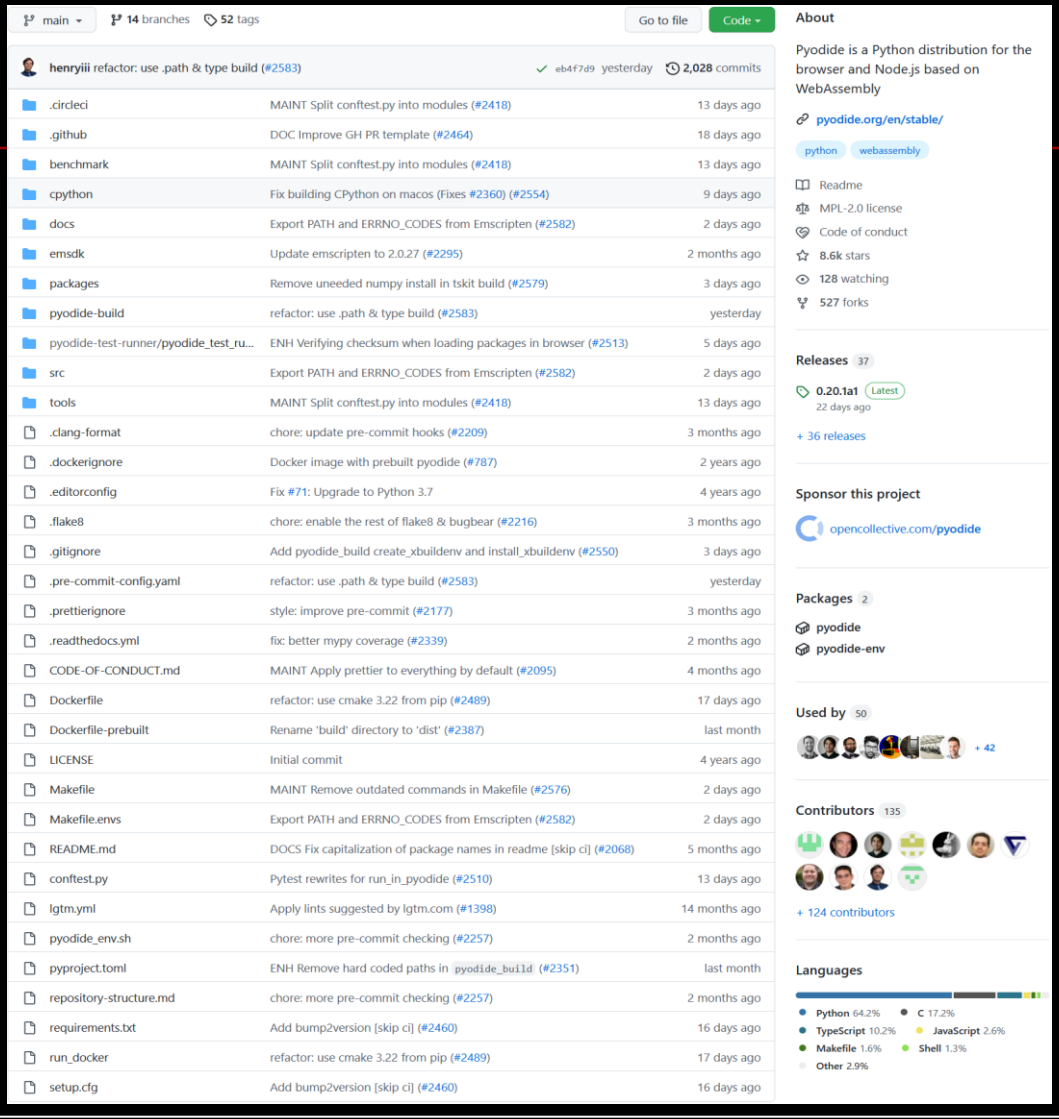
Pyodide comes with a robust Javascript ⟺ Python foreign function interface so that you can freely mix these two languages in your code with minimal friction. This includes full support for error handling (throw an error in one language, catch it in the other), async/await, and much more.

When used inside a browser, Python has full access to the Web APIs.

- **Should not support WASI yet.**
- **Officially only support X64, but it seems that we could add support for ARM as Emscripten SDK should be available on ARM soon.**
- **…**

Project

- https://github.com/pyodide/pyodide



main · 14 branches · 52 tags · Go to file · Code

henryiii refactor: use .path & type build (#2583) — eb4f7d9 yesterday — 2,028 commits

.circleci	MAINT Split conftest.py into modules (#2418)	13 days ago
.github	DOC Improve GH PR template (#2464)	18 days ago
benchmark	MAINT Split conftest.py into modules (#2418)	13 days ago
cpython	Fix building CPython on macos (Fixes #2360) (#2554)	9 days ago
docs	Export PATH and ERRNO_CODES from Emscripten (#2582)	2 days ago
emsdk	Update emscripten to 2.0.27 (#2295)	2 months ago
packages	Remove uneeded numpy install in tskit build (#2579)	3 days ago
pyodide-build	refactor: use .path & type build (#2583)	yesterday
pyodide-test-runner/pyodide_test_ru...	ENH Verifying checksum when loading packages in browser (#2513)	5 days ago
src	Export PATH and ERRNO_CODES from Emscripten (#2582)	2 days ago
tools	MAINT Split conftest.py into modules (#2418)	13 days ago
.clang-format	chore: update pre-commit hooks (#2209)	3 months ago
.dockerignore	Docker image with prebuilt pyodide (#787)	2 years ago
.editorconfig	Fix #71: Upgrade to Python 3.7	4 years ago
.flake8	chore: enable the rest of flake8 & bugbear (#2216)	3 months ago
.gitignore	Add pyodide_build create_xbuildenv and install_xbuildenv (#2550)	3 days ago
.pre-commit-config.yaml	refactor: use .path & type build (#2583)	yesterday
.prettierignore	style: improve pre-commit (#2177)	3 months ago
.readthedocs.yml	fix: better mypy coverage (#2339)	2 months ago
CODE-OF-CONDUCT.md	MAINT Apply prettier to everything by default (#2095)	4 months ago
Dockerfile	refactor: use cmake 3.22 from pip (#2489)	17 days ago
Dockerfile-prebuilt	Rename 'build' directory to 'dist' (#2387)	last month
LICENSE	Initial commit	4 years ago
Makefile	MAINT Remove outdated commands in Makefile (#2576)	2 days ago
Makefile.envs	Export PATH and ERRNO_CODES from Emscripten (#2582)	2 days ago
README.md	DOCS Fix capitalization of package names in readme [skip ci] (#2068)	5 months ago
conftest.py	Pytest rewrites for run_in_pyodide (#2510)	13 days ago
lgtm.yml	Apply lints suggested by lgtm.com (#1398)	14 months ago
pyodide_env.sh	chore: more pre-commit checking (#2257)	2 months ago
pyproject.toml	ENH Remove hard coded paths in `pyodide_build` (#2351)	last month
repository-structure.md	chore: more pre-commit checking (#2257)	2 months ago
requirements.txt	Add bump2version [skip ci] (#2460)	16 days ago
run_docker	refactor: use cmake 3.22 from pip (#2489)	17 days ago
setup.cfg	Add bump2version [skip ci] (#2460)	16 days ago

About

Pyodide is a Python distribution for the browser and Node.js based on WebAssembly

pyodide.org/en/stable/

python · webassembly

Readme · MPL-2.0 license · Code of conduct · 8.6k stars · 128 watching · 527 forks

Releases 37

0.20.1a1 Latest — 22 days ago

+ 36 releases

Sponsor this project

opencollective.com/pyodide

Packages 2

pyodide · pyodide-env

Used by 50

+ 42

Contributors 135

+ 124 contributors

Languages

Python 64.2% · C 17.2% · TypeScript 10.2% · JavaScript 2.6% · Makefile 1.6% · Shell 1.3% · Other 2.9%

Stats

```
[mydev@fedora pyodide-main]$ git branch
* main
[mydev@fedora pyodide-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    1          160          137            5           18
 Batch                   1           35           26            1            8
 C                      12         5883         4326         1012          545
 C Header               14         1224          527          515          182
 CMake                   3          298          240           21           37
 C++                     3           72           64            0            8
 CSS                     1           23           16            3            4
 Dockerfile              1           98           64           21           13
 JavaScript             12         1021          822          147           52
 JSON                    8        15667        15667            0            0
 Makefile                5          476          327           41          108
 Python                192        25866        21310          926         3630
 ReStructuredText        1           95           69            0           26
 Shell                   4          479          232           66          181
 Plain Text              3           44            0           44            0
 TOML                    3           65           57            1            7
 TypeScript             12         3662         2235         1216          211
 YAML                  139         3257         3041           88          128
-------------------------------------------------------------------------------
 HTML                    3           42           40            2            0
 |- CSS                  1            4            4            0            0
 |- JavaScript           3          184          172            7            5
 (Total)                             230          216            9            5
-------------------------------------------------------------------------------
 Markdown               45         5688            0         4305         1383
 |- BASH                 3           28           28            0            0
 |- C                    3           93           77           13            3
 |- HTML                 2           72           66            1            5
 |- JavaScript           5           54           49            5            0
 |- Python               4           35           29            2            4
 |- YAML                 1           17           14            0            3
 (Total)                            5987          263         4326         1398
===============================================================================
 Total                 463        64642        49639         8442         6561
===============================================================================
[mydev@fedora pyodide-main]$
```

2) Wasmer Python

- https://github.com/wasmerio/wasmer-python

A complete and mature WebAssembly runtime for Python based on Wasmer.

Features

- Secure by default. No file, network, or environment access, unless explicitly enabled.
- Fast. Run WebAssembly at near-native speeds.
- Compliant with latest WebAssembly Proposals (SIMD, Reference Types, Threads, ...)

The `wasmer` package brings the required API to execute WebAssembly modules. In a nutshell, `wasmer` compiles the WebAssembly module into compiled code, and then executes it. `wasmer` is designed to work in various environments and platforms: From nano single-board computers to large and powerful servers, including more exotic ones. To address those requirements, Wasmer provides 2 engines and 3 compilers.

Succinctly, an *engine* is responsible to drive the *compilation* and the *execution* of a WebAssembly module. By extension, a *headless* engine can only execute a WebAssembly module, i.e. a module that has previously been compiled, or compiled, serialized and deserialized. By default, the `wasmer` package comes with 2 headless engines:

1. `wasmer.engine.Universal`, the compiled machine code lives in memory,
2. `wasmer.engine.Native`, the compiled machine code lives in a shared object file (`.so`, `.dylib`, or `.dll`), and is natively executed.

Because `wasmer` does not embed compilers in its package, engines are headless, i.e. they can't compile WebAssembly module; they can only execute them. Compilers live in their own standalone packages. Let's briefly introduce them:

Compiler package	Description	PyPi
`wasmer_compiler_singlepass`	Super fast compilation times, slower execution times. Not prone to JIT-bombs. *Ideal for blockchains*	pypi v1.1.0 downloads 30k
`wasmer_compiler_cranelift`	Fast compilation times, fast execution times. *Ideal for development*	pypi v1.1.0 downloads 3M
`wasmer_compiler_llvm`	Slow compilation times, very fast execution times (close to native, sometimes faster). *Ideal for Production*	pypi v1.1.0 downloads 18k

We generally recommend `wasmer_compiler_cranelift` for development purposes and `wasmer_compiler_llvm` in production.

- https://wasmerio.github.io/wasmer-python/api/wasmer/wasmer.html
- Supported platforms

Platform	Architecture	Triple	Packages	
Linux	amd64	x86_64-unknown-linux-gnu	wasmer	✅
			wasmer_compiler_singlepass	✅
			wasmer_compiler_cranelift	✅
			wasmer_compiler_llvm	✅
	aarch64	aarch64-unknown-linux-gnu	wasmer	✅
			wasmer_compiler_singlepass	❌ [1]
			wasmer_compiler_cranelift	✅
			wasmer_compiler_llvm	✅
Darwin	amd64	x86_64-apple-darwin	wasmer	✅
			wasmer_compiler_singlepass	✅
			wasmer_compiler_cranelift	✅
			wasmer_compiler_llvm	✅
Windows	amd64	x86_64-pc-windows-msvc	wasmer	✅
			wasmer_compiler_singlepass	✅
			wasmer_compiler_cranelift	✅
			wasmer_compiler_llvm	❌ [2]

- [1] `wasmer_compiler_singlepass` does not support `aarch64` for the moment
- [2] `wasmer_compiler_llvm` is not packaging properly on Windows for the moment

Wheels are all built for the following Python versions:

- Python 3.7,
- Python 3.8.
- Python 3.9.
- Python 3.10,

▶ Learn about the “fallback” `py3-none-any` wheel

- ...

Examples

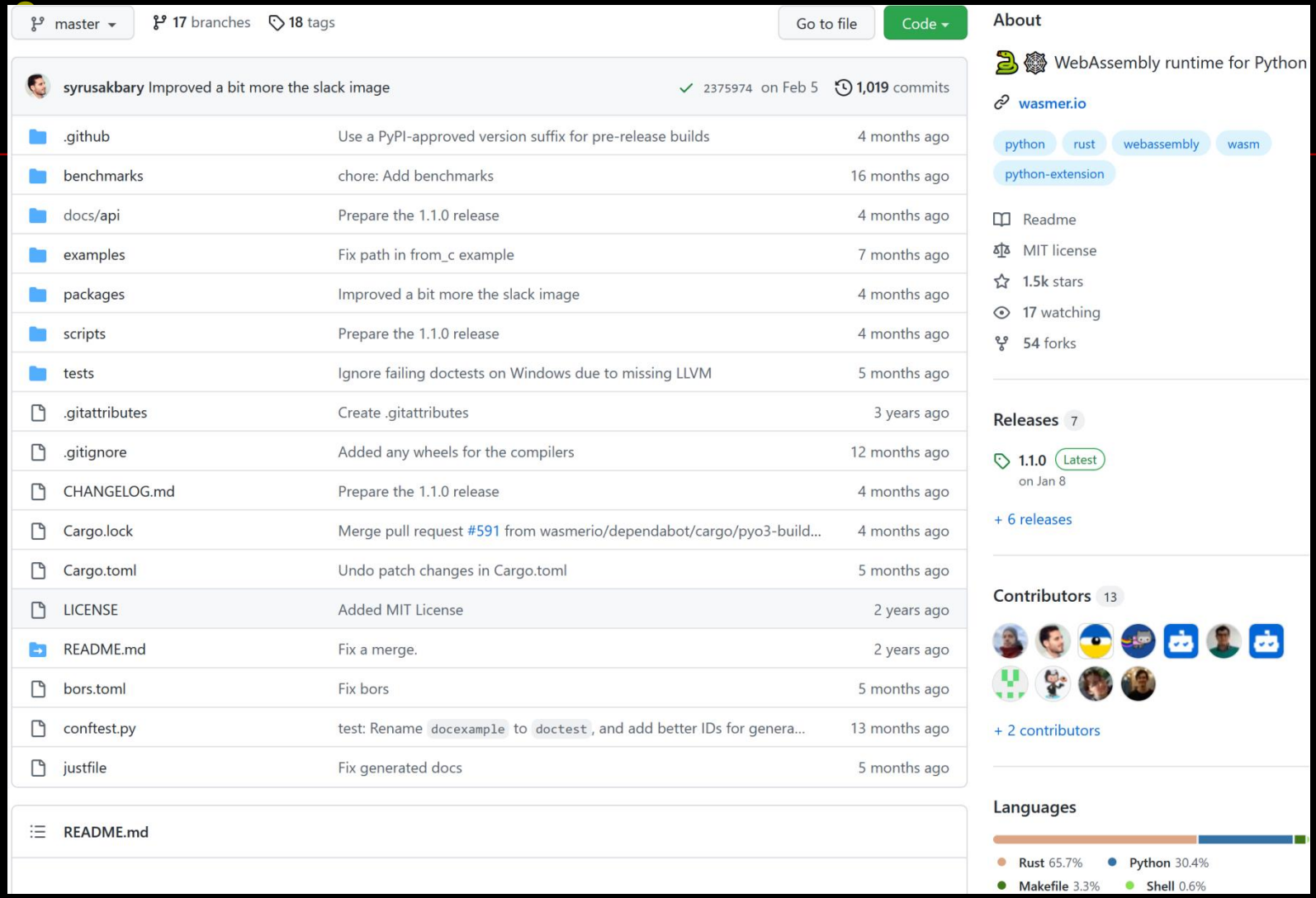
```python
from wasmer import engine, Store, Module, Instance

store = Store()

# Let's compile the module to be able to execute it!
module = Module(store, """
(module
  (type (func (param i32 i32) (result i32)))
  (func (export "sum") (type 0) (param i32) (param i32) (result i32)
    local.get 0
    local.get 1
    i32.add))
""")

# Now the module is compiled, we can instantiate it.
instance = Instance(module)

# Call the exported `sum` function.
result = instance.exports.sum(5, 37)

print(result) # 42!
```

And then, finally, enjoy by running:

```
$ python examples/appendices/simple.py
```

- https://github.com/wasmerio/wasmer-python/tree/master/examples

Project



master · 17 branches · 18 tags · Go to file · Code

syrusakbary Improved a bit more the slack image — 2375974 on Feb 5 — 1,019 commits

.github	Use a PyPI-approved version suffix for pre-release builds	4 months ago
benchmarks	chore: Add benchmarks	16 months ago
docs/api	Prepare the 1.1.0 release	4 months ago
examples	Fix path in from_c example	7 months ago
packages	Improved a bit more the slack image	4 months ago
scripts	Prepare the 1.1.0 release	4 months ago
tests	Ignore failing doctests on Windows due to missing LLVM	5 months ago
.gitattributes	Create .gitattributes	3 years ago
.gitignore	Added any wheels for the compilers	12 months ago
CHANGELOG.md	Prepare the 1.1.0 release	4 months ago
Cargo.lock	Merge pull request #591 from wasmerio/dependabot/cargo/pyo3-build...	4 months ago
Cargo.toml	Undo patch changes in Cargo.toml	5 months ago
LICENSE	Added MIT License	2 years ago
README.md	Fix a merge.	2 years ago
bors.toml	Fix bors	5 months ago
conftest.py	test: Rename `docexample` to `doctest`, and add better IDs for genera...	13 months ago
justfile	Fix generated docs	5 months ago

README.md

About

WebAssembly runtime for Python

wasmer.io

python · rust · webassembly · wasm · python-extension

Readme
MIT license
1.5k stars
17 watching
54 forks

Releases 7

1.1.0 Latest
on Jan 8

+ 6 releases

Contributors 13

+ 2 contributors

Languages

Rust 65.7% · Python 30.4% · Makefile 3.3% · Shell 0.6%

3) PyScript

- https://pyscript.net/

Say Hello to PyScript

PyScript is a framework that allows users to create rich Python applications in the browser using HTML's interface. PyScript aims to give users a first-class programming language that has consistent styling rules, is more expressive, and is easier to learn.

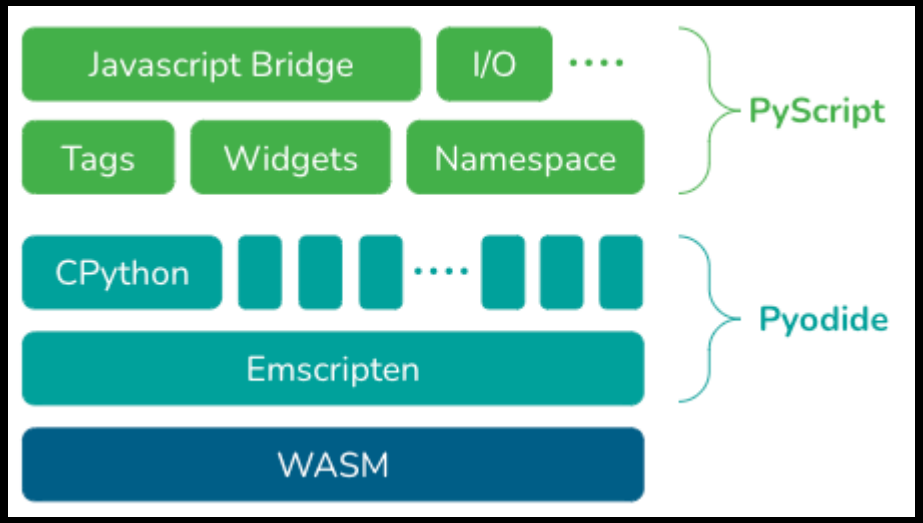
What is PyScript? Well, here are some of the core components:

- **Python in the browser:** Enable drop-in content, external file hosting (made possible by the Pyodide project, thank you!), and application hosting without the reliance on server-side configuration
- **Python ecosystem:** Run many popular packages of Python and the scientific stack (such as numpy, pandas, scikit-learn, and more)
- **Python with JavaScript:** Bi-directional communication between Python and Javascript objects and namespaces
- **Environment management:** Allow users to define what packages and files to include for the page code to run
- **Visual application development:** Use readily available curated UI components, such as buttons, containers, text boxes, and more
- **Flexible framework:** A flexible framework that can be leveraged to create and share new pluggable and extensible components directly in Python

All that to say... PyScript is just HTML, only a bit (okay, maybe a lot) more powerful, thanks to the rich and accessible ecosystem of Python libraries.

In short, our mission is to bring programming for the 99%.

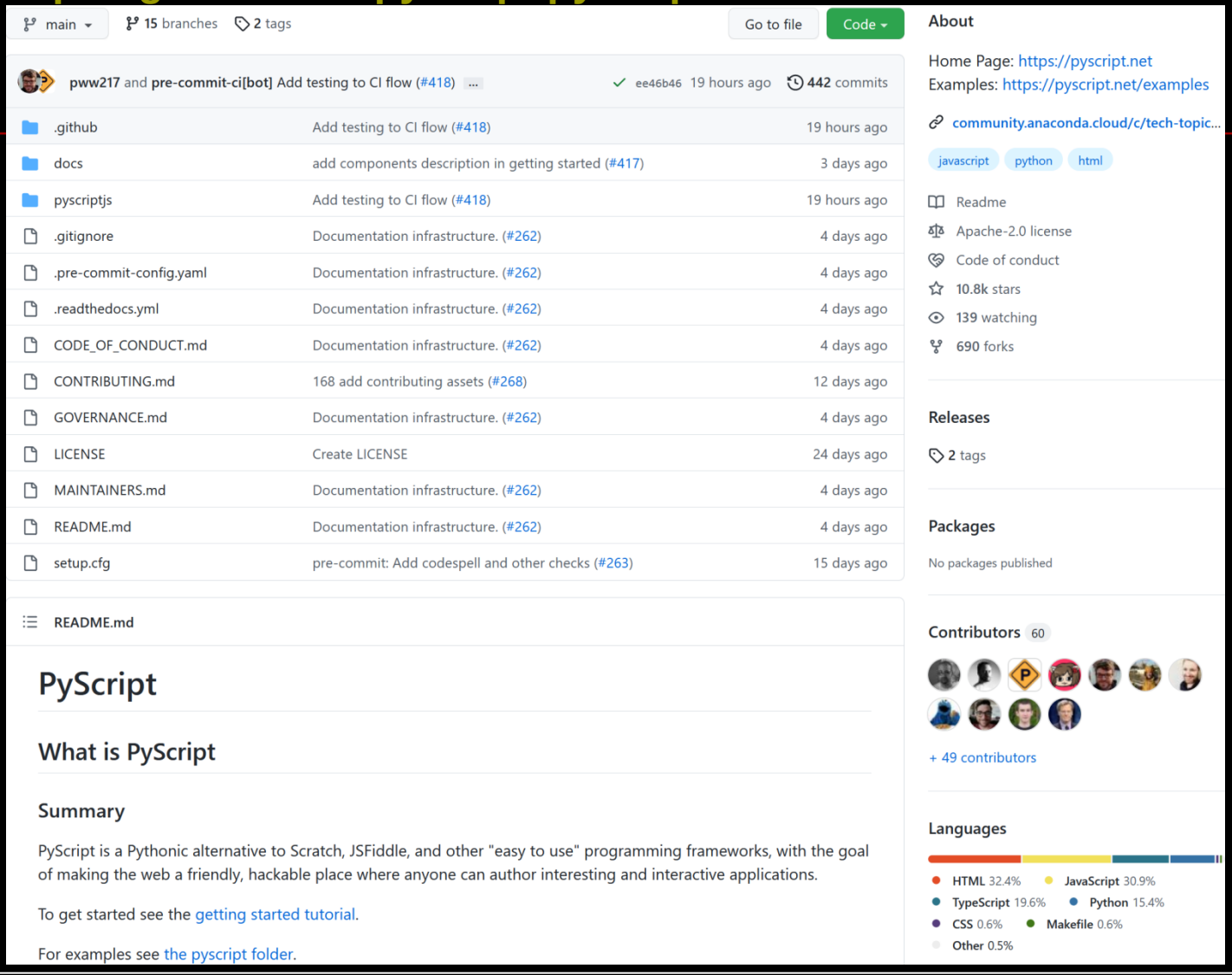
Architecture & Design



Source: https://anaconda.cloud/pyscript-python-in-the-browser

Project

- https://github.com/pyscript/pyscript



main ▾ 15 branches 2 tags Go to file Code ▾

pww217 and pre-commit-ci[bot] Add testing to CI flow (#418) ... ee46b46 19 hours ago 442 commits

.github	Add testing to CI flow (#418)	19 hours ago
docs	add components description in getting started (#417)	3 days ago
pyscriptjs	Add testing to CI flow (#418)	19 hours ago
.gitignore	Documentation infrastructure. (#262)	4 days ago
.pre-commit-config.yaml	Documentation infrastructure. (#262)	4 days ago
.readthedocs.yml	Documentation infrastructure. (#262)	4 days ago
CODE_OF_CONDUCT.md	Documentation infrastructure. (#262)	4 days ago
CONTRIBUTING.md	168 add contributing assets (#268)	12 days ago
GOVERNANCE.md	Documentation infrastructure. (#262)	4 days ago
LICENSE	Create LICENSE	24 days ago
MAINTAINERS.md	Documentation infrastructure. (#262)	4 days ago
README.md	Documentation infrastructure. (#262)	4 days ago
setup.cfg	pre-commit: Add codespell and other checks (#263)	15 days ago

README.md

PyScript

What is PyScript

Summary

PyScript is a Pythonic alternative to Scratch, JSFiddle, and other "easy to use" programming frameworks, with the goal of making the web a friendly, hackable place where anyone can author interesting and interactive applications.

To get started see the getting started tutorial.

For examples see the pyscript folder.

About

Home Page: https://pyscript.net
Examples: https://pyscript.net/examples

community.anaconda.cloud/c/tech-topic...

javascript python html

Readme
Apache-2.0 license
Code of conduct
10.8k stars
139 watching
690 forks

Releases

2 tags

Packages

No packages published

Contributors 60

+ 49 contributors

Languages

HTML 32.4% JavaScript 30.9%
TypeScript 19.6% Python 15.4%
CSS 0.6% Makefile 0.6%
Other 0.5%

Stats

```
[mydev@fedora pyscript-main]$ git branch
* main
[mydev@fedora pyscript-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Batch                   1           35           26            1            8
 CSS                     4          110          101            1            8
 JavaScript             26         2834         2385          117          332
 JSON                    3         6891         6890            0            1
 Makefile                2          117           78            7           32
 Python                 16         1547         1187           96          264
 Shell                   1            7            5            1            1
 SVG                     1           72           72            0            0
 Plain Text              3           59            0           48           11
 TOML                    1           48           39            1            8
 TypeScript             14         1559         1232           97          230
 YAML                    2           32           31            0            1
-------------------------------------------------------------------------------
 HTML                   28         2528         2116            9          403
 |- CSS                  5           52           48            0            4
 |- HTML                 1            6            6            0            0
 |- JavaScript           5           69           60            0            9
 (Total)                            2655         2230            9          416
-------------------------------------------------------------------------------
 Markdown               18          624            0          420          204
 |- HTML                 2          108          102            0            6
 |- Python               1            8            5            1            2
 (Total)                             740          107          421          212
-------------------------------------------------------------------------------
 Svelte                  2            9            7            0            2
 |- CSS                  1           31           29            0            2
 |- JavaScript           1            1            1            0            0
 |- PostCSS              1            3            3            0            0
 (Total)                              44           40            0            4
===============================================================================
 Total                 122        16750        14423          799         1528
===============================================================================
[mydev@fedora pyscript-main]$
```

Good Resources

- https://anaconda.cloud/pyscript-pycon2022-peter-wang-keynote
- https://anaconda.cloud/s/pyscript
- https://engineering.anaconda.com/2022/04/welcome-pyscript.html
- …

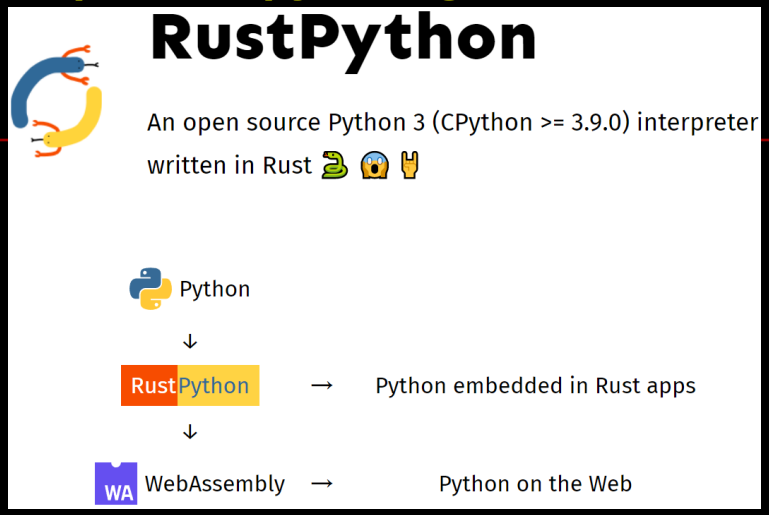
VII. Rust-based implementation

1) Overview

- …

2) RustPython

- https://rustpython.github.io/



RustPython is a Python interpreter written in Rust. RustPython can be embedded into Rust programs to use Python as a scripting language for your application, or it can be compiled to WebAssembly in order to run Python in the browser. RustPython is free and open-source under the MIT license.

- Should support WASI yet!
- Built-in support for any platform that support by Rust toolchain.
- Embedding

Embedding RustPython into your Rust Applications

Interested in exposing Python scripting in an application written in Rust, perhaps to allow quickly tweaking logic where Rust's compile times would be inhibitive? Then `examples/hello_embed.rs` and `examples/mini_repl.rs` may be of some assistance.

WASI

You can compile RustPython to a standalone WebAssembly WASI module so it can run anywhere.

Build

```
$ cargo build --target wasm32-wasi --no-default-features --features freeze-stdlib,stdlib --release
```

Run by wasmer

```
$ wasmer run --dir . target/wasm32-wasi/release/rustpython.wasm extra_tests/snippets/stdlib_random.py
```

Run by wapm

```
$ wapm install rustpython
$ wapm run rustpython
>>>>> 2+2
4
```

Building the WASI file

You can build the WebAssembly WASI file with:

```
cargo build --release --target wasm32-wasi --features="freeze-stdlib"
```

> Note: we use the `freeze-stdlib` to include the standard library inside the binary. You also have to run once `rustup target add wasm32-wasi`.

JIT

RustPython has an **very** experimental JIT compiler that compile python functions into native code.

Building

By default the JIT compiler isn't enabled, it's enabled with the `jit` cargo feature.

```
$ cargo run --features jit
```

This requires autoconf, automake, libtool, and clang to be installed.

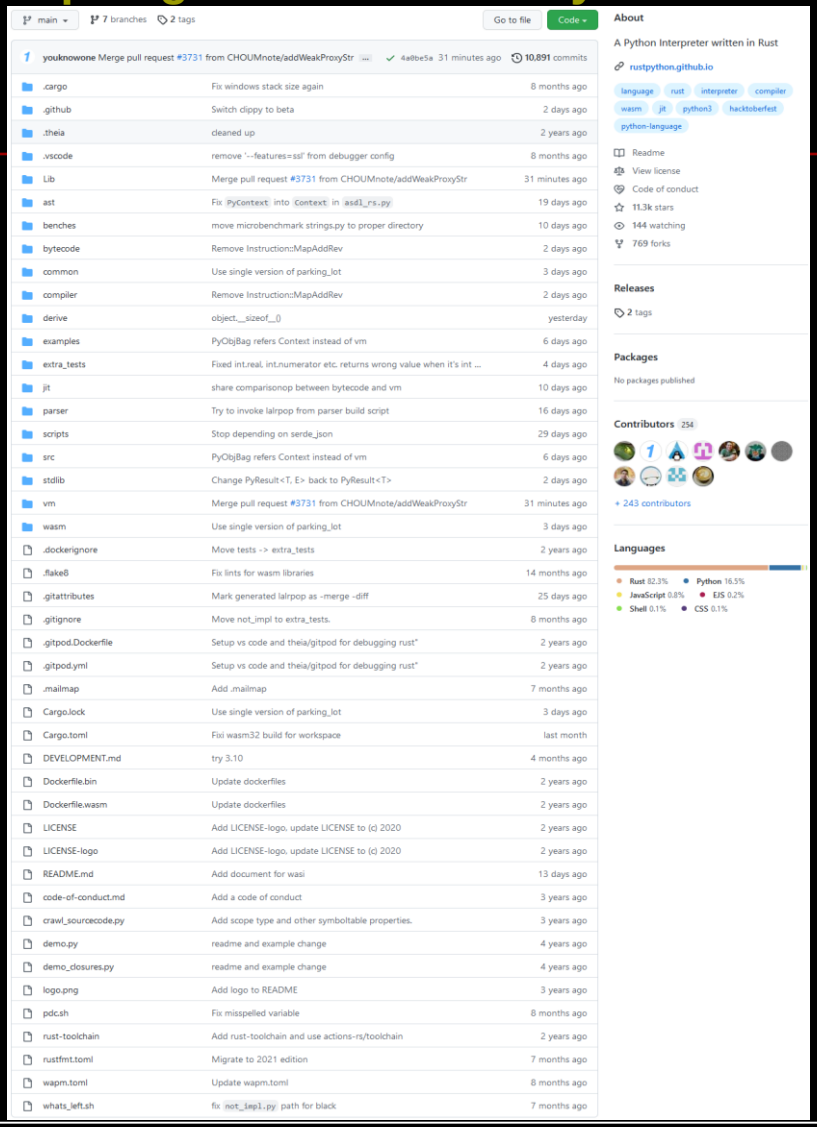
Using

To compile a function, call `__jit__()` on it.

```
def foo():
    a = 5
    return 10 + a

foo.__jit__()  # this will compile foo to native code and subsequent calls will execute that native code
assert foo() == 15
```

Project

- **https://github.com/RustPython/RustPython**



main · 7 branches · 2 tags · Go to file · Code

youknowone Merge pull request #3731 from CHOUMnote/addWeakProxyStr … 4e0be5a 31 minutes ago 10,891 commits

.cargo	Fix windows stack size again	8 months ago
.github	Switch clippy to beta	2 days ago
.theia	cleaned up	2 years ago
.vscode	remove '--features=ssl' from debugger config	8 months ago
Lib	Merge pull request #3731 from CHOUMnote/addWeakProxyStr	31 minutes ago
ast	Fix `PyContext` into `Context` in `asdl_rs.py`	19 days ago
benches	move microbenchmark strings.py to proper directory	10 days ago
bytecode	Remove Instruction::MapAddRev	2 days ago
common	Use single version of parking_lot	3 days ago
compiler	Remove Instruction::MapAddRev	2 days ago
derive	object.__sizeof__()	yesterday
examples	PyObjBag refers Context instead of vm	6 days ago
extra_tests	Fixed int.real, int.numerator etc. returns wrong value when it's int …	4 days ago
jit	share comparisonop between bytecode and vm	10 days ago
parser	Try to invoke lalrpop from parser build script	16 days ago
scripts	Stop depending on serde_json	29 days ago
src	PyObjBag refers Context instead of vm	6 days ago
stdlib	Change PyResult<T, E> back to PyResult<T>	2 days ago
vm	Merge pull request #3731 from CHOUMnote/addWeakProxyStr	31 minutes ago
wasm	Use single version of parking_lot	3 days ago
.dockerignore	Move tests -> extra_tests	2 years ago
.flake8	Fix lints for wasm libraries	14 months ago
.gitattributes	Mark generated lalrpop as -merge -diff	25 days ago
.gitignore	Move not_impl to extra_tests.	8 months ago
.gitpod.Dockerfile	Setup vs code and theia/gitpod for debugging rust"	2 years ago
.gitpod.yml	Setup vs code and theia/gitpod for debugging rust"	2 years ago
.mailmap	Add .mailmap	7 months ago
Cargo.lock	Use single version of parking_lot	3 days ago
Cargo.toml	Fixi wasm32 build for workspace	last month
DEVELOPMENT.md	try 3.10	4 months ago
Dockerfile.bin	Update dockerfiles	2 years ago
Dockerfile.wasm	Update dockerfiles	2 years ago
LICENSE	Add LICENSE-logo, update LICENSE to (c) 2020	2 years ago
LICENSE-logo	Add LICENSE-logo, update LICENSE to (c) 2020	2 years ago
README.md	Add document for wasi	13 days ago
code-of-conduct.md	Add a code of conduct	3 years ago
crawl_sourcecode.py	Add scope type and other symboltable properties.	3 years ago
demo.py	readme and example change	4 years ago
demo_closures.py	readme and example change	4 years ago
logo.png	Add logo to README	3 years ago
pdc.sh	Fix misspelled variable	8 months ago
rust-toolchain	Add rust-toolchain and use actions-rs/toolchain	2 years ago
rustfmt.toml	Migrate to 2021 edition	7 months ago
wapm.toml	Update wapm.toml	8 months ago
whats_left.sh	fix `not_impl.py` path for black	7 months ago

About

A Python Interpreter written in Rust

rustpython.github.io

language · rust · interpreter · compiler · wasm · jit · python3 · hacktoberfest · python-language

Readme · View license · Code of conduct · 11.3k stars · 144 watching · 769 forks

Releases

2 tags

Packages

No packages published

Contributors 254

+ 243 contributors

Languages

Rust 82.3% · Python 16.5% · JavaScript 0.8% · EJS 0.2% · Shell 0.1% · CSS 0.1%

Stats

```
[mydev@fedora RustPython-main]$ git branch
* main
[mydev@fedora RustPython-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    2           56           44            4            8
 Batch                   2           54           39            0           15
 C Shell                 1           25           11            5            9
 CSS                     4          373          301            8           64
 Fish                    1           64           38           13           13
 JavaScript             12          982          774          120           88
 JSON                    3           98           98            0            0
 PowerShell              2          249          110          107           32
 Python               1153       533242       427332        39987        65923
 ReStructuredText        1          216          165            0           51
 Shell                  11          260          159           36           65
 Plain Text             17         5048            0         4719          329
 TOML                   15          575          479           26           70
 XSL                     1            5            5            0            0
 XML                    55          297          274            7           16
-------------------------------------------------------------------------------
 HTML                    2          531          506            0           25
 |- CSS                 1            7            7            0            0
 (Total)                             538          513            0           25
-------------------------------------------------------------------------------
 Markdown               12          856            0          596          260
 |- Python              2           12            8            1            3
 |- Shell               3           18           18            0            0
 |- TOML                1            5            5            0            0
 |- TypeScript          1           36           17           17            2
 (Total)                             927           48          614          265
-------------------------------------------------------------------------------
 Rust                 261       143170       128086         5064        10020
 |- Markdown          128         1765            6         1542          217
 (Total)                          144935       128092         6606        10237
===============================================================================
 Total                1555       687944       558482        52252        77210
===============================================================================
[mydev@fedora RustPython-main]$
```

Build and test on RPi4

- Dev Env

```
[mydev@fedora RustPython-main]$ uname -a
Linux fedora 5.14.10-200.fc34.aarch64 #1 SMP Thu Oct 7 20:33:59 UTC 2021 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora RustPython-main]$
[mydev@fedora RustPython-main]$ cat ~/.cargo/config
[source.crates-io]
registry = "https://github.com/rust-lang/crates.io-index"
replace-with = 'ustc'

[source.ustc]
registry = "https://mirrors.ustc.edu.cn/crates.io-index"

[source.sjtu]
registry = "https://mirrors.sjtug.sjtu.edu.cn/git/crates.io-index/"

[source.tuna]
registry = "https://mirrors.tuna.tsinghua.edu.cn/git/crates.io-index.git"

[source.rustcc]
registry = "https://code.aliyun.com/rustcc/crates.io-index.git"
[mydev@fedora RustPython-main]$
```

```
[mydev@fedora RustPython-main]$ rustup -V
rustup 1.24.3 (ce5817a94 2021-05-31)
info: This is the version for the rustup toolchain manager, not the rustc compiler.
info: The currently active `rustc` version is `rustc 1.55.0 (c8dfcfe04 2021-09-06)`
[mydev@fedora RustPython-main]$
[mydev@fedora RustPython-main]$ rustup toolchain list
stable-aarch64-unknown-linux-gnu (default) (override)
[mydev@fedora RustPython-main]$
```

Build

```
[mydev@fedora RustPython-main]$ cargo build --release
  Downloaded Inflector v0.11.4 (registry `ustc`)
  Downloaded siphasher v0.3.6 (registry `ustc`)
  Downloaded smawk v0.3.1 (registry `ustc`)
  Downloaded socket2 v0.4.1 (registry `ustc`)
  Downloaded sre-engine v0.1.2 (registry `ustc`)
  Downloaded string_cache v0.8.1 (registry `ustc`)
  Downloaded strum v0.21.0 (registry `ustc`)
  Downloaded strum_macros v0.21.1 (registry `ustc`)
  Downloaded subtle v2.4.1 (registry `ustc`)
  Downloaded syn-ext v0.3.0 (registry `ustc`)
  Downloaded textwrap v0.13.4 (registry `ustc`)
  Downloaded timsort v0.1.2 (registry `ustc`)
...
  Downloaded volatile v0.3.0 (registry `ustc`)
  Downloaded xml-rs v0.8.4 (registry `ustc`)
  Downloaded 82 crates (3.6 MB) in 27.05s
   Compiling autocfg v1.0.1
   Compiling unicode-xid v0.2.2
   Compiling proc-macro2 v1.0.28
   Compiling syn v1.0.75
   Compiling libc v0.2.102
   Compiling serde v1.0.127
...
   Compiling rustpython-pylib v0.1.0 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/vm/pylib-crate)
   Compiling static_assertions v1.1.0
   Compiling arrayvec v0.5.2
   Compiling itoa v0.4.7
   Compiling lalrpop-util v0.19.6
   Compiling crossbeam-utils v0.8.5
   Compiling endian-type v0.1.2
   Compiling unicode-segmentation v1.8.0
   Compiling smawk v0.3.1
   Compiling build_const v0.2.2
   Compiling hexf-parse v0.1.0
   Compiling maplit v1.0.2
   Compiling utf8parse v0.2.0
   Compiling matches v0.1.9
   Compiling subtle v2.4.1
   Compiling ascii v1.0.0
   Compiling volatile v0.3.0
   Compiling exitcode v1.1.2
   Compiling strum v0.21.0
   Compiling half v1.7.1
   Compiling timsort v0.1.2
   Compiling keccak v0.1.0
   Compiling adler32 v1.2.0
   Compiling hex v0.4.3
   Compiling paste v1.0.5
   Compiling rustpython-stdlib v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/stdlib)
   Compiling unicode-casing v0.1.0
...
```

```
   Compiling rustpython-vm v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/vm)
   Compiling unic-normal v0.9.0
   Compiling flate2 v1.0.20
   Compiling chrono v0.4.19
   Compiling rand_chacha v0.3.1
   Compiling mt19937 v2.0.1
   Compiling caseless v0.2.1
   Compiling pmutil v0.5.3
   Compiling syn-ext v0.3.0
   Compiling rustyline v9.0.0
   Compiling thiserror-impl v1.0.26
   Compiling derivative v2.2.0
   Compiling strum_macros v0.21.1
   Compiling term v0.7.0
   Compiling sha3 v0.9.1
   Compiling sha2 v0.9.8
   Compiling sha-1 v0.9.8
   Compiling md-5 v0.9.1
   Compiling blake2 v0.9.2
   Compiling rand v0.8.4
   Compiling is-macro v0.1.9
   Compiling thiserror v1.0.26
   Compiling ascii-canvas v3.0.0
   Compiling result-like v0.3.0
   Compiling phf_generator v0.9.1
   Compiling lalrpop v0.19.6
   Compiling phf_macros v0.9.0
   Compiling toml v0.5.8
   Compiling bincode v1.3.3
   Compiling num-complex v0.4.0
   Compiling phf v0.9.0
   Compiling rustpython-ast v0.1.0 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/ast)
   Compiling proc-macro-crate v1.0.0
   Compiling rustpython-bytecode v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/bytecode)
   Compiling rustpython-common v0.0.0 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/common)
   Compiling num_enum_derive v0.5.4
   Compiling rustpython-compiler-core v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/compiler)
   Compiling num_enum v0.5.4
   Compiling sre-engine v0.1.2
   Compiling rustpython-parser v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/parser)
   Compiling rustpython-compiler v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/compiler/porcelain)
   Compiling rustpython-derive v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/derive)
   Compiling rustpython v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main)
    Finished release [optimized] target(s) in 21m 19s
[mydev@fedora RustPython-main]$
```

```
[mydev@fedora RustPython-main]$ cd target/release/;ll
total 21268
drwxr-xr-x. 1 mydev mydev      212 Oct 16 10:48 ./
drwxr-xr-x. 1 mydev mydev       70 Oct 16 10:48 ../
drwxr-xr-x. 1 mydev mydev     4402 Oct 16 10:27 build/
-rw-r--r--. 1 mydev mydev        0 Oct 16 10:27 .cargo-lock
drwxr-xr-x. 1 mydev mydev    47122 Oct 16 10:48 deps/
drwxr-xr-x. 1 mydev mydev        0 Oct 16 10:27 examples/
drwxr-xr-x. 1 mydev mydev    17344 Oct 16 10:27 .fingerprint/
drwxr-xr-x. 1 mydev mydev        0 Oct 16 10:27 incremental/
-rw-r--r--. 1 mydev mydev    19761 Oct 16 10:48 librustpython.d
-rw-r--r--. 2 mydev mydev  1032420 Oct 16 10:42 librustpython.rlib
-rwxr-xr-x. 2 mydev mydev 20739296 Oct 16 10:48 rustpython*
-rw-r--r--. 1 mydev mydev    19844 Oct 16 10:48 rustpython.d
[mydev@fedora release]$ file ./rustpython
./rustpython: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1]=48d2d641383b78f7029e4751745eed99f07e181f
, for GNU/Linux 3.7.0, with debug_info, not stripped
[mydev@fedora release]$
```

Test

```
[mydev@fedora release]$ ./rustpython --help
RustPython 0.1.2
RustPython Team
Rust implementation of the Python language

USAGE:
    rustpython [OPTIONS] [-c CMD | -m MODULE | FILE] [PYARGS]...

FLAGS:
    -b              issue warnings about using bytes where strings are usually expected (-bb: issue errors)
    -d              Debug the parser.
    -B              don't write .pyc files on import
    -h, --help      Prints help information
    -E              Ignore environment variables PYTHON* such as PYTHONPATH
    -i              Inspect interactively after running the script.
    -I              isolate Python from the user's environment (implies -E and -s)
    -S              don't imply 'import site' on initialization
    -s              don't add user site directory to sys.path.
    -O              Optimize. Set __debug__ to false. Remove debug statements.
    -q              Be quiet at startup.
    -u              force the stdout and stderr streams to be unbuffered; this option has no effect on stdin; also
                    PYTHONUNBUFFERED=x
    -V, --version   Prints version information
    -v              Give the verbosity (can be applied multiple times)

OPTIONS:
    -c <cmd, args>...                    run the given string as a program
    -X <implementation-option>...        set implementation-specific option
        --install-pip <get-pip args>...  install the pip package manager for rustpython
    -m <module, args>...                 run library module as script
    -W <warning-control>...              warning control; arg is action:message:category:module:lineno

ARGS:
    <script, args>...
[mydev@fedora release]$ ./rustpython
Welcome to the magnificent Rust Python 0.1.2 interpreter 😱 🖖
No previous history.
>>>>> 3+5
8
>>>>>
```

VIII. RPython-based implementation

1) Overview

- …

1.1 RPython

- https://rpython.readthedocs.io/en/latest/getting-started.html

RPython is a subset of Python that can be statically compiled. The PyPy interpreter is written mostly in RPython (with pieces in Python), while the RPython compiler is written in Python. The hard to understand part is that Python is a meta-programming language for RPython, that is, "being valid RPython" is a question that only makes sense on the live objects **after** the imports are done. This might require more explanation. You start writing RPython from `entry_point`, a good starting point is rpython/translator/goal/targetnopstandalone.py. This does not do all that much, but is a start. Now if code analyzed (in this case `entry_point`) calls some functions, those calls will be followed. Those followed calls have to be RPython themselves (and everything they call etc.), however not entire module files. To show how you can use metaprogramming, we can do a silly example (note that closures are not RPython):

```
def generator(operation):
    if operation == 'add':
       def f(a, b):
           return a + b
    else:
       def f(a, b):
           return a - b
    return f

add = generator('add')
sub = generator('sub')

def entry_point(argv):
    print add(sub(int(argv[1]), 3) 4)
    return 0
```

In this example `entry_point` is RPython, `add` and `sub` are RPython, however, `generator` is not.

...

- ...

2) PyPy

- https://www.pypy.org/

PyPy is a replacement for CPython. It is built using the RPython language that was co-developed with it. The main reason to use it instead of CPython is speed: it runs generally faster (see next section).

PyPy implements **Python 2.7.18, and 3.7.10**. It supports all of the core language, passing the Python 2.7 test suite and almost all of the 3.7 test suite (with minor modifications) It supports most of the commonly used Python standard library modules. For known differences with CPython, see our compatibility page.

The following CPU architectures are supported and maintained:

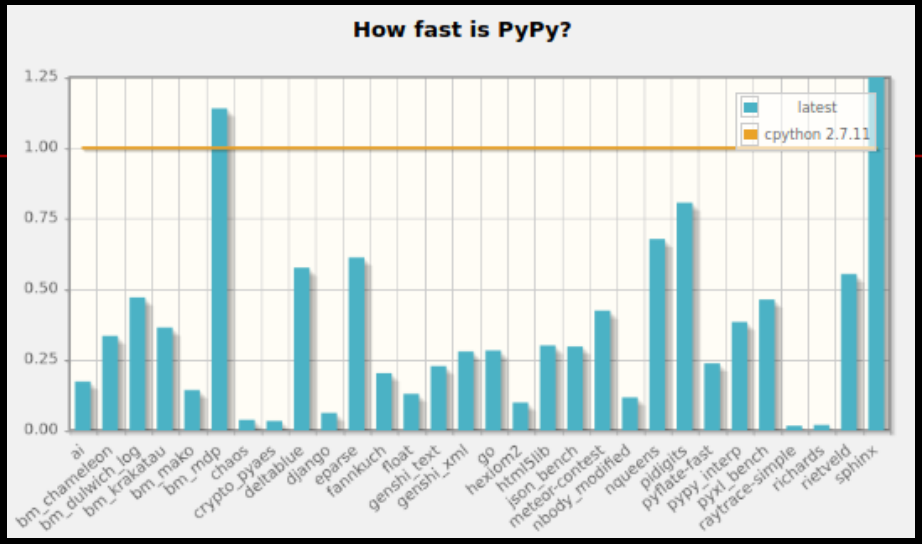
- x86 (IA-32) and x86_64
- ARM platforms (ARMv6 or ARMv7, with VFPv3)
- AArch64
- PowerPC 64bit both little and big endian
- System Z (s390x)

PyPy's x86 version runs on several operating systems, such as Linux (32/64 bits), Mac OS X (64 bits), Windows (32 bits), OpenBSD, FreeBSD. All non-x86 versions are only supported on Linux.

- https://doc.pypy.org
- https://www.pypy.org/compat.html
- …

- **Performance**

  On average, PyPy is 4.5 times faster than CPython



Features

Speed

Our main executable comes with a Just-in-Time compiler. It is really fast in running most benchmarks —including very large and complicated Python applications, not just 10-liners.

There are two cases that you should be aware where PyPy will *not* be able to speed up your code:

- Short-running processes: if it doesn't run for at least a few seconds, then the JIT compiler won't have enough time to warm up.
- If all the time is spent in run-time libraries (i.e. in C functions), and not actually running Python code, the JIT compiler will not help.

So the case where PyPy works best is when executing long-running programs where a significant fraction of the time is spent executing Python code. This is the case covered by the majority of our benchmarks, but not all of them --- the goal of PyPy is to get speed but still support (ideally) any Python program.

Memory usage

Memory-hungry Python programs (several hundreds of MBs or more) might end up taking less space than they do in CPython. It is not always the case, though, as it depends on a lot of details. Also note that the baseline is higher than CPython's.

Stackless

Support for Stackless and greenlets are now integrated in the normal PyPy. More detailed information is available here.

Other features

PyPy has many secondary features and semi-independent projects. We will mention here:

- **Other languages:** we also implemented other languages that makes use of our RPython toolchain: Prolog (almost complete), as well as Smalltalk, JavaScript, Io, Scheme and Gameboy.

  There is also a Ruby implementation called Topaz and a PHP implementation called HippyVM.

Sandboxing

PyPy's *sandboxing* is a working prototype for the idea of running untrusted user programs. Unlike other sandboxing approaches for Python, PyPy's does not try to limit language features considered "unsafe". Instead we replace all calls to external libraries (C or platform) with a stub that communicates with an external process handling the policy.

> Note
>
> **Please be aware that it is a prototype only.** It needs work to become more complete, and you are welcome to help. In particular, almost none of the extension modules work (not even `time`), and `pypy_interact` is merely a demo. Also, a more complete system would include a way to do the same as `pypy_interact` from other languages than Python, to embed a sandboxed interpreter inside programs written in other languages.

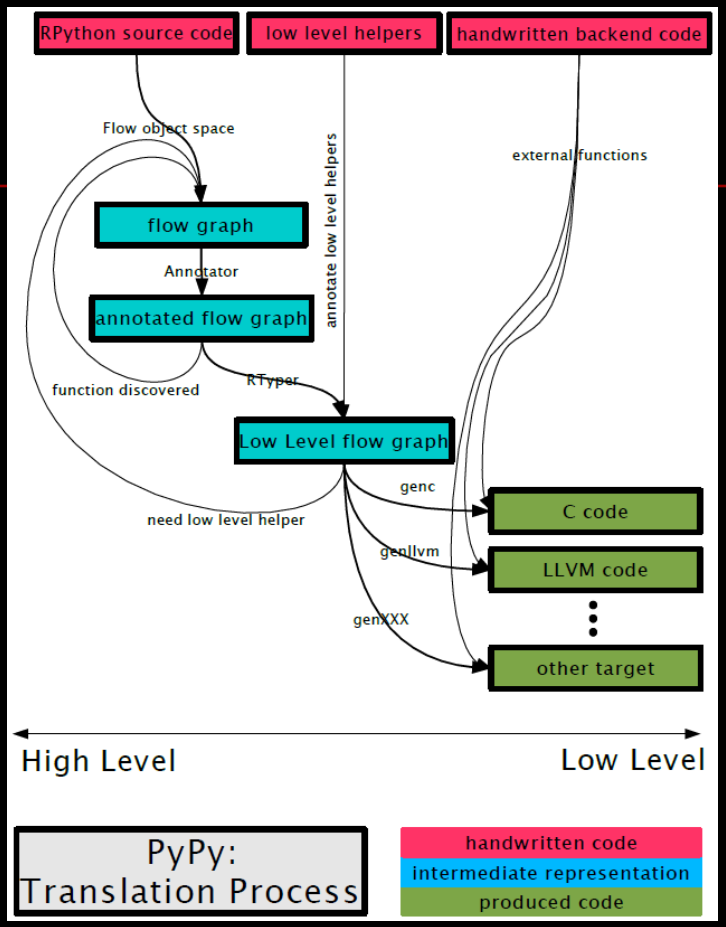
To run the sandboxed process, you need to get the full sources and build `pypy-sandbox` from it (see Building from source). These instructions give you a `pypy-c` that you should rename to `pypy-sandbox` to avoid future confusion. Then run:

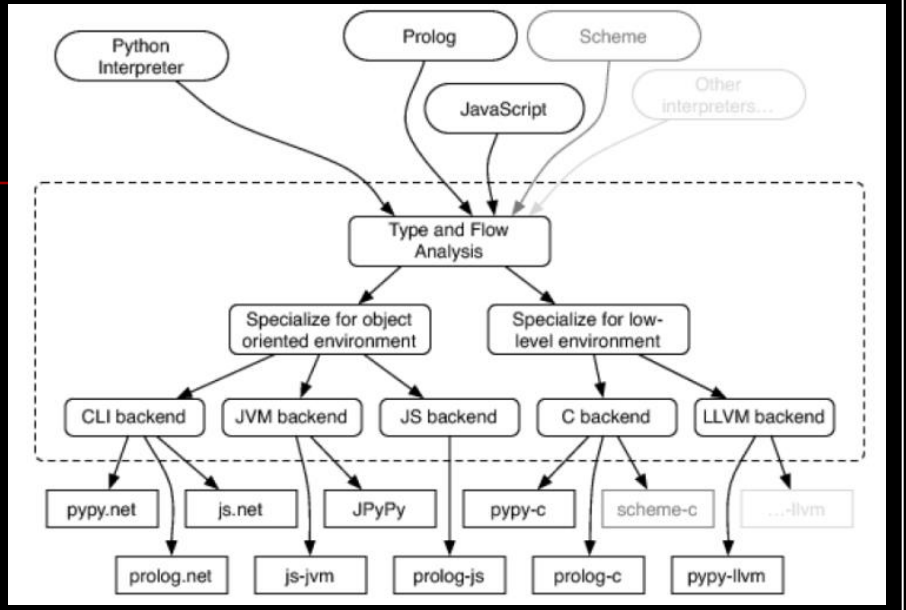
```
cd pypy/sandbox
pypy_interact.py path/to/pypy-sandbox
# don't confuse it with pypy/goal/pyinteractive.py!
```

Technologies

- **Meta-tracing**
  http://stefan-marr.de/papers/oopsla-marr-ducasse-meta-tracing-vs-partial-evaluation-artifacts/submitted-draft.pdf
- **…**

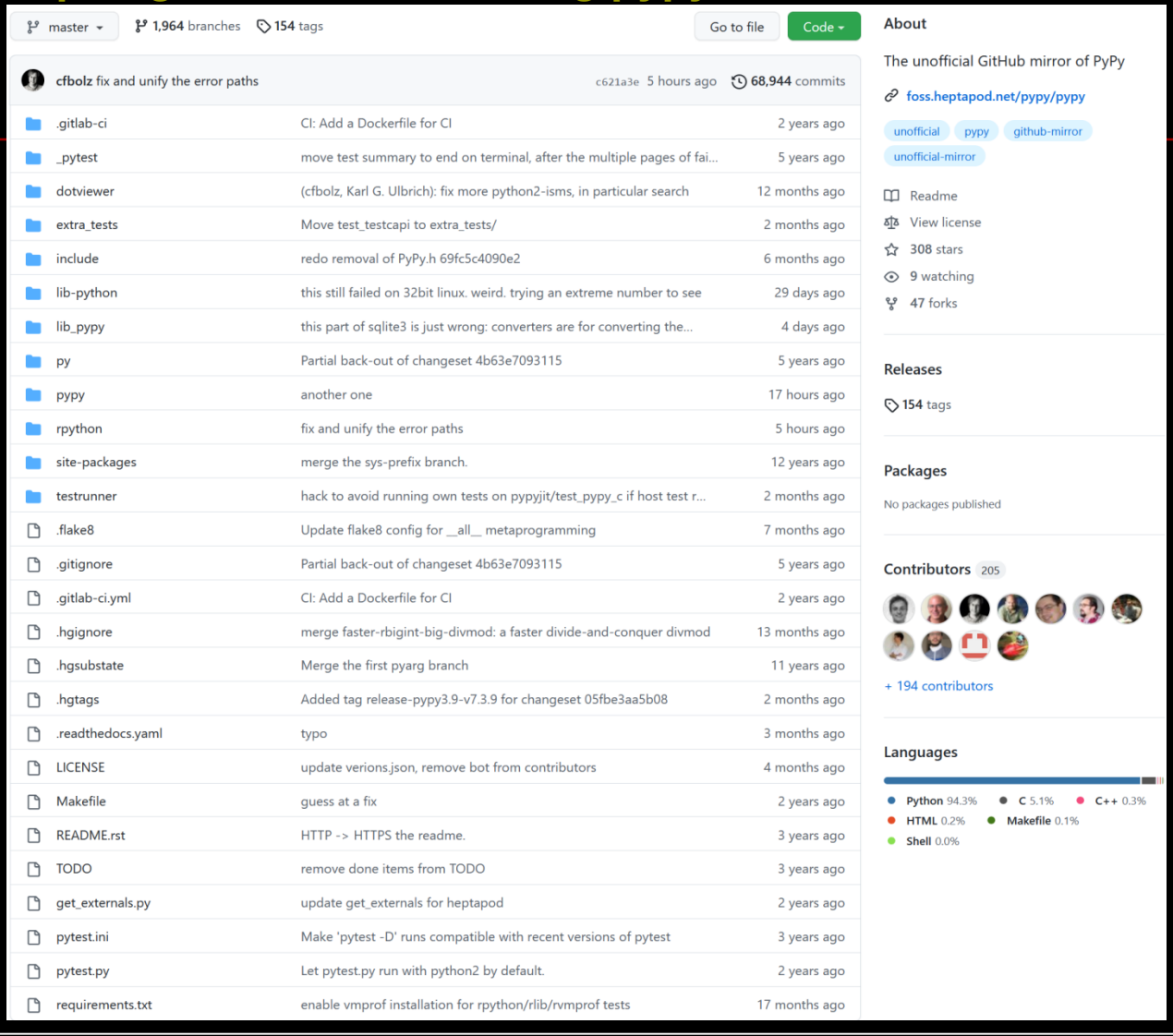
How it works


RPython source code
low level helpers
handwritten backend code
Flow object space
annotate low level helpers
external functions
flow graph
Annotator
annotated flow graph
function discovered
RTyper
Low Level flow graph
genc
C code
need low level helper
genllvm
LLVM code
genXXX
other target
High Level
Low Level
PyPy:
Translation Process
handwritten code
intermediate representation
produced code


Python Interpreter
Prolog
Scheme
JavaScript
Other interpreters...
Type and Flow Analysis
Specialize for object oriented environment
Specialize for low-level environment
CLI backend
JVM backend
JS backend
C backend
LLVM backend
pypy.net
js.net
JPyPy
pypy-c
scheme-c
...-llvm
prolog.net
js-jvm
prolog-js
prolog-c
pypy-llvm

Project (The unofficial GitHub mirror of PyPy)

- https://github.com/mozillazg/pypy



master · 1,964 branches · 154 tags · Go to file · Code

cfbolz fix and unify the error paths — c621a3e 5 hours ago — 68,944 commits

.gitlab-ci	CI: Add a Dockerfile for CI	2 years ago
_pytest	move test summary to end on terminal, after the multiple pages of fai...	5 years ago
dotviewer	(cfbolz, Karl G. Ulbrich): fix more python2-isms, in particular search	12 months ago
extra_tests	Move test_testcapi to extra_tests/	2 months ago
include	redo removal of PyPy.h 69fc5c4090e2	6 months ago
lib-python	this still failed on 32bit linux. weird. trying an extreme number to see	29 days ago
lib_pypy	this part of sqlite3 is just wrong: converters are for converting the...	4 days ago
py	Partial back-out of changeset 4b63e7093115	5 years ago
pypy	another one	17 hours ago
rpython	fix and unify the error paths	5 hours ago
site-packages	merge the sys-prefix branch.	12 years ago
testrunner	hack to avoid running own tests on pypyjit/test_pypy_c if host test r...	2 months ago
.flake8	Update flake8 config for __all__ metaprogramming	7 months ago
.gitignore	Partial back-out of changeset 4b63e7093115	5 years ago
.gitlab-ci.yml	CI: Add a Dockerfile for CI	2 years ago
.hgignore	merge faster-rbigint-big-divmod: a faster divide-and-conquer divmod	13 months ago
.hgsubstate	Merge the first pyarg branch	11 years ago
.hgtags	Added tag release-pypy3.9-v7.3.9 for changeset 05fbe3aa5b08	2 months ago
.readthedocs.yaml	typo	3 months ago
LICENSE	update verions.json, remove bot from contributors	4 months ago
Makefile	guess at a fix	2 years ago
README.rst	HTTP -> HTTPS the readme.	3 years ago
TODO	remove done items from TODO	3 years ago
get_externals.py	update get_externals for heptapod	2 years ago
pytest.ini	Make 'pytest -D' runs compatible with recent versions of pytest	3 years ago
pytest.py	Let pytest.py run with python2 by default.	2 years ago
requirements.txt	enable vmprof installation for rpython/rlib/rvmprof tests	17 months ago

About

The unofficial GitHub mirror of PyPy

foss.heptapod.net/pypy/pypy

unofficial · pypy · github-mirror · unofficial-mirror

Readme · View license · 308 stars · 9 watching · 47 forks

Releases

154 tags

Packages

No packages published

Contributors 205

+ 194 contributors

Languages

Python 94.3% · C 5.1% · C++ 0.3% · HTML 0.2% · Makefile 0.1% · Shell 0.0%

Stats

```
[mydev@fedora pypy-master]$ git branch
* master
[mydev@fedora pypy-master]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Assembly                2          145           97           18           30
 Autoconf                3          970          726          107          137
 Automake                1          136           71           36           29
 Batch                   5          451          392            3           56
 C                     139        55273        43624         5376         6273
 C Header              183        36060        28927         3967         3166
 C++                    14         2778         2235          121          422
 CSS                     1           63           49            0           14
 Emacs Lisp              1           62           45            7           10
 HCL                     2           13           13            0            0
 INI                     3            4            4            0            0
 JSON                    3         2569         2569            0            0
 Makefile                6          499          394           14           91
 Markdown                2           24            0           17            7
 Module-Definition       5         1257         1191            0           66
 Python               4532      2252266      2006745        83590       161931
 ReStructuredText      204        29734        21670            0         8064
 Shell                   7          694          475          135           84
 Plain Text            406       664767            0       660684         4083
 VBScript                1            1            0            1            0
 Vim script              1           33           31            2            0
 XML                    17          433          313            4          116
-------------------------------------------------------------------------------
 HTML                    4         1477         1424            6           47
 |- CSS                  2           34           34            0            0
 |- JavaScript           2           17           16            1            0
 (Total)                            1528         1474            7           47
===============================================================================
 Total                5542      3049760      2111045       754089       184626
===============================================================================
[mydev@fedora pypy-master]$
[mydev@fedora pypy-master]$ du -sh
639M    .
[mydev@fedora pypy-master]$
```

X. Tittle-tattle

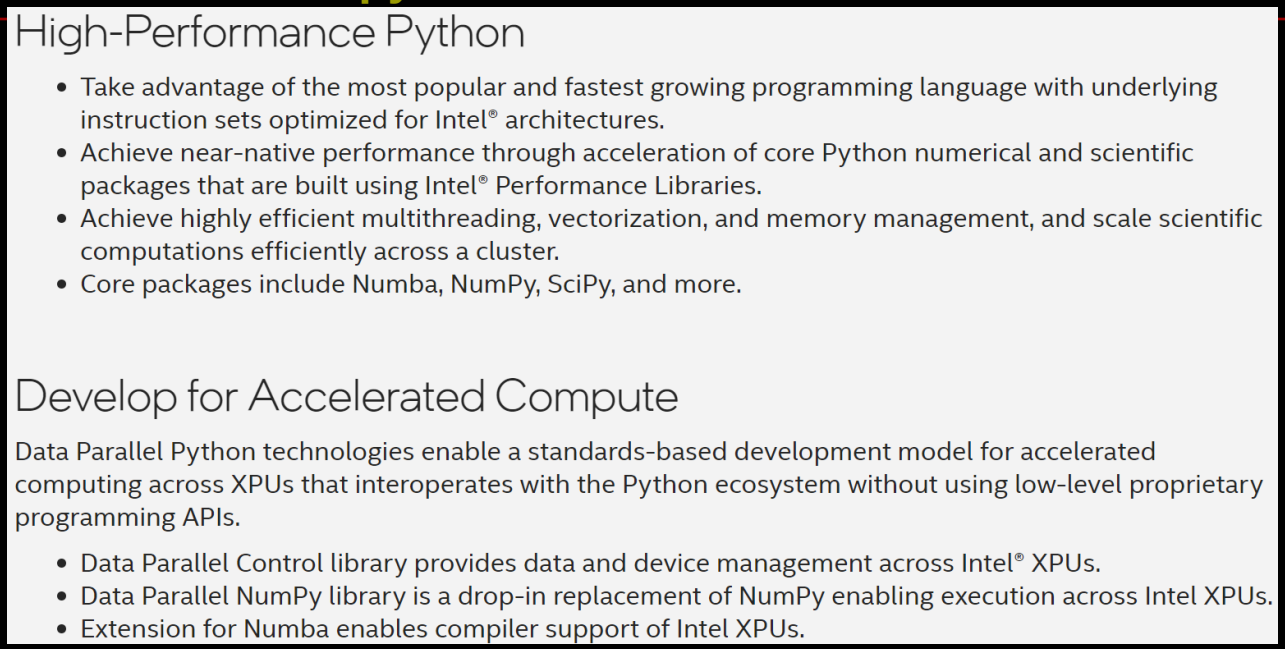
1) Vendor-specific Distributions

- https://

1.1 Intel Distribution for Python

Overview

- **https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-python.html**



High-Performance Python

- Take advantage of the most popular and fastest growing programming language with underlying instruction sets optimized for Intel® architectures.
- Achieve near-native performance through acceleration of core Python numerical and scientific packages that are built using Intel® Performance Libraries.
- Achieve highly efficient multithreading, vectorization, and memory management, and scale scientific computations efficiently across a cluster.
- Core packages include Numba, NumPy, SciPy, and more.

Develop for Accelerated Compute

Data Parallel Python technologies enable a standards-based development model for accelerated computing across XPUs that interoperates with the Python ecosystem without using low-level proprietary programming APIs.

- Data Parallel Control library provides data and device management across Intel® XPUs.
- Data Parallel NumPy library is a drop-in replacement of NumPy enabling execution across Intel XPUs.
- Extension for Numba enables compiler support of Intel XPUs.

- **https://github.com/IntelPython**
  Examples and other resources for Intel Distribution for Python.

2) Derivatives

Overview

- https://

2.1 MyPy

Overview

- **http://www.mypy-lang.org/**

Mypy is an optional static type checker for Python that aims to combine the benefits of dynamic (or "duck") typing and static typing. Mypy combines the expressive power and convenience of Python with a powerful type system and compile-time type checking. Mypy type checks standard Python programs; run them using any Python VM with basically no runtime overhead.

- **Seamless dynamic and static typing**

From Python...

```
def fib(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

...to statically typed Python

```
def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

- **Why is it**

Why mypy?

Compile-time type checking

Static typing makes it easier to find bugs with less debugging.

Easier maintenance

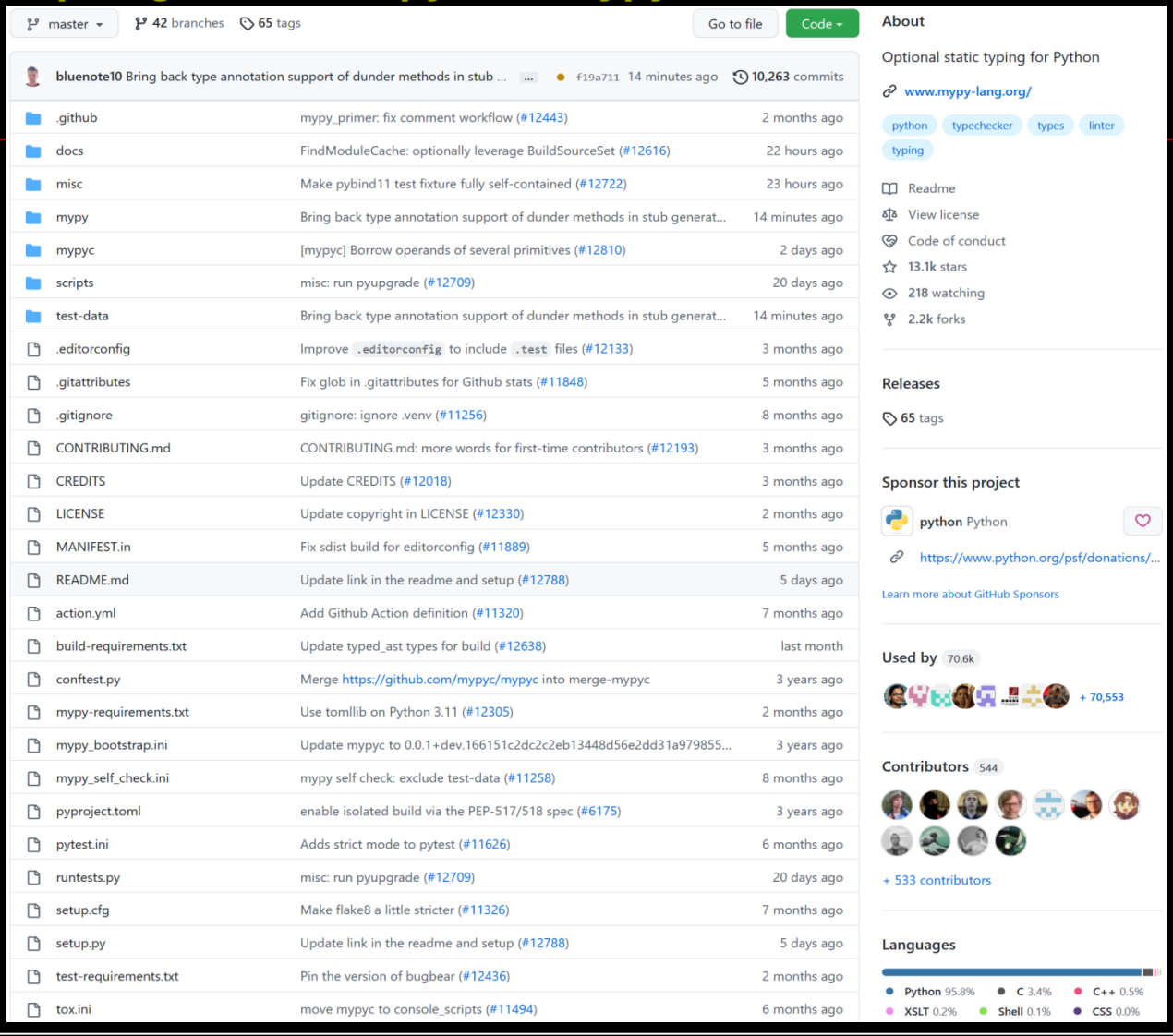
Type declarations act as machine-checked documentation. Static typing makes your code easier to understand and easier to modify without introducing bugs.

Grow your programs from dynamic to static typing

You can develop programs with dynamic typing and add static typing after your code has matured, or migrate existing Python code to static typing.

Project

- https://github.com/python/mypy



master · 42 branches · 65 tags · Go to file · Code

bluenote10 Bring back type annotation support of dunder methods in stub ... f19a711 14 minutes ago 10,263 commits

.github	mypy_primer: fix comment workflow (#12443)	2 months ago
docs	FindModuleCache: optionally leverage BuildSourceSet (#12616)	22 hours ago
misc	Make pybind11 test fixture fully self-contained (#12722)	23 hours ago
mypy	Bring back type annotation support of dunder methods in stub generat...	14 minutes ago
mypyc	[mypyc] Borrow operands of several primitives (#12810)	2 days ago
scripts	misc: run pyupgrade (#12709)	20 days ago
test-data	Bring back type annotation support of dunder methods in stub generat...	14 minutes ago
.editorconfig	Improve `.editorconfig` to include `.test` files (#12133)	3 months ago
.gitattributes	Fix glob in .gitattributes for Github stats (#11848)	5 months ago
.gitignore	gitignore: ignore .venv (#11256)	8 months ago
CONTRIBUTING.md	CONTRIBUTING.md: more words for first-time contributors (#12193)	3 months ago
CREDITS	Update CREDITS (#12018)	3 months ago
LICENSE	Update copyright in LICENSE (#12330)	2 months ago
MANIFEST.in	Fix sdist build for editorconfig (#11889)	5 months ago
README.md	Update link in the readme and setup (#12788)	5 days ago
action.yml	Add Github Action definition (#11320)	7 months ago
build-requirements.txt	Update typed_ast types for build (#12638)	last month
conftest.py	Merge https://github.com/mypyc/mypyc into merge-mypyc	3 years ago
mypy-requirements.txt	Use tomllib on Python 3.11 (#12305)	2 months ago
mypy_bootstrap.ini	Update mypyc to 0.0.1+dev.166151c2dc2c2eb13448d56e2dd31a979855...	3 years ago
mypy_self_check.ini	mypy self check: exclude test-data (#11258)	8 months ago
pyproject.toml	enable isolated build via the PEP-517/518 spec (#6175)	3 years ago
pytest.ini	Adds strict mode to pytest (#11626)	6 months ago
runtests.py	misc: run pyupgrade (#12709)	20 days ago
setup.cfg	Make flake8 a little stricter (#11326)	7 months ago
setup.py	Update link in the readme and setup (#12788)	5 days ago
test-requirements.txt	Pin the version of bugbear (#12436)	2 months ago
tox.ini	move mypyc to console_scripts (#11494)	6 months ago

About

Optional static typing for Python

www.mypy-lang.org/

python · typechecker · types · linter · typing

Readme
View license
Code of conduct
13.1k stars
218 watching
2.2k forks

Releases

65 tags

Sponsor this project

python Python

https://www.python.org/psf/donations/...

Learn more about GitHub Sponsors

Used by 70.6k

+ 70,553

Contributors 544

+ 533 contributors

Languages

Python 95.8% · C 3.4% · C++ 0.5% · XSLT 0.2% · Shell 0.1% · CSS 0.0%

Stats

```
[mydev@fedora mypy-master]$ git branch
* master
[mydev@fedora mypy-master]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Autoconf                1           47           31            8            8
 Batch                   2          277          238            2           37
 C                      13         3747         3012          385          350
 C Header                5         2671         1635          607          429
 C++                    11         9818         6421         2173         1224
 CSS                     1          104           70           10           24
 Emacs Lisp              1           22           18            2            2
 INI                     4          136          110           13           13
 Makefile                3          258          171           41           46
 Python                366       114820        88970        10388        15462
 ReStructuredText       54        15738        10959            0         4779
 Shell                   5           88           48           23           17
 Plain Text              7          142            0          115           27
 TOML                    3           17           14            2            1
 XSL                     2          181          168            3           10
 YAML                    1           83           75            0            8
-------------------------------------------------------------------------------
 Markdown               10         1643            0         1171          472
 |- Python               2            7            6            0            1
 (Total)                            1650            6         1171          473
===============================================================================
 Total                489       149799       111946        14943        22910
===============================================================================
[mydev@fedora mypy-master]$
```

mypyc

- **https://github.com/mypyc/mypyc**

Mypyc compiles Python modules to C extensions. It uses standard Python type hints to generate fast code. Mypyc uses mypy to perform type checking and type inference.

Mypyc can compile anything from one module to an entire codebase. The mypy project has been using mypyc to compile mypy since 2019, giving it a 4x performance boost over regular Python.

- **Features**

- Support most features in the stdlib `typing` module
- Compile clean, regular-looking Python code with type annotations
- Expressive type system, including generics, optional types, union types and tuple types
- Powerful type inference -- no need to annotate most variables
- All code is valid Python, and all Python editors and IDEs work just fine
- Access to all stdlib and third-party libraries in compiled code
- Strict runtime enforcement of type annotations for runtime type safety
- Ahead-of-time compilation for fast program startup
- Compiled code runs as normal Python code (compilation is optional)
- Both static type checking (via mypy) and runtime type checking

- **https://mypyc.readthedocs.io/**
- **…**

Good Resources

- https://mypy.readthedocs.io/
- https://peps.python.org/pep-0484/
- https://peps.python.org/pep-0526/
- …

X. Development

1) Dev Env

1.1 Overview

- https://

1.3 Shell
1.3.1 IPython
Overview

- https://ipython.org/

IPython provides a rich architecture for interactive computing with:

- A powerful interactive shell.
- A kernel for Jupyter.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into your own projects.
- Easy to use, high performance tools for parallel computing.

```
In [1]: print('Hello IPython')
Hello IPython

In [2]: 21 * 2
Out[2]: 42

In [3]: def say_hello(name):
   ...:     print('Hello {name}'.format(name=name))
   ...:
```

```
In [1]: %timeit range(1000)
100000 loops, best of 3: 7.76 us per loop

In [2]: %%timeit x = range(10000)
...: max(x)
...:
1000 loops, best of 3: 223 us per loop
```

- Jupyter Notebook(formerly known as the IPython Notebook)

Jupyter and the future of IPython

IPython is a growing project, with increasingly language-agnostic components. IPython 3.x was the last monolithic release of IPython, containing the notebook server, qtconsole, etc. As of IPython 4.0, the language-agnostic parts of the project: the notebook format, message protocol, qtconsole, notebook web application, etc. have moved to new projects under the name Jupyter. IPython itself is focused on interactive Python, part of which is providing a Python kernel for Jupyter.

- https://github.com/ipython
- https://ipython.readthedocs.io/
- ...

2) Benchmarking

2.1 Overview

- https://

1.2 Pyperformance

Overview

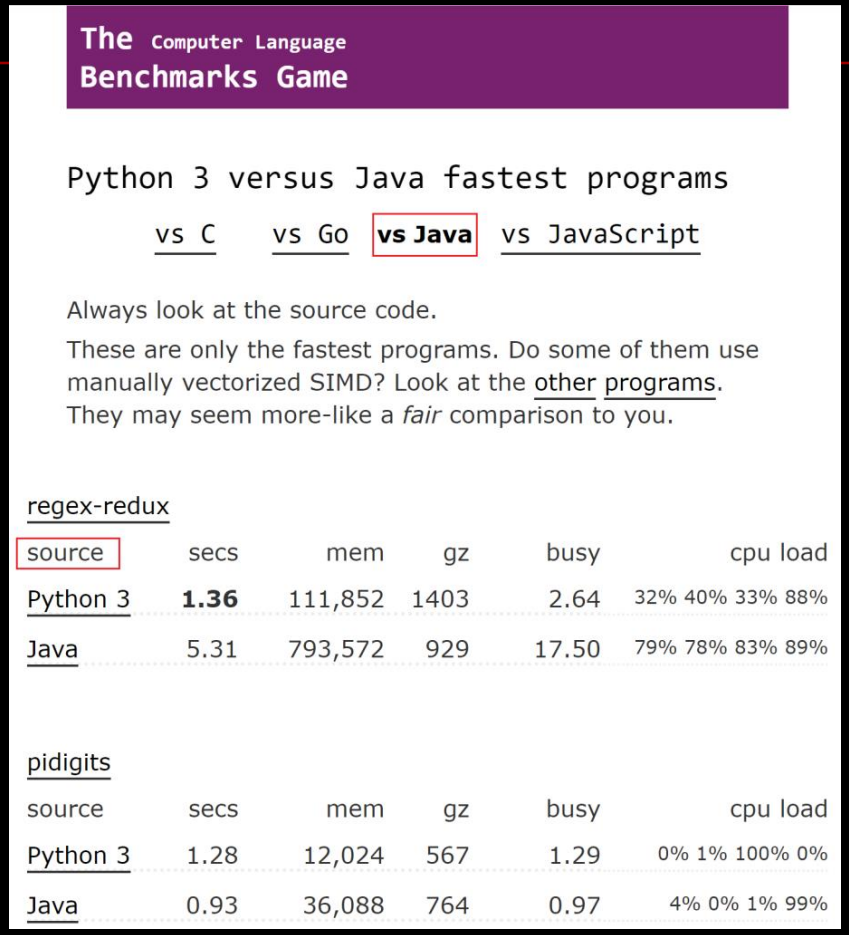
- https://github.com/python/pyperformance
- Python Performance Benchmark Suite (mainly targets CPython)
- https://pyperformance.readthedocs.io/
- https://pyperformance.readthedocs.io/usage.html#run-benchmarks
  ...
- Be fit for standalone Python runtime, but not friendly for cases like IronPython, Pyjion, RustPython...

Summary

- Prebuilt PyPy for Linux AArch64 is not stable during the test, crash or timeout often occurred.
- RustPython is awesome, but it is still in development and lacks of some features to be used as a standalone Python runtime for testing.
- Made IronPython successfully running on RPi4B, but still got failed to run Pyperformance since the latter does not well handle the case of “Python executable” which need a “wrapper” (just like Mono for IronPython).

1.2 Computer Language Benchmarks Game

- httpshttps://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html



The Computer Language Benchmarks Game

Python 3 versus Java fastest programs

vs C vs Go **vs Java** vs JavaScript

Always look at the source code.

These are only the fastest programs. Do some of them use manually vectorized SIMD? Look at the other programs. They may seem more-like a *fair* comparison to you.

regex-redux

source	secs	mem	gz	busy	cpu load
Python 3	**1.36**	111,852	1403	2.64	32% 40% 33% 88%
Java	5.31	793,572	929	17.50	79% 78% 83% 89%

pidigits

source	secs	mem	gz	busy	cpu load
Python 3	1.28	12,024	567	1.29	0% 1% 100% 0%
Java	0.93	36,088	764	0.97	4% 0% 1% 99%

...

- **https://benchmarksgame-team.pages.debian.net/benchmarksgame/how-programs-are-measured.html**

Measured on a quad-core 3.0GHz Intel® i5-3330® with 15.8 GiB of RAM and 2TB SATA disk drive; using Ubuntu™ 21.04 x86_64 GNU/Linux 5.11.0-18-generic.

No doubt on current hardware all the programs would be faster. Would that make the measurements more informative or just different?

Who wrote the programs

The programs have been crowd sourced, contributed to the project by an ever-changing self-selected group.

How programs are measured

1. Each program is run and measured at the smallest input value, program output redirected to a file and compared to expected output. As long as the output matches expected output, the program is then run and measured at the next larger input value until measurements have been made at every input value.
2. If the program gives the expected output within an arbitrary cutoff time (120 seconds) the program is measured again (5 more times) with output redirected to `/dev/null`.
3. If the program doesn't give the expected output within an arbitrary timeout (usually one hour) the program is forced to quit. If measurements at a smaller input value have been successful within an arbitrary cutoff time (120 seconds), the program is measured again (5 more times) at that smaller input value, with output redirected to `/dev/null`.
4. The measurements shown on the website are either:
   - within the arbitrary cutoff - the lowest time and highest memory use from 6 measurements
   - outside the arbitrary cutoff - the sole time and memory use measurement
5. For sure, programs taking 4 and 5 hours were only measured once!

How programs are timed

Each program is run as a child-process of a Python script using `Popen`:

- secs - The time is taken before forking the child-process and after the child-process exits, using `time.time()`
- cpu - The script child-process `usr+sys rusage` time is taken using `os.wait3`. Rarely (for example OCaml), that may not measure all processes forked from the script child-process.
- busy - The GTop cpu idle and GTop cpu total are taken before forking the child-process and after the child-process exits. The sum of GTop cpu not-idle for each core, scaled by secs.

On win32:

- secs - The time is taken before forking the child-process and after the child-process exits, using `QueryPerformanceCounter`
- cpu - `QueryInformationJobObject(hJob,JobObjectBasicAccountingInformation)` `TotalKernelTime + TotalUserTime`

(Note: Those measurements include startup time).

How program memory use is measured

By sampling `GLIBTOP_PROC_MEM_RESIDENT` for the program and it's child processes every 0.04 seconds. Obviously those measurements are unlikely to be reliable for programs that run for less than 0.04 seconds.

On win32: `QueryInformationJobObject(hJob,JobObjectExtendedLimitInformation)` `PeakJobMemoryUsed`

How source code size is measured

We start with the source-code markup you can see, remove comments, remove duplicate whitespace characters, and then apply minimum GZip compression. The measurement is the size in bytes of that GZip compressed source-code file.

Thanks to Brian Hurt for the idea of using **size of compressed source code** instead of lines of code.

median source code gzip (July 2018)	
Perl	513
TypeScript	532
Lua	553
Ruby	568
Dart	610
Chapel	632
Racket	638
Python	672
PHP	736
Hack	745
JavaScript	779
Erlang	792
Go	829
Haskell	842
Pascal	846
F#	876
OCaml	914
Java	945
Smalltalk	950
Lisp	1004
Fortran	1019
C++	1044
C#	1059
C	1115
Swift	1164
Rust	1319
Ada	1819

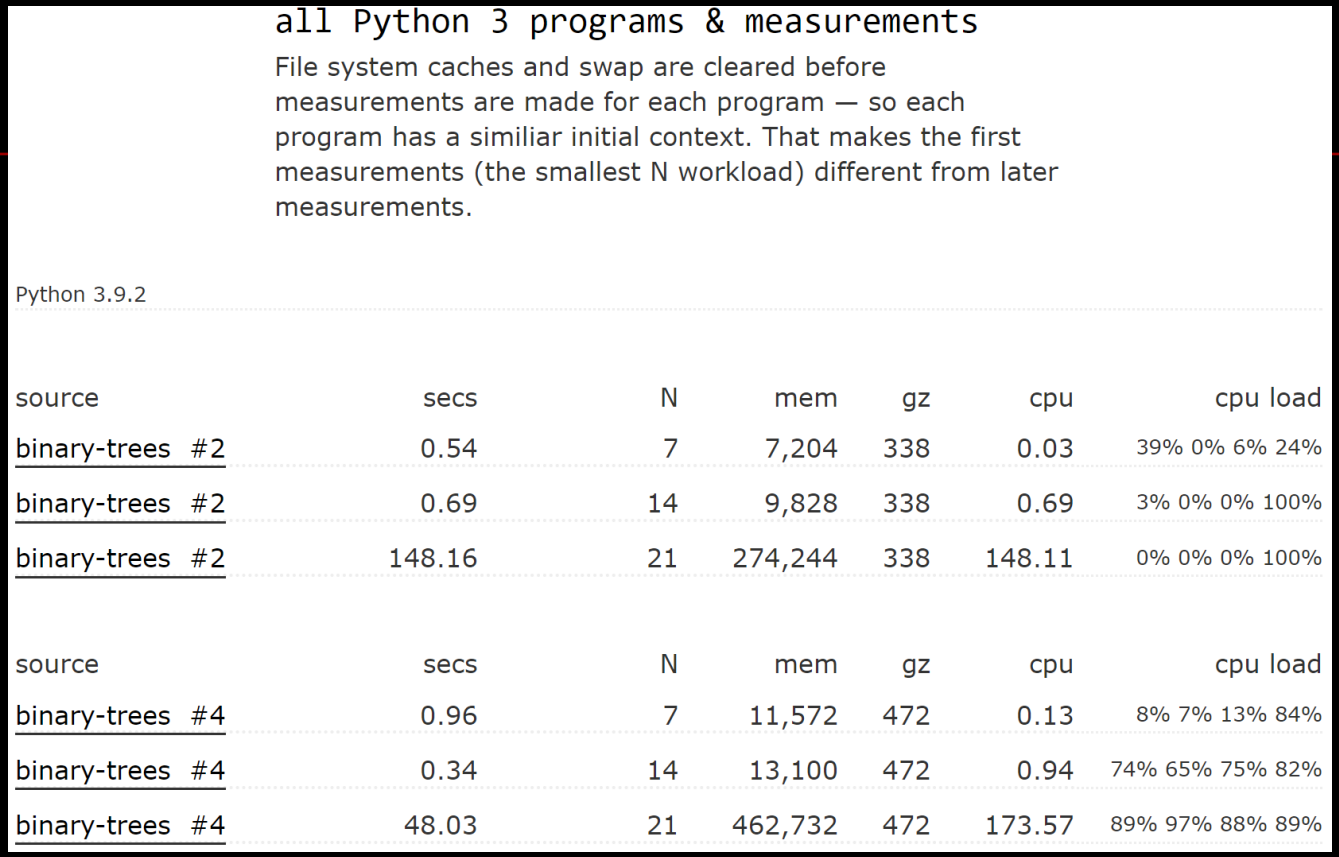
(Note: There is some evidence that complexity metrics don't provide any more information than SLoC or LoC.)

How CPU load is measured

The GTop cpu idle and GTop cpu total are taken before forking the child-process and after the child-process exits. The percentages represent the proportion of cpu not-idle to cpu total for each core.

On win32: `GetSystemTimes` `UserTime` `IdleTime` are taken before forking the child-process and after the child-process exits. The percentage represents the proportion of `TotalUserTime` to `UserTime + IdleTime` (because that's like the percentage you'll see in Task Manager).

- **https://benchmarksgame-team.pages.debian.net/benchmarksgame/measurements/python3.html**



all Python 3 programs & measurements

File system caches and swap are cleared before measurements are made for each program — so each program has a similiar initial context. That makes the first measurements (the smallest N workload) different from later measurements.

Python 3.9.2

source	secs	N	mem	gz	cpu	cpu load
binary-trees #2	0.54	7	7,204	338	0.03	39% 0% 6% 24%
binary-trees #2	0.69	14	9,828	338	0.69	3% 0% 0% 100%
binary-trees #2	148.16	21	274,244	338	148.11	0% 0% 0% 100%

source	secs	N	mem	gz	cpu	cpu load
binary-trees #4	0.96	7	11,572	472	0.13	8% 7% 13% 84%
binary-trees #4	0.34	14	13,100	472	0.94	74% 65% 75% 82%
binary-trees #4	48.03	21	462,732	472	173.57	89% 97% 88% 89%

...

1.3 Multilingual

- https://github.com/kostya/benchmarks

3) Profiling
3.1 Official

- **https://docs.python.org/3/library/profile.html**

`cProfile` and `profile` provide *deterministic profiling* of Python programs. A *profile* is a set of statistics that describes how often and for how long various parts of the program executed. These statistics can be formatted into reports via the `pstats` module.

The Python standard library provides two different implementations of the same profiling interface:

1. `cProfile` is recommended for most users; it's a C extension with reasonable overhead that makes it suitable for profiling long-running programs. Based on `lsprof`, contributed by Brett Rosen and Ted Czotter.
2. `profile`, a pure Python module whose interface is imitated by `cProfile`, but which adds significant overhead to profiled programs. If you're trying to extend the profiler in some way, the task might be easier with this module. Originally designed and written by Jim Roskind.

Note: The profiler modules are designed to provide an execution profile for a given program, not for benchmarking purposes (for that, there is `timeit` for reasonably accurate results). This particularly applies to benchmarking Python code against C code: the profilers introduce overhead for Python code, but not for C-level functions, and so the C code would seem faster than any Python one.

...

- **https://docs.python.org/3/library/timeit.html**

This module provides a simple way to time small bits of Python code. It has both a Command-Line Interface as well as a callable one. It avoids a number of common traps for measuring execution times. See also Tim Peters' introduction to the "Algorithms" chapter in the second edition of *Python Cookbook*, published by O'Reilly.

3.2 Pyroscope

- https://pyroscope.io/
- https://github.com/pyroscope-io/pyroscope

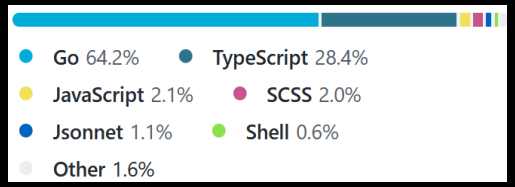
Continuous Profiling Platform! Debug performance issues down to a single line of code.

- Find performance issues and bottlenecks in your code
- Resolve issues with high CPU utilization
- Understand the call tree of your application
- Track changes over time

- Features

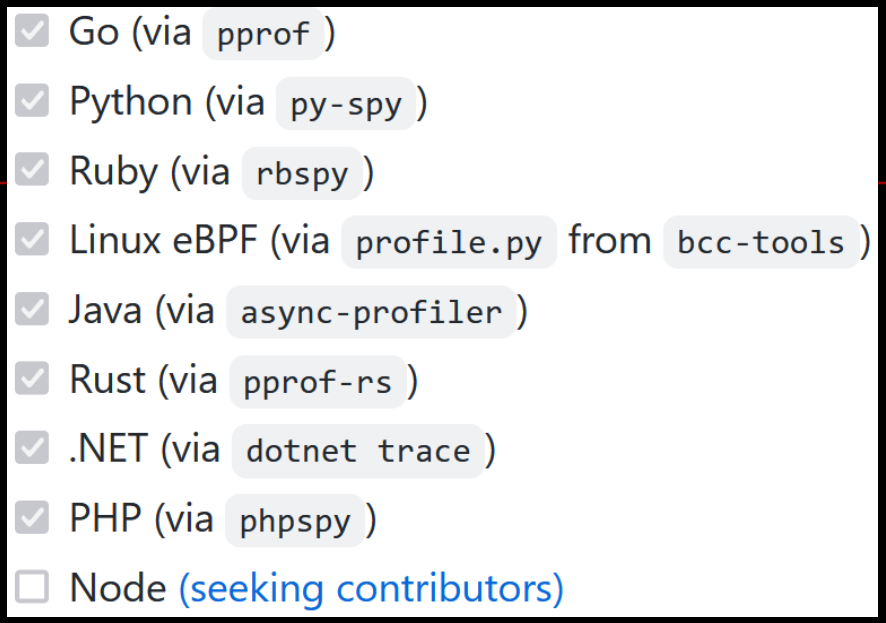
- Can store years of profiling data from multiple applications
- You can look at years of data at a time or zoom in on specific events
- Low CPU overhead
- Efficient compression, low disk space requirements
- Snappy UI

- Languages



- https://pyroscope.io/blog/
- https://pyroscope.io/docs/ebpf/
- …

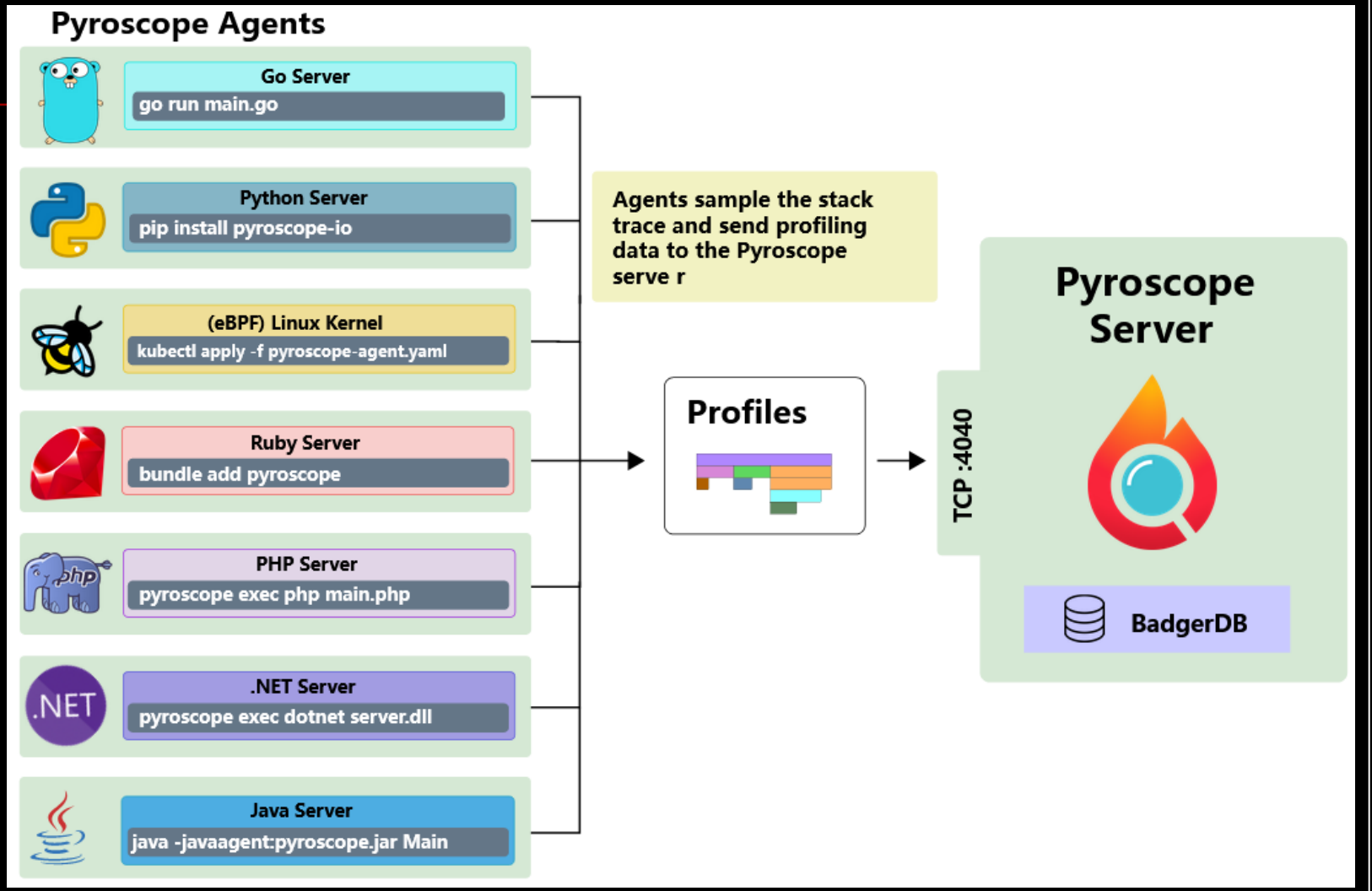
Supported Integrations

- 

- **https://pyroscope.io/docs/supported-integrations**

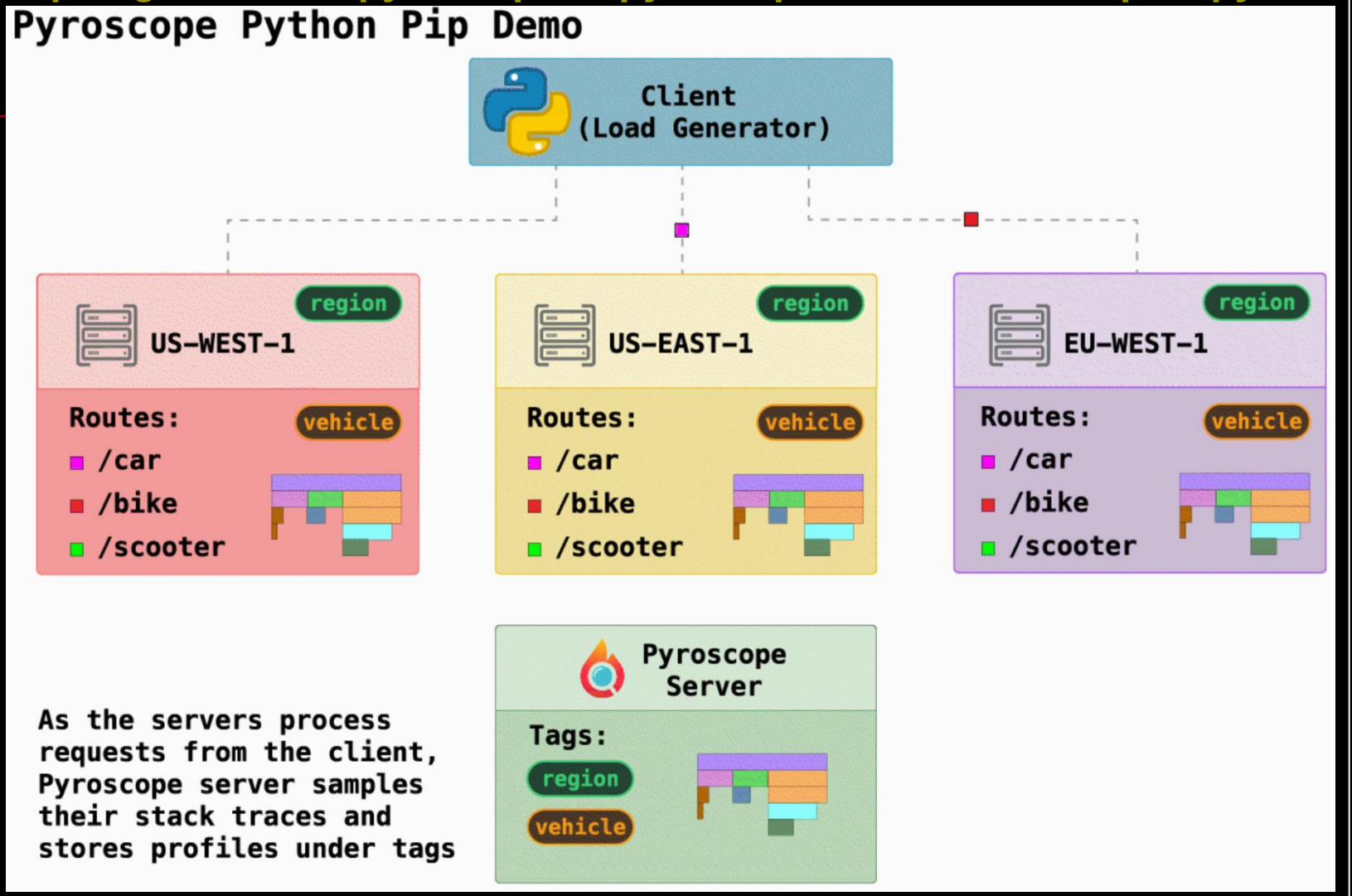
How it works

- https://pyroscope.io/docs/



Example

- https://github.com/pyroscope-io/pyroscope/tree/main/examples/python



3.3 gProfiler

Overview

- https://gprofiler.io/

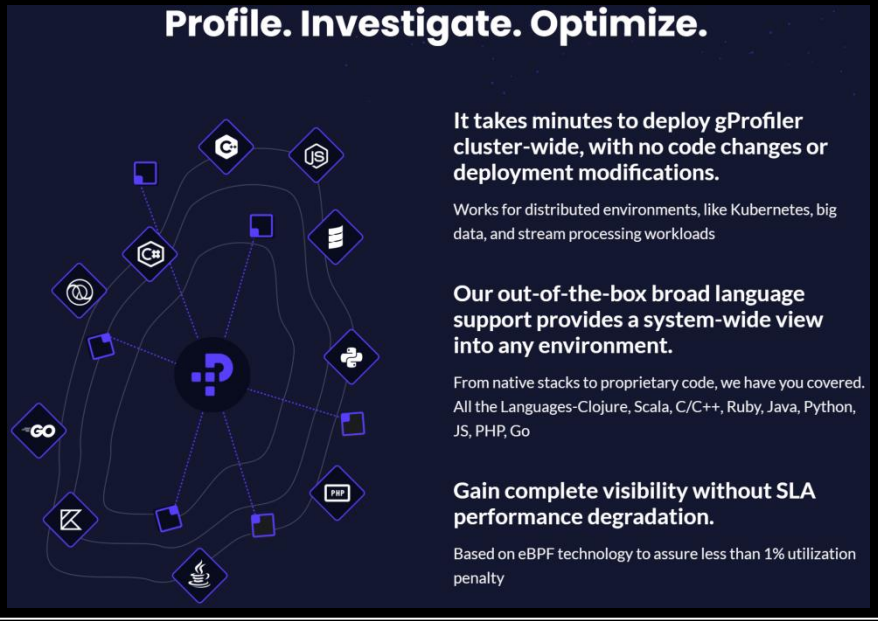
  An open-source, continuous profiler for production – across any environment, at any scale.

The must-have tool for performance and cost optimization

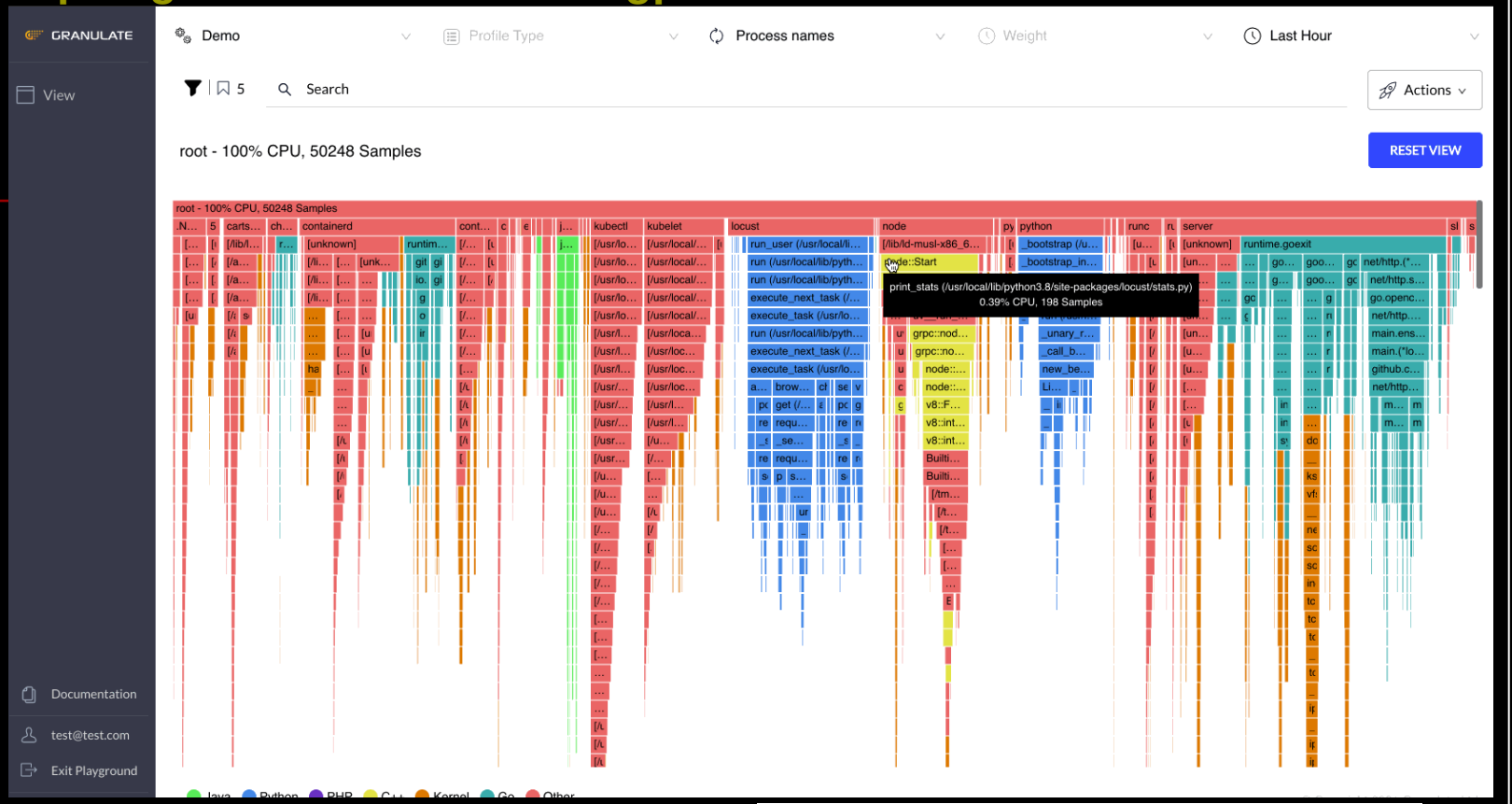
gProfiler enables any team to leverage cluster-wide profiling to investigate performance with minimal overhead.

By continuously analyzing code performance across your entire environment, you can optimize the most resource-consuming parts of your code, improve application performance, and reduce costs.

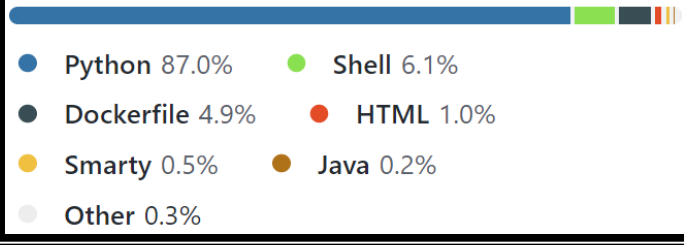
- Features



- https://github.com/Granulate/gprofiler



Languages



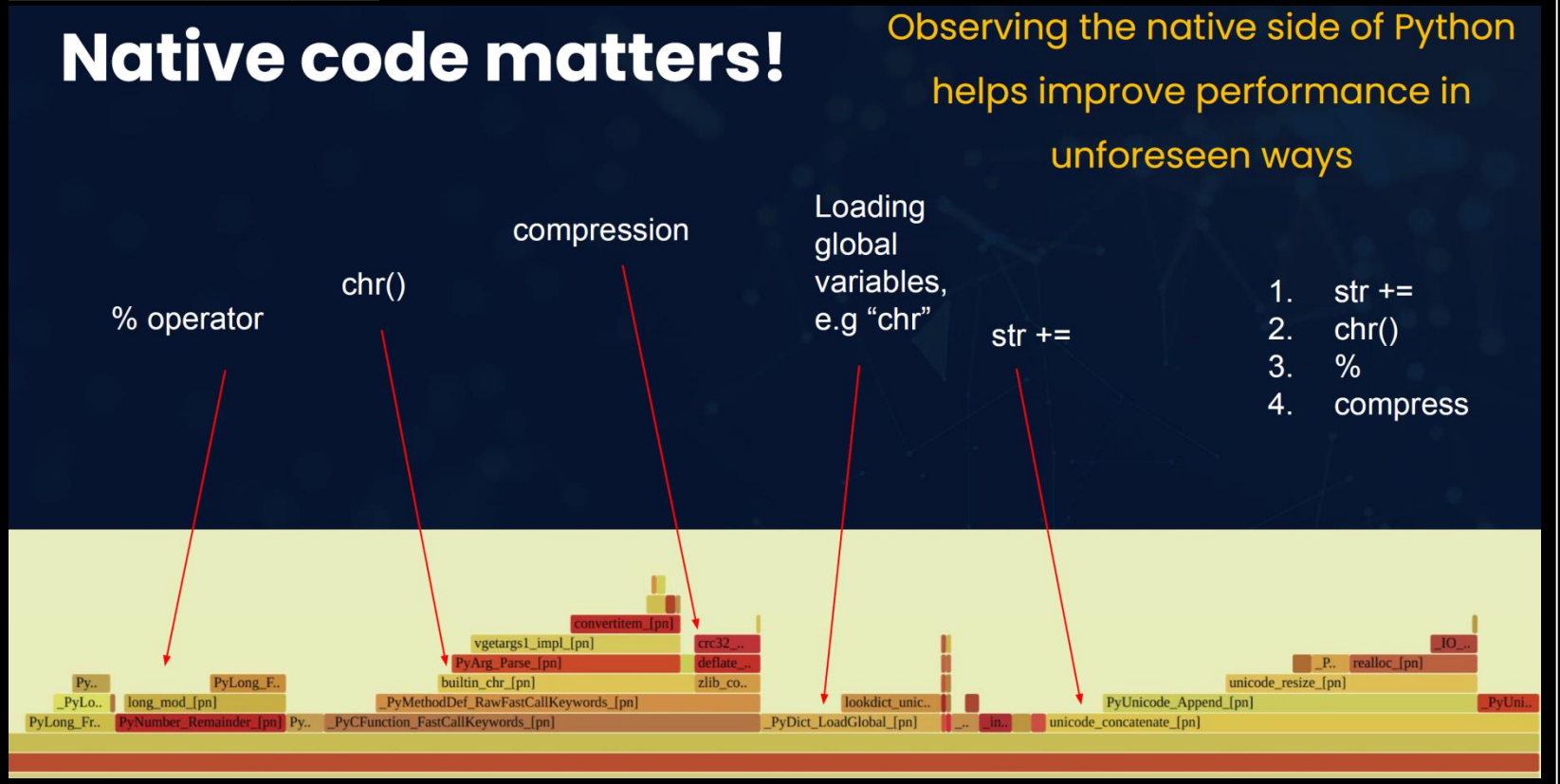
Python profiling options

- `--no-python` : Alias of `--python-mode disabled`.
- `--python-mode` : Controls which profiler is used for Python.
  - `auto` - (default) try with PyPerf (eBPF), fall back to py-spy.
  - `pyperf` - Use PyPerf with no py-spy fallback.
  - `pyspy` / `py-spy` - Use py-spy.
  - `disabled` - Disable profilers for Python.

Profiling using eBPF incurs lower overhead & provides kernel & native stacks.

Example

```python
import zlib

def func(n):
    buf = ""
    for i in range(n):
        buf += chr(i % 128)
    zlib.compress(buf.encode())

while True: func(999999)
```



Source: https://www.conf42.com/assets/slides/Conf42%20Python%202022%20Slides%20-%20Yonathan%20Goldschmidt.pdf

3.4 Miscs

Overview

- …

3.4.1 pyperf

- https://pyperf.readthedocs.io/

  A toolkit to write, run and analyze benchmarks.

  Features:

- Simple API to run reliable benchmarks: see examples.
- Automatically calibrate a benchmark for a time budget.
- Spawn multiple worker processes.
- Compute the mean and standard deviation.
- Detect if a benchmark result seems unstable: see the pyperf check command.
- pyperf stats command to analyze the distribution of benchmark results (min/max, mean, median, percentiles, etc.).
- pyperf compare_to command tests if a difference if significant. It supports comparison between multiple benchmark suites (made of multiple benchmarks)
- pyperf timeit command line tool for quick but reliable Python microbenchmarks
- pyperf system tune command to tune your system to run stable benchmarks.
- Automatically collect metadata on the computer and the benchmark: use the pyperf metadata command to display them, or the pyperf collect_metadata command to manually collect them.
- `--track-memory` and `--tracemalloc` options to track the memory usage of a benchmark.
- JSON format to store benchmark results.
- Support multiple units: seconds, bytes and integer.

- https://github.com/psf/pyperf
- Languages

- …

Example

- https://pyperf.readthedocs.io/en/latest/examples.html
- …

3.4.2 py-spy

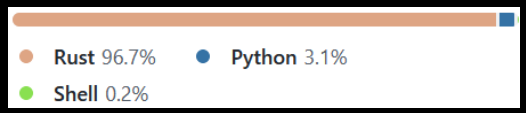
- https://github.com/benfred/py-spy

  Sampling profiler for Python programs.

  py-spy is a sampling profiler for Python programs. It lets you visualize what your Python program is spending time on without restarting the program or modifying the code in any way. py-spy is extremely low overhead: it is written in Rust for speed and doesn't run in the same process as the profiled Python program. This means py-spy is safe to use against production Python code.

  py-spy works on Linux, OSX, Windows and FreeBSD, and supports profiling all recent versions of the CPython interpreter (versions 2.3-2.7 and 3.3-3.10).

- **Languages**



- **FAQ**

  Why do we need another Python profiler?

  This project aims to let you profile and debug any running Python program, even if the program is serving production traffic.

  While there are many other python profiling projects, almost all of them require modifying the profiled program in some way. Usually, the profiling code runs inside of the target python process, which will slow down and change how the program operates. This means it's not generally safe to use these profilers for debugging issues in production services since they will usually have a noticeable impact on performance.

How does py-spy work?

py-spy works by directly reading the memory of the python program using the process_vm_readv system call on Linux, the vm_read call on OSX or the ReadProcessMemory call on Windows.

Figuring out the call stack of the Python program is done by looking at the global PyInterpreterState variable to get all the Python threads running in the interpreter, and then iterating over each PyFrameObject in each thread to get the call stack. Since the Python ABI changes between versions, we use rust's bindgen to generate different rust structures for each Python interpreter class we care about and use these generated structs to figure out the memory layout in the Python program.

Getting the memory address of the Python Interpreter can be a little tricky due to Address Space Layout Randomization. If the target python interpreter ships with symbols it is pretty easy to figure out the memory address of the interpreter by dereferencing the `interp_head` or `_PyRuntime` variables depending on the Python version. However, many Python versions are shipped with either stripped binaries or shipped without the corresponding PDB symbol files on Windows. In these cases we scan through the BSS section for addresses that look like they may point to a valid PyInterpreterState and check if the layout of that address is what we expect.

Can py-spy profile native extensions?

Yes! py-spy supports profiling native python extensions written in languages like C/C++ or Cython, on x86_64 Linux and Windows. You can enable this mode by passing `--native` on the command line. For best results, you should compile your Python extension with symbols. Also worth noting for Cython programs is that py-spy needs the generated C or C++ file in order to return line numbers of the original .pyx file. Read the blog post for more information.

...

Example

- https://github.com/benfred/py-spy/tree/master/examples
- …

3.5 Good Resources

- https://cython.readthedocs.io/en/latest/src/tutorial/profiling_tutorial.html
- …

4) Debugging

4.1 Overview

- https://

4.2 Built-in Print, Logging, Tracing

4.2.1 Print

Overview

- https://adamj.eu/tech/2021/10/08/tips-for-debugging-with-print/

1. Debug variables with f-strings and =

On Python 3.8+ we can use an f-string with the `=` specifier to achieve the same with less typing. This specifier prints the variable's name, "=", and the `repr()` of its value:

```
>>> print(f"{widget_count=}")
widget_count=9001
```

2. Make output "pop" with emoji

Make your debug statements stand out among other output with emoji:

```
print("👉 spam()")
```

You can also then jump to your debug output with your terminal's "find" function.

Here are some good emojis for debugging, which may also help express associated emotions:

- 👉 "Reached this point"
- ❌ "Failed as expected"
- ✅ "Worked randomly"
- ☒ "Trying hard"
- 🤡 "I feel like a software clown"
- 😱 "WTF"

To type emoji faster, use the keyboard shortcut for your OS:

- Windows: **Windows Key + .**
- macOS: **Control + Command + Space**
- Ubuntu: **Control + .**
- Other Linuxes: 🤷 consult documentation

3. Use `rich` or `pprint` for pretty printing

Rich is a terminal formatting library. It bundles many tools for prettifying terminal output and can be installed with `pip install rich`.

Rich's `print()` function is useful for debugging objects. It neatly indents large, nested data structures, and adds syntax highlighting:

```
>>> from rich import print as rprint
>>> rprint(luke)
{
    'id': 1,
    'name': 'Luke Skywalker',
    'films': [3, 4, 5, 6, 7, 8],
    'birth_year': '19BBY'
}
>>> |
```

https://rich.readthedocs.io/en/latest/introduction.html#quick-start

If you're not at liberty to install Rich, you can use Python's `pprint()` function instead. The extra "p" stands for "pretty". `pprint()` also indents data structures, albeit without colour or style:

```
>>> from pprint import pprint
>>> pprint(luke)
{'birth_year': '19BBY',
 'films': [3, 4, 5, 6, 7, 8],
 'id': 1,
 'name': 'Luke Skywalker'}
```

4. Use `locals()` to debug all local variables

Local variables are all the variables defined within the current function. We can grab all the local variables with the `locals()` builtin, which returns them in a dictionary. This is convenient for debugging several variables at once, even more so when combined with Rich or pprint.

We can use `locals()` like so:

```
from rich import print as rprint
```

```
def broken():
    numerator = 1
    denominator = 0
    rprint("👉", locals())
    return numerator / denominator
```

When we run this code, we see the dictionary of values before the exception:

```
>>> broken()
👉 {'numerator': 1, 'denominator': 0}
Traceback (most recent call last):
  ...
ZeroDivisionError: division by zero
```

There's also the `globals()` builtin which returns all the global variables, that is, those defined at the module scope, such as imported variables and classes. `globals()` is less useful for debugging since global variables don't usually change, but it is good to know about.

5. Use `vars()` to debug all of an object's attributes

The `vars()` builtin returns a dictionary of an object's attributes. This is useful when we want to debug many attributes at once:

```
>>> rprint(vars(widget))
{'id': 1, 'name': 'Batara-Widget'}
```

Brillo.

(`vars()` without an argument is also equivalent to `locals()`, in about half the typing.)

`vars()` works by accessing the `__dict__` attribute of the object. Most Python objects have this as the dictionary of their (writeable) attributes. We can also use `__dict__` directly, although it's a little more typing:

```
>>> rprint(widget.__dict__)
{'id': 1, 'name': 'Batara-Widget'}
```

`vars()` and `__dict__` don't work for *every* object. If an object's class uses `__slots__`, or it's built in C, then it won't have a `__dict__` attribute.

6. Debug your filename and line number to make returning there easy

Many terminals allow us to open on filenames from output. And text editors support opening files at a given line when a colon and the line number follows the filename. For example, this output would allow us to open `example.py`, line 12:

```
./example.py:12
```

We can use this capability in our `print()` calls to ease reopening the code we're trying to debug:

```
print(f"{__file__}:12 {widget_count=}")
```

This uses the Python magic variable `__file__` to get the name of the current file. We have to provide the line number ourselves.

Running this we see:

```
$ python example.py
/Users/me/project/example.py:12 widget_count=9001
```

✨Bonus✨ 7. Try icecream

The icecream package provides a handy debugging shortcut function, `ic()`. This function combines some of the tools we've been looking at. Called without arguments, `ic()` debugs details about where and when it was called:

```
from icecream import ic


def main():
    print("Starting main")
    ic()
    print("Finishing")


if __name__ == "__main__":
    main()
```

Called with arguments, `ic()` inspects and prints each expression (through source inspection) and its result:

```
from icecream import ic


def main():
    a = 1
    b = 2
    ic(a, b, a / b)


if __name__ == "__main__":
    main()
```

```
$ python example.py
ic| a: 1, b: 2, a / b: 0.5
```

This includes some syntax highlighting like Rich.

icecream also has some other handy abilities like installing as a builtin so you don't need to import it. Check out its documentation for more info.

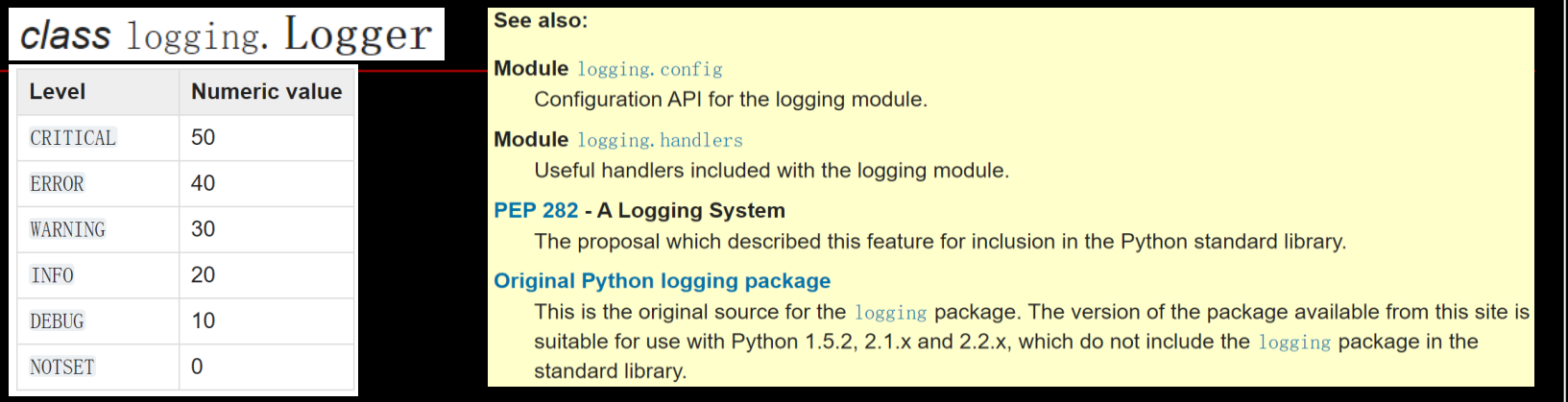
https://github.com/gruns/icecream

IceCream — Never use print() to debug again

4.2.2 Logging

4.2.2.1 Official

- https://docs.python.org/3/library/logging.html



class logging. Logger

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

See also:

Module logging.config
Configuration API for the logging module.

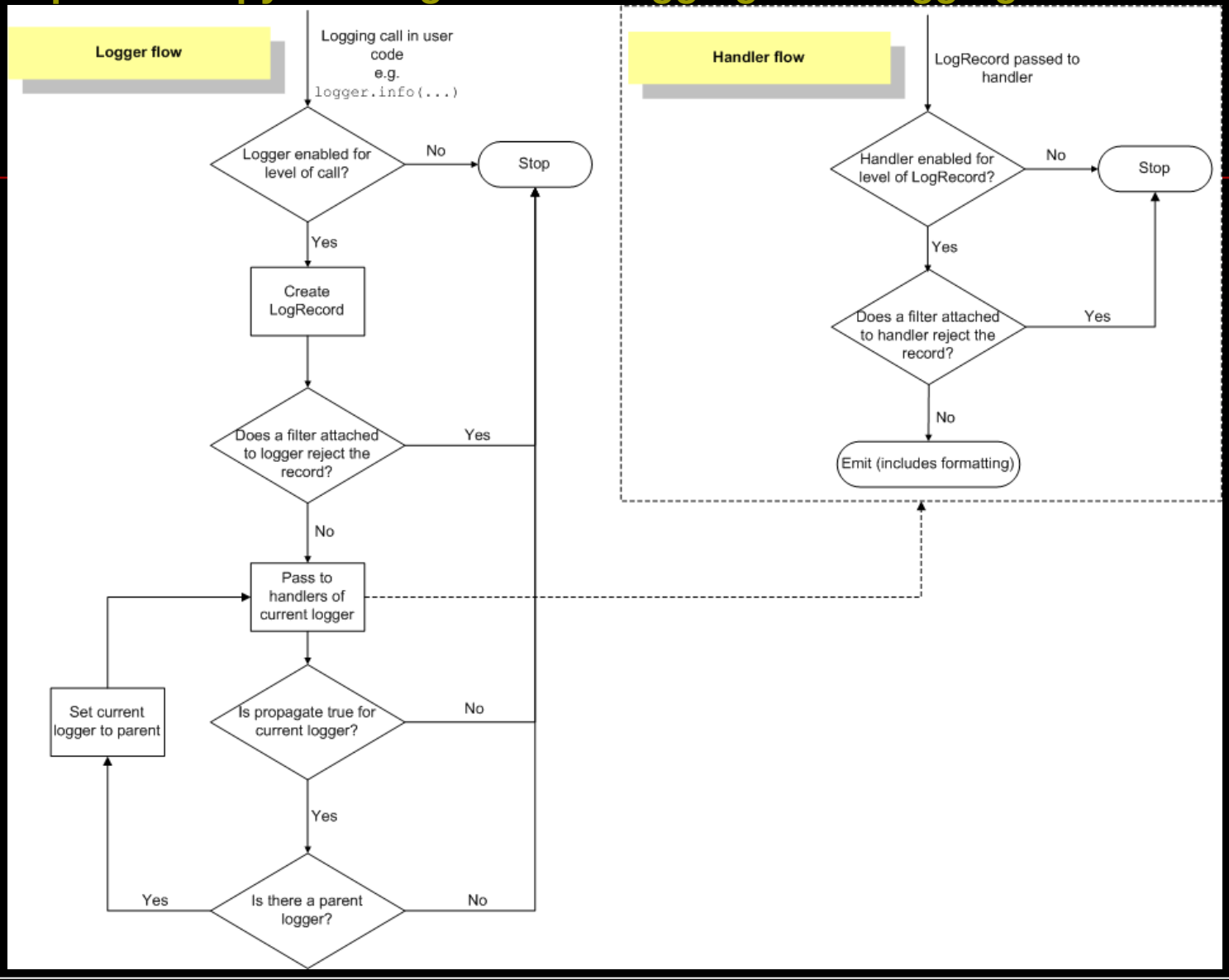
Module logging.handlers
Useful handlers included with the logging module.

PEP 282 - A Logging System
The proposal which described this feature for inclusion in the Python standard library.

Original Python logging package
This is the original source for the logging package. The version of the package available from this site is suitable for use with Python 1.5.2, 2.1.x and 2.2.x, which do not include the logging package in the standard library.

- https://docs.python.org/3/howto/logging.html#logging-basic-tutorial
- https://docs.python.org/3/howto/logging.html#logging-advanced-tutorial
- …
- https://realpython.com/python-logging/
- https://rollbar.com/blog/logging-in-python/
- https://machinelearningmastery.com/logging-in-python/
- …

- https://docs.python.org/3/howto/logging.html#logging-flow



Example

```
import logging
import auxiliary_module

# create logger with 'spam_application'
logger = logging.getLogger('spam_application')
logger.setLevel(logging.DEBUG)
# create file handler which logs even debug messages
fh = logging.FileHandler('spam.log')
fh.setLevel(logging.DEBUG)
# create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.ERROR)
# create formatter and add it to the handlers
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
ch.setFormatter(formatter)
# add the handlers to the logger
logger.addHandler(fh)
logger.addHandler(ch)

logger.info('creating an instance of auxiliary_module.Auxiliary')
a = auxiliary_module.Auxiliary()
logger.info('created an instance of auxiliary_module.Auxiliary')
logger.info('calling auxiliary_module.Auxiliary.do_something')
a.do_something()
logger.info('finished auxiliary_module.Auxiliary.do_something')
logger.info('calling auxiliary_module.some_function()')
auxiliary_module.some_function()
logger.info('done with auxiliary_module.some_function()')
```

Source: https://jupyterlite.readthedocs.io/en/latest/

4.2.2.2 Loguru

- https://github.com/Delgan/loguru

Python logging made (stupidly) simple.

Loguru is a library which aims to bring enjoyable logging in Python.

Did you ever feel lazy about configuring a logger and used `print()` instead?... I did, yet logging is fundamental to every application and eases the process of debugging. Using **Loguru** you have no excuse not to use logging from the start, this is as simple as `from loguru import logger`.

Also, this library is intended to make Python logging less painful by adding a bunch of useful functionalities that solve caveats of the standard loggers. Using logs in your application should be an automatism, **Loguru** tries to make it both pleasant and powerful.

```
> File "demo.py", line 21, in <module>
    main(1, 2, z)  # Oops...
    │        └ 0
    └ <function main at 0x7ff810e63bf8>

  File "demo.py", line 16, in main
    x * y / z
    │ │   └ 0
    │ └ 2
    └ 1

ZeroDivisionError: division by zero
>>> import sys
>>> logger.remove()
>>> logger.add(sys.stdout, format="[{time:HH:mm:ss}] <lvl>{message}</lvl>", level="INFO")
2
>>> logger.success("Logging using {style} formatting, finally!", style="braces")
[23:04:15] Logging using braces formatting, finally!
>>> logger.opt(colors=True).warning("Several available <blue>options</blue>")
[23:04:29] Several available options
>>> logger.opt(lazy=True).trace("Not executed: {}", lambda:
```

- Languages

- ...

4.2.3 Tracing

4.2.3.1 Official

- https://docs.python.org/3/library/trace.html

```
python -m trace --count -C . somefile.py ...
```

- https://docs.python.org/3/library/sys.html

sys.settrace(*tracefunc*)

Set the system's trace function, which allows you to implement a Python source code debugger in Python. The function is thread-specific; for a debugger to support multiple threads, it must register a trace function using `settrace()` for each thread being debugged or use `threading.settrace()`.

Trace functions should have three arguments: *frame*, *event*, and *arg*. *frame* is the current stack frame. *event* is a string: `'call'`, `'line'`, `'return'`, `'exception'` or `'opcode'`. *arg* depends on the event type.

The trace function is invoked (with *event* set to `'call'`) whenever a new local scope is entered; it should return a reference to a local trace function to be used for the new scope, or `None` if the scope shouldn't be traced.

- https://docs.python.org/3/library/functools.html

Higher-order functions and operations on callable objects.

- ...

Examples

```
# trace_example/main.py

from recurse import recurse


def main():
    print('This is the main program.')
    recurse(2)


if __name__ == '__main__':
    main()
```

ecurse() function invokes itself until the level argument reaches 0.

```
# trace_example/recurse.py


def recurse(level):
    print('recurse({})'.format(level))
    if level:
        recurse(level - 1)


def not_called():
    print('This function is never called.')
```

It is easy use `trace` directly from the command line. The statements being executed as the program runs are printed when the `--trace` option is given. This example also ignores the location of the Python standard library to avoid tracing into `importlib` and other modules that might be more interesting in another example, but that clutter up the output in this simple example.

```
$ python3 -m trace --ignore-dir=.../lib/python3.7 \
--trace trace_example/main.py

 --- modulename: main, funcname: <module>
main.py(7): """
main.py(10): from recurse import recurse
 --- modulename: recurse, funcname: <module>
recurse.py(7): """
recurse.py(11): def recurse(level):
recurse.py(17): def not_called():
main.py(13): def main():
main.py(18): if __name__ == '__main__':
main.py(19):     main()
 --- modulename: main, funcname: main
main.py(14):     print('This is the main program.')
This is the main program.
main.py(15):     recurse(2)
 --- modulename: recurse, funcname: recurse
recurse.py(12):     print('recurse({})'.format(level))
recurse(2)
recurse.py(13):     if level:
recurse.py(14):         recurse(level - 1)
 --- modulename: recurse, funcname: recurse
recurse.py(12):     print('recurse({})'.format(level))
recurse(1)
recurse.py(13):     if level:
recurse.py(14):         recurse(level - 1)
 --- modulename: recurse, funcname: recurse
recurse.py(12):     print('recurse({})'.format(level))
recurse(0)
recurse.py(13):     if level:
```

The first part of the output shows the setup operations performed by `trace`. The rest of the output shows the entry into each function, including the module where the function is located, and then the lines of the source file as they are executed. `recurse()` is entered three times, as expected based on the way it is called in `main()`.

Source: http://pymotw.com/3/trace/

A `call` event is generated before every function call. The frame passed to the callback can be used to find out which function is being called and from where.

```
# sys_settrace_call.py

#!/usr/bin/env python3
# encoding: utf-8

import sys


def trace_calls(frame, event, arg):
    if event != 'call':
        return
    co = frame.f_code
    func_name = co.co_name
    if func_name == 'write':
        # Ignore write() calls from printing
        return
    func_line_no = frame.f_lineno
    func_filename = co.co_filename
    if not func_filename.endswith('sys_settrace_call.py'):
        # Ignore calls not in this module
        return
    caller = frame.f_back
    caller_line_no = caller.f_lineno
    caller_filename = caller.f_code.co_filename
    print('* Call to', func_name)
    print('*  on line {} of {}'.format(
        func_line_no, func_filename))
    print('*  from line {} of {}'.format(
        caller_line_no, caller_filename))
    return


def b():
    print('inside b()\n')


def a():
    print('inside a()\n')
    b()


sys.settrace(trace_calls)
a()
```

This example ignores calls to `write()`, as used by `print` to write to `sys.stdout`.

```
$ python3 sys_settrace_call.py

* Call to a
*  on line 35 of sys_settrace_call.py
*  from line 41 of sys_settrace_call.py
inside a()

* Call to b
*  on line 31 of sys_settrace_call.py
*  from line 37 of sys_settrace_call.py
inside b()
```

Source: https://pymotw.com/3/sys/tracing.html

4.2.3.2 Hunter

- https://github.com/ionelmc/python-hunter

Hunter is a flexible code tracing toolkit, not for measuring coverage, but for debugging, logging, inspection and other nefarious purposes. It has a simple Python API, a convenient terminal API and a CLI tool to attach to processes.

- Languages

- https://python-hunter.readthedocs.io/en/latest/introduction.html

Example

-

```
import hunter
hunter.trace(module='posixpath', action=hunter.CallPrinter)

import os
os.path.join('a', 'b')
```

```
>>> os.path.join('a', 'b')
        /usr/lib/python3.6/posixpath.py:75    call      def join(a, *p):
        /usr/lib/python3.6/posixpath.py:80    line          a = os.fspath(a)
        /usr/lib/python3.6/posixpath.py:81    line          sep = _get_sep(a)
        /usr/lib/python3.6/posixpath.py:41    call      def _get_sep(path):
        /usr/lib/python3.6/posixpath.py:42    line          if isinstance(path, bytes):
        /usr/lib/python3.6/posixpath.py:45    line              return '/'
        /usr/lib/python3.6/posixpath.py:45    return            return '/'
                                              ...       return value: '/'
        /usr/lib/python3.6/posixpath.py:82    line          path = a
        /usr/lib/python3.6/posixpath.py:83    line          try:
        /usr/lib/python3.6/posixpath.py:84    line              if not p:
        /usr/lib/python3.6/posixpath.py:86    line              for b in map(os.fspath, p):
        /usr/lib/python3.6/posixpath.py:87    line                  if b.startswith(sep):
        /usr/lib/python3.6/posixpath.py:89    line                  elif not path or path.endswith(sep):
        /usr/lib/python3.6/posixpath.py:92    line                      path += sep + b
        /usr/lib/python3.6/posixpath.py:86    line              for b in map(os.fspath, p):
        /usr/lib/python3.6/posixpath.py:96    line          return path
        /usr/lib/python3.6/posixpath.py:96    return        return path
                                              ...       return value: 'a/b'
'a/b'
```

4.3 PDB series

4.3.1 Official PDB

- https://docs.python.org/3/library/pdb.html

The module `pdb` defines an interactive source code debugger for Python programs. It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame. It also supports post-mortem debugging and can be called under program control.

The debugger is extensible – it is actually defined as the class `Pdb`. This is currently undocumented but easily understood by reading the source. The extension interface uses the modules `bdb` and `cmd`.

The debugger's prompt is `(Pdb)`. Typical usage to run a program under control of the debugger is:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```

Changed in version 3.3: Tab-completion via the `readline` module is available for commands and command arguments, e.g. the current global and local names are offered as arguments of the `p` command.

`pdb.py` can also be invoked as a script to debug other scripts. For example:

```
python3 -m pdb myscript.py
```

When invoked as a script, pdb will automatically enter post-mortem debugging if the program being debugged exits abnormally. After post-mortem debugging (or after normal exit of the program), pdb will restart the program. Automatic restarting preserves pdb's state (such as breakpoints) and in most cases is more useful than quitting the debugger upon program's exit.

New in version 3.2: `pdb.py` now accepts a `-c` option that executes commands as if given in a `.pdbrc` file, see Debugger Commands.

New in version 3.7: `pdb.py` now accepts a `-m` option that execute modules similar to the way `python3 -m` does. As with a script, the debugger will pause execution just before the first line of the module.

The typical usage to break into the debugger is to insert:

```
import pdb; pdb.set_trace()
```

at the location you want to break into the debugger, and then run the program. You can then step through the code following this statement, and continue running without the debugger using the `continue` command.

New in version 3.7: The built-in `breakpoint()`, when called with defaults, can be used instead of `import pdb; pdb.set_trace()`.

...

4.3.2 IPDB

- https://github.com/gotcha/ipdb

ipdb exports functions to access the IPython debugger, which features tab completion, syntax highlighting, better tracebacks, better introspection with the same interface as the pdb module.

- https://ipython.readthedocs.io/en/stable/api/generated/IPython.core.debugger.html

This is an extension to PDB which adds a number of new features. Note that there is also the `IPython.terminal.debugger` class which provides UI improvements.

We also strongly recommend to use this via the `ipdb` package, which provides extra configuration options.

Among other things, this subclass of PDB:

- supports many IPython magics like pdef/psource
- hide frames in tracebacks based on `__tracebackhide__`
- allows to skip frames based on `__debuggerskip__`

The skipping and hiding frames are configurable via the `skip_predicates` command.

By default, frames from readonly files will be hidden, frames containing `__tracebackhide__=True` will be hidden.

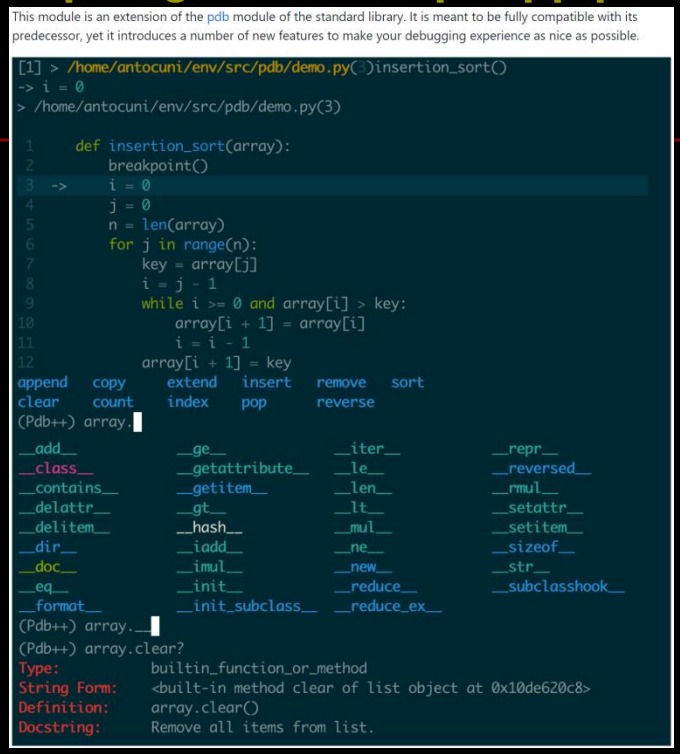
Frames containing `__debuggerskip__` will be stepped over, frames who's parent frames value of `__debuggerskip__` is `True` will be skipped.

$SRC_IPYTHON/core/debugger.py

- https://switowski.com/blog/ipython-debugging
- …

4.3.3 PDB++

- https://github.com/pdbpp/pdbpp



- Features

- colorful TAB completion of Python expressions (through fancycompleter)
- optional syntax highlighting of code listings (through Pygments)
- sticky mode
- several new commands to be used from the interactive `(Pdb++)` prompt
- smart command parsing (hint: have you ever typed `r` or `c` at the prompt to print the value of some variable?)
- additional convenience functions in the `pdb` module, to be used from your program

- ...

4.4 Snoop series

4.4.1 PySnooper

- https://github.com/cool-RR/pysnooper

About Never use print for debugging again

PySnooper is a poor man's debugger. If you've used Bash, it's like `set -x` for Python, except it's fancier.

Your story: You're trying to figure out why your Python code isn't doing what you think it should be doing. You'd love to use a full-fledged debugger with breakpoints and watches, but you can't be bothered to set one up right now.

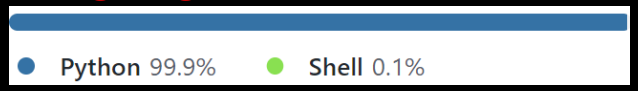
You want to know which lines are running and which aren't, and what the values of the local variables are.

Most people would use `print` lines, in strategic locations, some of them showing the values of variables.

PySnooper lets you do the same, except instead of carefully crafting the right `print` lines, you just add one decorator line to the function you're interested in. You'll get a play-by-play log of your function, including which lines ran and when, and exactly when local variables were changed.

What makes **PySnooper** stand out from all other code intelligence tools? You can use it in your shitty, sprawling enterprise codebase without having to do any setup. Just slap the decorator on, as shown below, and redirect the output to a dedicated log file by specifying its path as the first argument.

- Languages



- https://github.com/cool-RR/PySnooper/blob/master/ADVANCED_USAGE.md
- …

Examples

```python
import pysnooper

@pysnooper.snoop()
def number_to_bits(number):
    if number:
        bits = []
        while number:
            number, remainder = divmod(number, 2)
            bits.insert(0, remainder)
        return bits
    else:
        return [0]

number_to_bits(6)
```

The output to stderr is:

```
Source path:... /PySnooper/example.py
Starting var:.. number = 6
20:11:56.124936 call         6 def number_to_bits(number):
20:11:56.125577 line         7     if number:
20:11:56.125701 line         8         bits = []
New var:....... bits = []
20:11:56.125787 line         9         while number:
20:11:56.125956 line        10             number, remainder = divmod(number, 2)
Modified var:.. number = 3
New var:....... remainder = 0
20:11:56.126042 line        11             bits.insert(0, remainder)
Modified var:.. bits = [0]
20:11:56.126314 line         9         while number:
20:11:56.126494 line        10             number, remainder = divmod(number, 2)
Modified var:.. number = 1
Modified var:.. remainder = 1
20:11:56.126623 line        11             bits.insert(0, remainder)
Modified var:.. bits = [1, 0]
20:11:56.126870 line         9         while number:
20:11:56.127025 line        10             number, remainder = divmod(number, 2)
Modified var:.. number = 0
20:11:56.127161 line        11             bits.insert(0, remainder)
Modified var:.. bits = [1, 1, 0]
20:11:56.127404 line         9         while number:
20:11:56.127621 line        12         return bits
20:11:56.127738 return      12         return bits
Return value:.. [1, 1, 0]
Elapsed time: 00:00:00.003039
```

4.4.2 Snoop

- https://github.com/alexmojaki/snoop

snoop is a powerful set of Python debugging tools. It's primarily meant to be a more featureful and refined version of PySnooper. It also includes its own version of icecream and some other nifty stuff.

You're trying to figure out why your Python code isn't doing what you think it should be doing. You'd love to use a full-fledged debugger with breakpoints and watches, but you can't be bothered to set one up right now.

You want to know which lines are running and which aren't, and what the values of the local variables are.

Most people would use `print` lines, in strategic locations, some of them showing the values of variables.

snoop lets you do the same, except instead of carefully crafting the right `print` lines, you just add one decorator line to the function you're interested in. You'll get a play-by-play log of your function, including which lines ran and when, and exactly when local variables were changed.

- Highlights

- ...

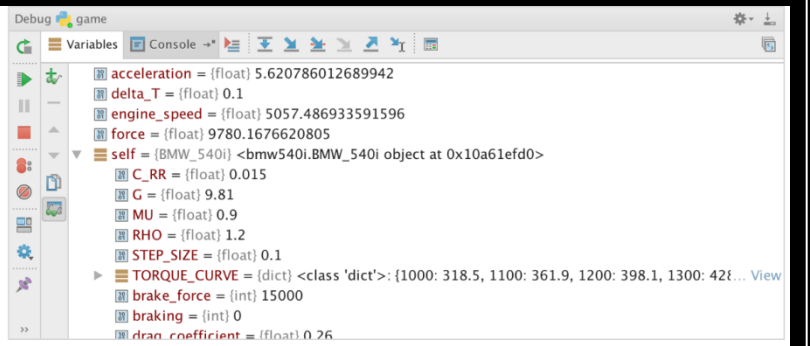
4.5 Debugger in IDEs

4.5.1 PyCharm

- https://www.jetbrains.com/pycharm/features/debugger.html

Visual Debugging

Some coders still debug using print statements, because the concept is hard and pdb is intimidating. PyCharm's python debugging GUI makes it easy to use a debugger by putting a visual face on the process. Getting started is simple and moving on to the major debugging features is easy.



Debug Everywhere

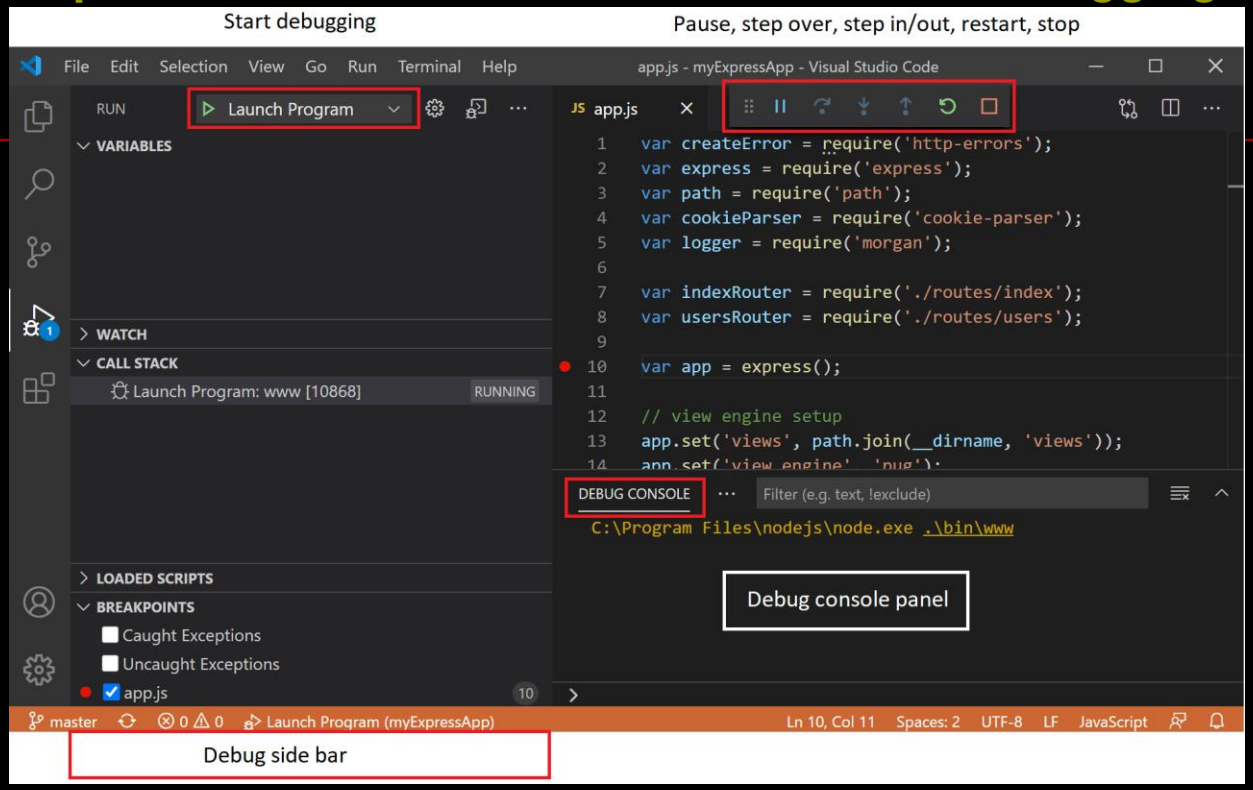
Of course PyCharm can debug code that you're running on your local computer, whether it's your system Python, a virtualenv, Anaconda, or a Conda env. In PyCharm Professional Edition you can also debug code you're running inside a Docker container, within a VM, or on a remote host through SSH.

…

- https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html#where-is-the-problem
- https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html
- https://betterprogramming.pub/mastering-the-pycharm-debugger-dd21e2333d51
- …

4.5.2 VSCode

- https://code.visualstudio.com/docs/editor/debugging



- https://code.visualstudio.com/docs/python/debugging
- https://devblogs.microsoft.com/python/python-in-visual-studio-code-may-2022-release/
- https://python.land/creating-python-programs/python-in-vscode
- …

4.6 Visualization

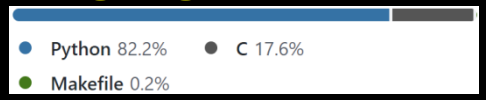
4.6.1 VizTracer

- https://github.com/gaogaotiantian/viztracer

VizTracer is a low-overhead logging/debugging/profiling tool that can trace and visualize your python code execution.

The front-end UI is powered by Perfetto. **Use "AWSD" to zoom/navigate**. More help can be found in "Support - Controls".

- Languages



- Highlights

- Detailed function entry/exit information on timeline with source code
- Super easy to use, no source code change for most features, no package dependency
- Supports threading, multiprocessing, subprocess and async
- Logs arbitrary function/variable using RegEx without code change
- Powerful front-end, able to render GB-level trace smoothly
- Works on Linux/MacOS/Windows

- https://viztracer.readthedocs.io/
- ...

4.7 Innovative

4.7.1 Cyberbrain

- https://github.com/laike9m/Cyberbrain

Python debugging, redefined.

Cyberbrain[1](电子脑) aims to free programmers from debugging. It lets you:

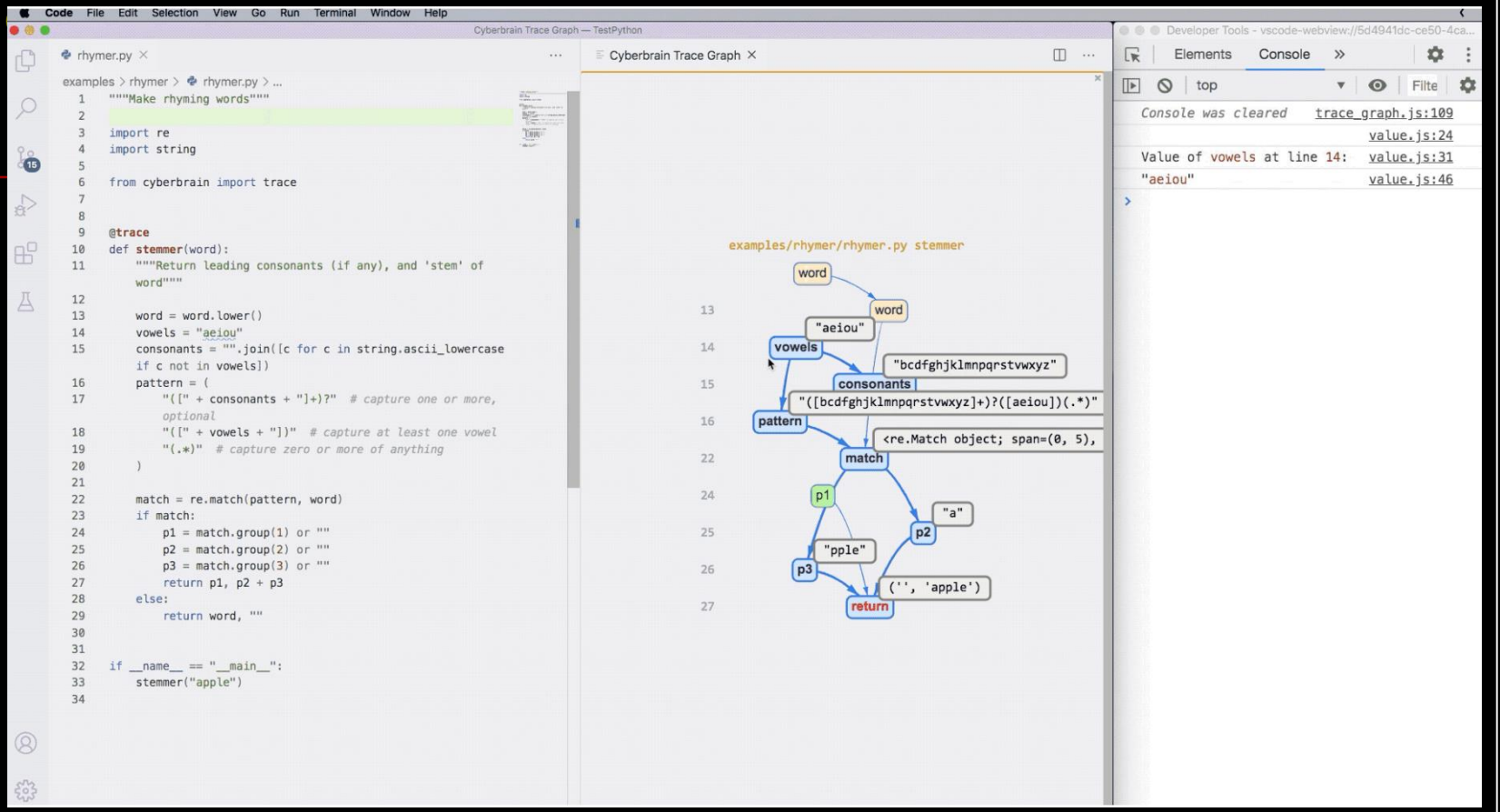
- **Backtrace variable changes**.
- See **every state** of program execution, including **variables' values**
- **Debug loops** with confidence.

Never spend hours stepping through a program, let Cyberbrain tell you what happened.

- https://github.com/laike9m/Cyberbrain/blob/master/docs/Features.md
- https://github.com/laike9m/Cyberbrain#roadmaps
- https://www.youtube.com/watch?v=eXlTVrNZ67Q
- Known limitations

- Overhead. See Lowering the overhead brought by Cyberbrain.
- Cyberbrain only traces the first call, no matter how many times the decorated function is called.
- You can only have one `@trace` decorator.
- No effort has been put on supporting threaded code (e.g. a multi-threaded web server), it may or may not work.
- Cyberbrain is not guaranteed to work if the program raises unhandled exceptions. You should first resolve or properly catch them.
- Cyberbrain can't display the content of certain objects, because they cannot be converted to JSON. In this case, Cyberbrain will show the repr string of that object. This should be rare.
- For generator functions, you need to manually call `trace.stop()`. see test_generator.py.
- `async` and multi-threading are not supported.

Workflow


Code File Edit Selection View Go Run Terminal Window Help
Cyberbrain Trace Graph — TestPython
rhymer.py
examples > rhymer > rhymer.py > ...
"""Make rhyming words"""
import re
import string
from cyberbrain import trace
@trace
def stemmer(word):
"""Return leading consonants (if any), and 'stem' of word"""
word = word.lower()
vowels = "aeiou"
consonants = "".join([c for c in string.ascii_lowercase if c not in vowels])
pattern = (
"([" + consonants + "]+)?" # capture one or more, optional
"([" + vowels + "])" # capture at least one vowel
"(.*)" # capture zero or more of anything
)
match = re.match(pattern, word)
if match:
p1 = match.group(1) or ""
p2 = match.group(2) or ""
p3 = match.group(3) or ""
return p1, p2 + p3
else:
return word, ""
if __name__ == "__main__":
stemmer("apple")
Cyberbrain Trace Graph
examples/rhymer/rhymer.py stemmer
word
word
"aeiou"
vowels
"bcdfghjklmnpqrstvwxyz"
consonants
"([bcdfghjklmnpqrstvwxyz]+)?([aeiou])(.*)"
pattern
<re.Match object; span=(0, 5),
match
p1
"a"
p2
"pple"
p3
('', 'apple')
return
Developer Tools - vscode-webview://5d4941dc-ce50-4ca...
Elements
Console
top
Filte
Console was cleared
trace_graph.js:109
value.js:24
Value of vowels at line 14:
value.js:31
"aeiou"
value.js:46

4.8 Good Resources

- https://realpython.com/python-debugging-pdb/
- https://pundit.pratt.duke.edu/wiki/Python:Debugging
- …

5) Notebook

Overview

- **https://ipython.org/**

IPython provides a rich architecture for interactive computing with:

- A powerful interactive shell.
- A kernel for Jupyter.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into your own projects.
- Easy to use, high performance tools for parallel computing.

```
In [1]: print('Hello IPython')
Hello IPython

In [2]: 21 * 2
Out[2]: 42

In [3]: def say_hello(name):
   ...:     print('Hello {name}'.format(name=name))
   ...:
```

```
In [1]: %timeit range(1000)
100000 loops, best of 3: 7.76 us per loop

In [2]: %%timeit x = range(10000)
...: max(x)
...:
1000 loops, best of 3: 223 us per loop
```

- **Jupyter Notebook(formerly known as the IPython Notebook)**

Jupyter and the future of IPython

IPython is a growing project, with increasingly language-agnostic components. IPython 3.x was the last monolithic release of IPython, containing the notebook server, qtconsole, etc. As of IPython 4.0, the language-agnostic parts of the project: the notebook format, message protocol, qtconsole, notebook web application, etc. have moved to new projects under the name Jupyter. IPython itself is focused on interactive Python, part of which is providing a Python kernel for Jupyter.

- **https://github.com/ipython**
- **https://ipython.readthedocs.io/**
- **...**

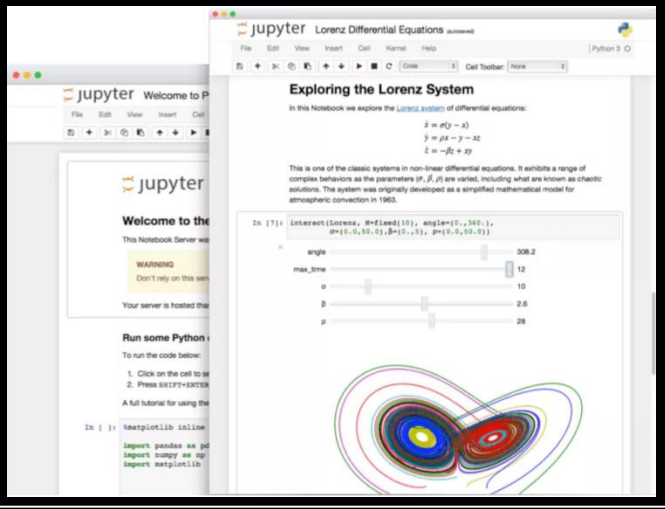
5.1 Jupyter

Overview

- **https://jupyter.org/**
  Free software, open standards, and web services for interactive computing across all programming languages.

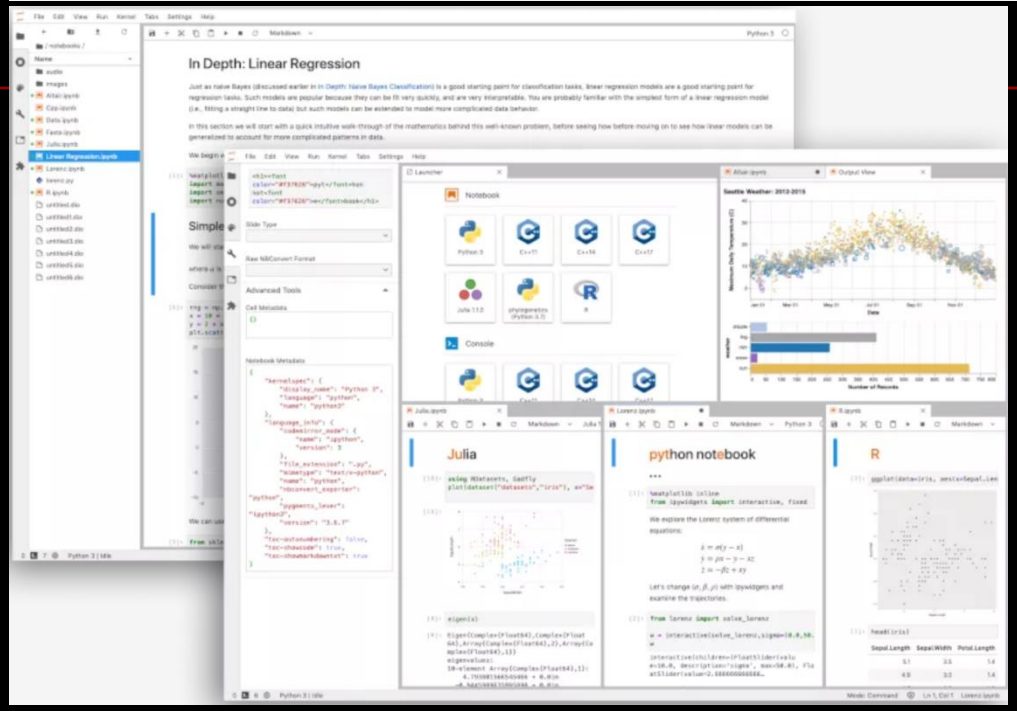
- **Jupyter Notebook: The Classic Notebook Interface**

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.



JupyterLab: A Next-Generation Notebook Interface

JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.



Jupyterhub

A multi-user version of the notebook designed for companies, classrooms and research labs

Pluggable authentication

Manage users and authentication with PAM, OAuth or integrate with your own directory service system.

Centralized deployment

Deploy the Jupyter Notebook to thousands of users in your organization on centralized infrastructure on- or off-site.

Container friendly

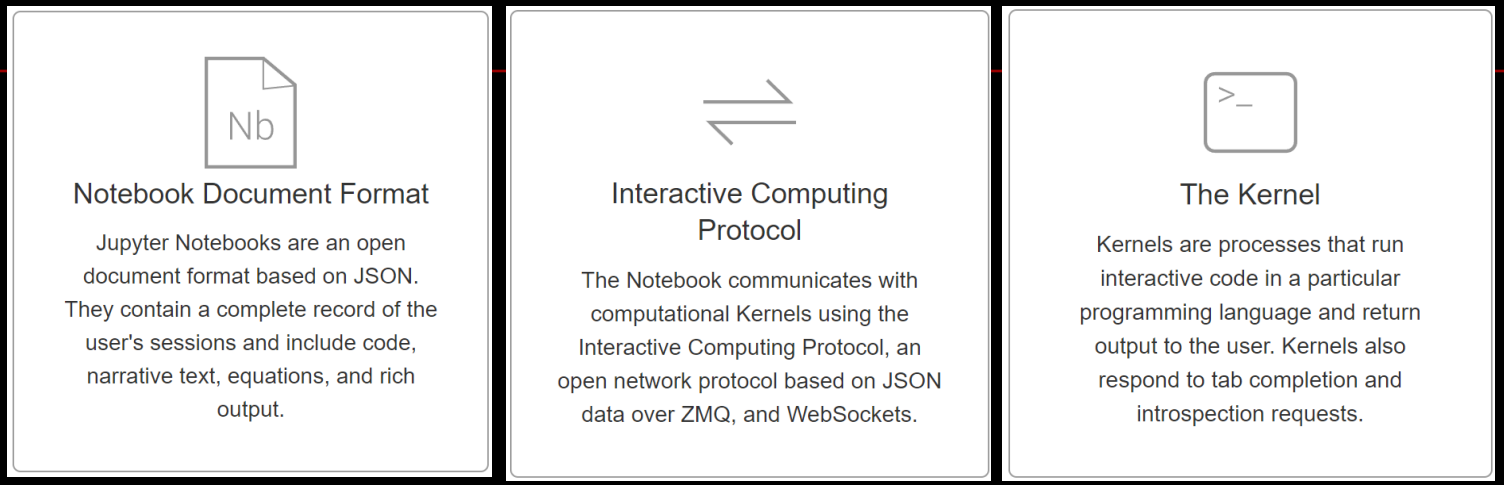
Use Docker and Kubernetes to scale your deployment, isolate user processes, and simplify software installation.

Code meets data

Deploy the Notebook next to your data to provide unified software management and data access within your organization.

- **Open Standards for Interactive Computing**

Project Jupyter promotes open standards that third-party developers can leverage to build customized applications. Think HTML and CSS for interactive computing on the web.



- **https://voila.readthedocs.io/en/stable/**

Voilà allows you to convert a Jupyter Notebook into an interactive dashboard that allows you to share your work with others. It is secure and customizable, giving you control over what your readers experience.

- **https://jupyter.org/widgets**
- **...**

5.1.1 JupyterLab

Overview

- **https://github.com/jupyterlab/jupyterlab**

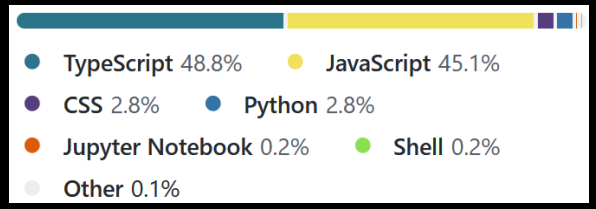
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. Currently ready for users.

JupyterLab is the next-generation user interface for Project Jupyter offering all the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible and powerful user interface. JupyterLab will eventually replace the classic Jupyter Notebook.

JupyterLab can be extended using npm packages that use our public APIs. The *prebuilt* extensions can be distributed via PyPI, conda, and other package managers. The *source* extensions can be installed directly from npm (search for jupyterlab-extension) but require additional build step. You can also find JupyterLab extensions exploring GitHub topic jupyterlab-extension. To learn more about extensions, see the user documentation.

The current JupyterLab releases are suitable for general usage, and the extension APIs will continue to evolve for JupyterLab extension developers.

- **Languages**



- **https://jupyterlab.readthedocs.io/**
- **…**

5.1.2 Kernels

Overview

- **https://github.com/jupyter/jupyter/wiki/Jupyter-kernels**

  Kernel Zero is IPython, which you can get through ipykernel, and is still a dependency of jupyter. The IPython kernel can be thought of as a reference implementation, as CPython is for Python.

- **https://jupyter-client.readthedocs.io/en/latest/kernels.html**

  A 'kernel' is a program that runs and introspects the user's code. IPython includes a kernel for Python code, and people have written kernels for several other languages.

  At kernel startup, Jupyter passes the kernel a connection file. This specifies how to set up communications with the frontend.

  There are three options for writing a kernel:

  1. You can reuse the IPython kernel machinery to handle the communications, and just describe how to execute your code. This is much simpler if the target language can be driven from Python. See Making simple Python wrapper kernels for details.
  2. You can implement the kernel machinery in your target language. This is more work initially, but the people using your kernel might be more likely to contribute to it if it's in the language they know.
  3. You can use the xeus library that is a C++ implementation of the Jupyter kernel protocol. Kernel authors only need to implement the language-specific logic in their implementation (execute code, auto-completion...). This is the simplest solution if your target language can be driven from C or C++: e.g. if it has a C-API like most scripting languages. Check out the xeus documentation for more details. Examples of kernels based on xeus include:

     - xeus-cling
     - xeus-python
     - JuniperKernel

- **https://github.com/dsblank/simple_kernel**
- **…**

5.1.2.1 Xeus

Overview

- **https://github.com/jupyter-xeus/xeus**

  A C++ implementation of the Jupyter kernel protocol.

  `xeus` is a library meant to facilitate the implementation of kernels for Jupyter. It takes the burden of implementing the Jupyter Kernel protocol so developers can focus on implementing the interpreter part of the kernel.

  Several Jupyter kernels are built upon xeus, such as `xeus-cling`, a kernel for the C++ programming language, and `xeus-python`, an alternative Python kernel for Jupyter.

- **Dependencies**

  `xeus` depends on the following libraries: `ZeroMQ`, `cppzmq`, `OpenSSL`, `nlohmann_json`, and `xtl`.

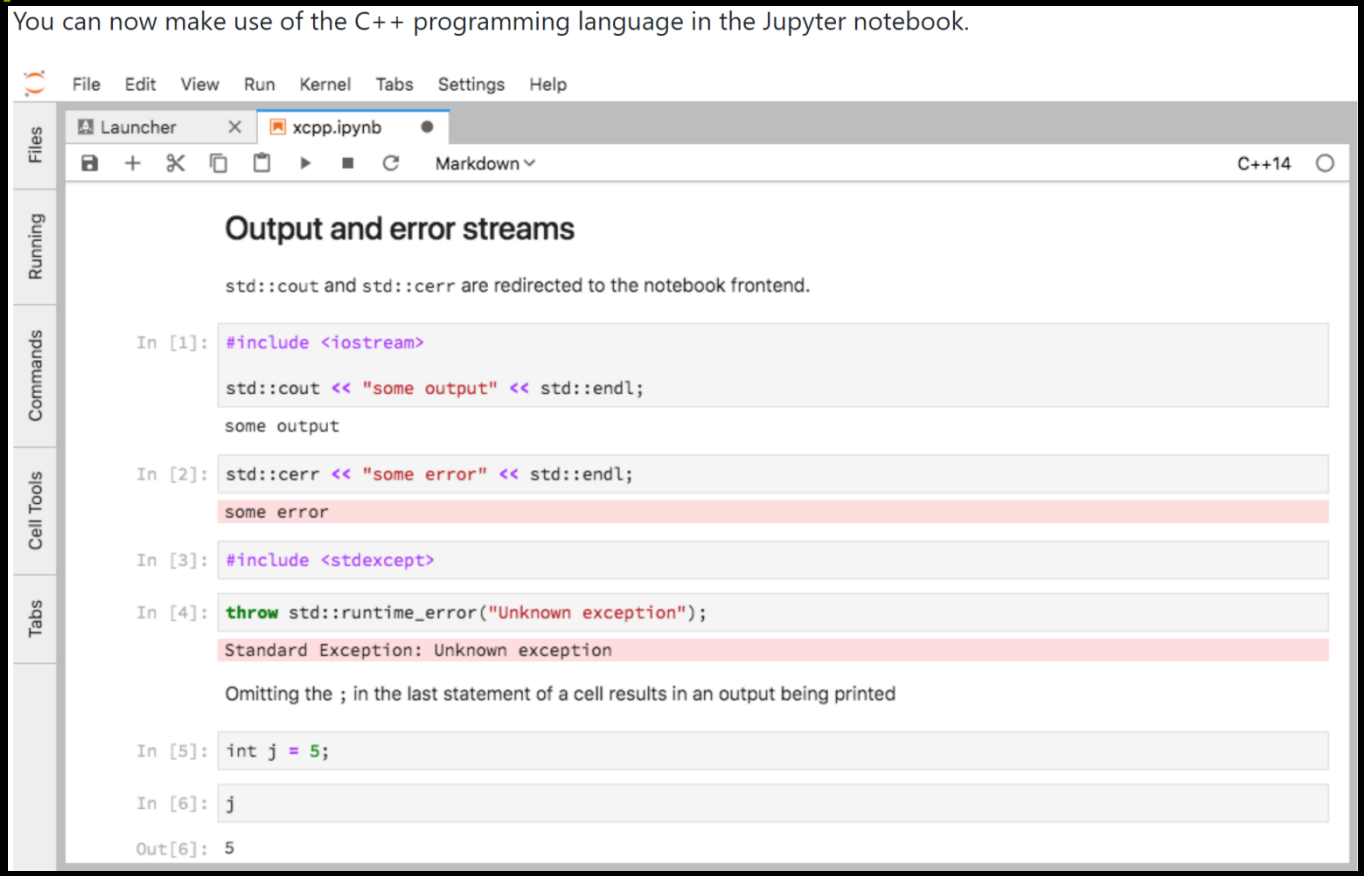
- **https://xeus.readthedocs.io/**

- Xeus-based kernels
  - xeus-cling
  - xeus-python
  - xeus-sql
  - xeus-sqlite
  - xeus-robot
  - xeus-lua

- …

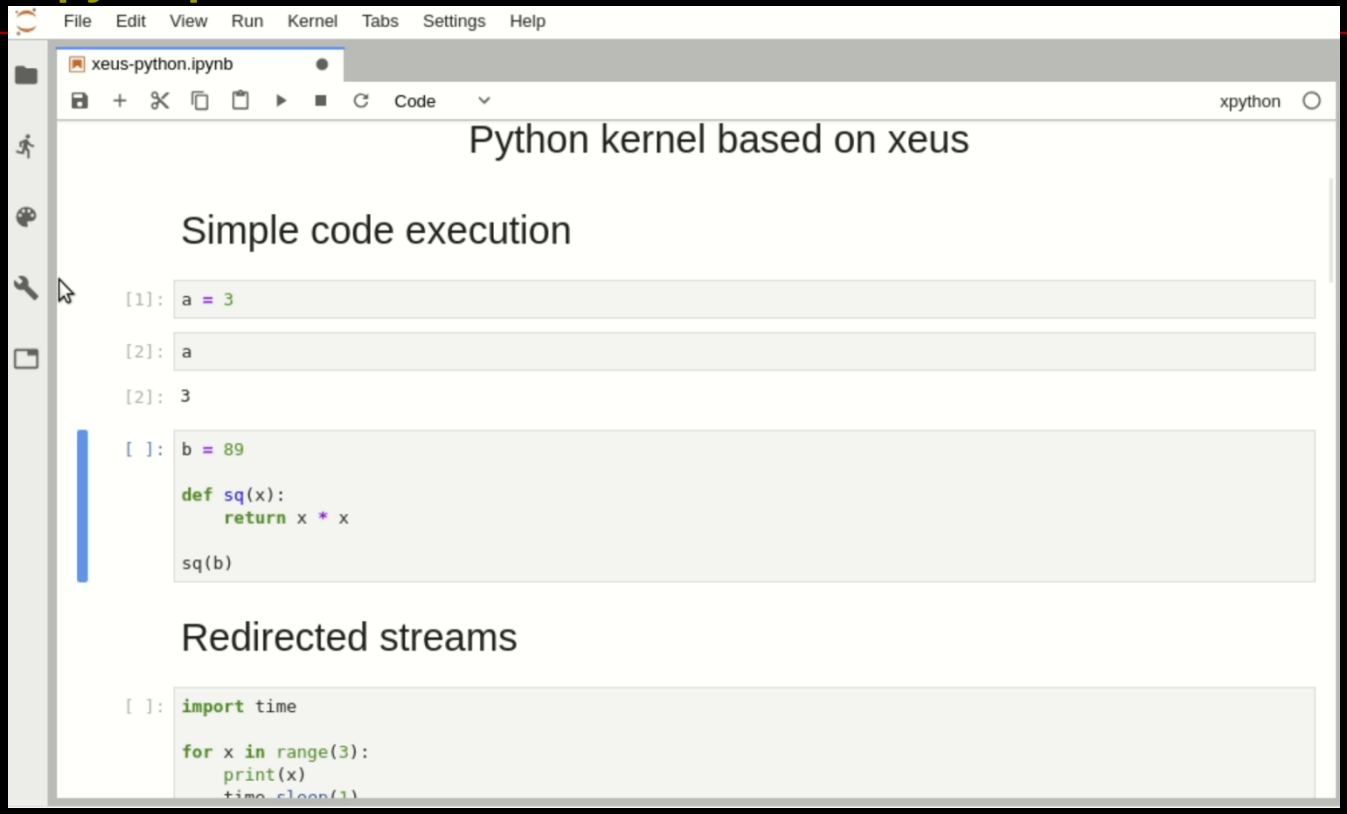
xeus-cling

- **https://github.com/jupyter-xeus/xeus-cling**
  A Jupyter kernel for C++ based on the C++ interpreter cling (https://root.cern/cling/) and the native implementation of the Jupyter protocol xeus.



xeus-python

- **https://github.com/jupyter-xeus/xeus-python**
  A Jupyter kernel for Python based on the native implementation of the Jupyter protocol xeus.



5.1.2.2 EvCxR

Overview

- https://github.com/google/evcxr

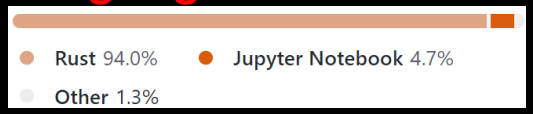
An evaluation context for Rust.

This project consists of several related crates.

- evcxr_jupyter - A Jupyter Kernel
- evcxr_repl - A Rust REPL
- evcxr - Common library shared by the above crates, may be useful for other purposes.
- evcxr_runtime - Functions and traits for interacting with Evcxr from libraries that users may use from Evcxr.

If you think you'd like a REPL, I'd definitely recommend checking out the Jupyter kernel. It's pretty much a REPL experience, but in a web browser.

- Languages



- https://github.com/google/evcxr/blob/main/evcxr_jupyter/README.md
- https://github.com/google/evcxr/blob/main/evcxr_repl/README.md
- https://github.com/google/evcxr/blob/main/evcxr/README.md
- https://github.com/google/evcxr/blob/main/evcxr_runtime/README.md
- https://crates.io/crates/evcxr_jupyter
- …

Example

Printing to outputs and evaluating expressions

Lets print something to stdout and stderr then return a final expression to see how that's presented. Note that stdout and stderr are separate streams, so may not appear in the same order is their respective print statements.

In [2]:
```
println!("Hello world");
eprintln!("Hello error");
format!("Hello {}", "world")
```

```
Hello error
Hello world
"Hello world"
```
Out[2]:

Assigning and making use of variables

We define a variable `message`, then in the subsequent cell, modify the string and finally print it out. We could also do all this in the one cell if we wanted.

In [3]:
```
let mut message = "Hello ".to_owned();
```

In [4]:
```
message.push_str("world!");
```

In [5]:
```
message
```

```
"Hello world!"
```
Out[5]:

Defining and redefining functions

Next we'll define a function

In [6]:
```
pub fn fib(x: i32) -> i32 {
    if x <= 2 {0} else {fib(x - 2) + fib(x - 1)}
}
```

In [7]:
```
(1..13).map(fib).collect::<Vec<i32>>()
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```
Out[7]:

Hmm, that doesn't look right. Lets redefine the function. In practice, we'd go back and edit the function above and reevalute it, but here, lets redefine it in a separate cell.

In [8]:
```
pub fn fib(x: i32) -> i32 {
    if x <= 2 {1} else {fib(x - 2) + fib(x - 1)}
}
```

In [9]:
```
let values = (1..13).map(fib).collect::<Vec<i32>>();
values
```

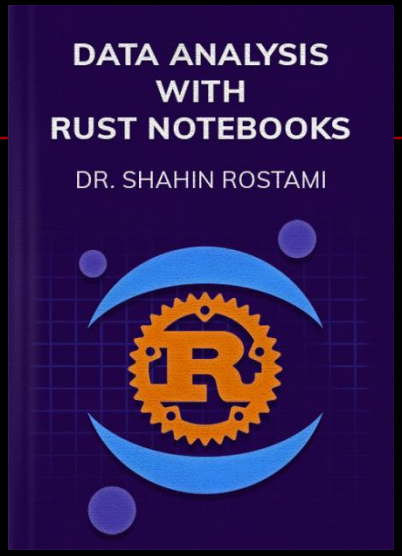
```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```
Out[9]:

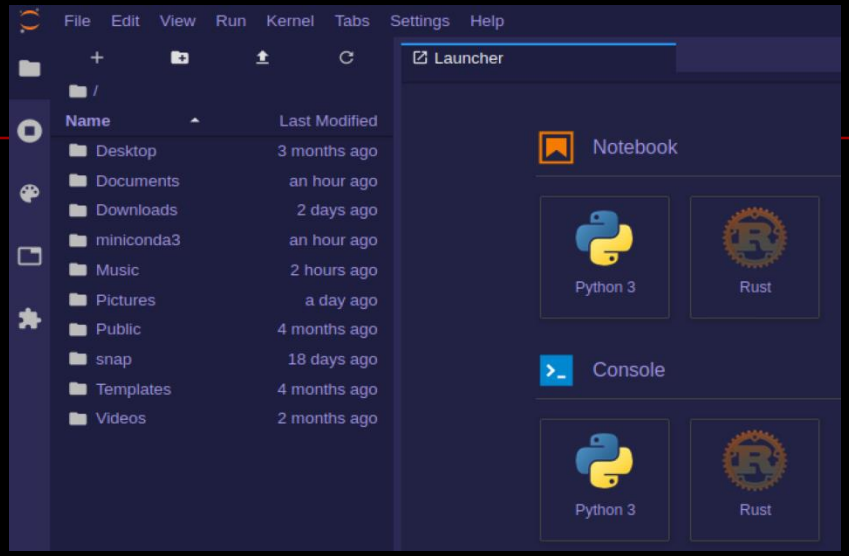
...

Source: https://github.com/google/evcxr/blob/main/evcxr_jupyter/samples/evcxr_jupyter_tour.ipynb

Good Resources

- **https://datacrayon.com/shop/product/data-analysis-with-rust-notebooks**





5.2 Lightweight

5.2.1 JupyterLite

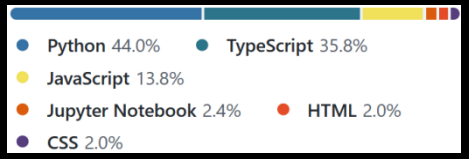
Overview

- https://github.com/jupyterlite/jupyterlite

Wasm powered Jupyter running in the browser.

JupyterLite is a JupyterLab distribution that **runs entirely in the browser** built from the ground-up using JupyterLab components and extensions.

- **Languages**

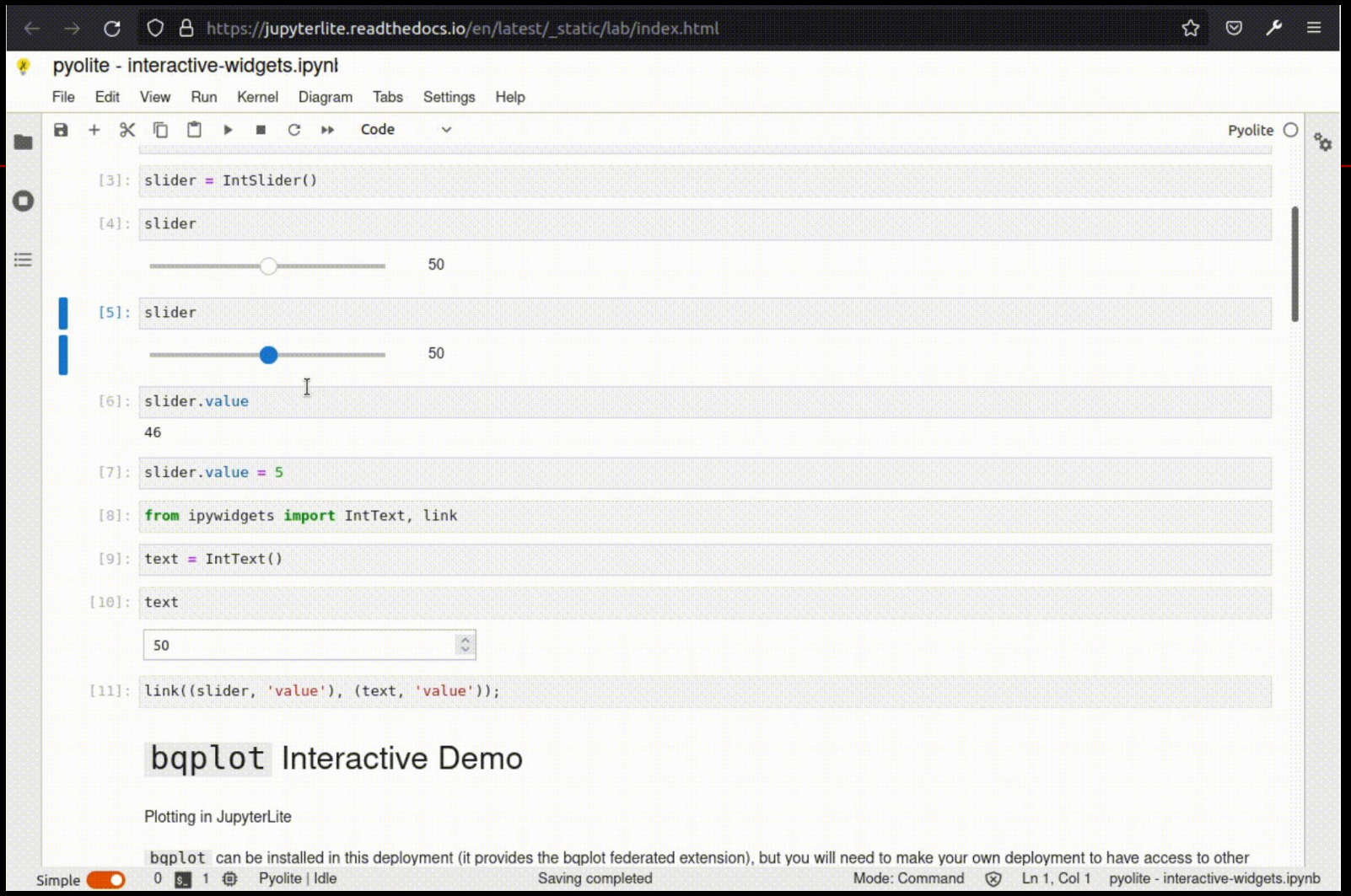


- **Browser-based Interactive Computing**

  - Python kernel backed by Pyodide running in a Web Worker
    - Initial support for interactive visualization libraries such as `altair`, `bqplot`, `ipywidgets`, `matplotlib`, and `plotly`
  - JavaScript and P5.js kernels running in an `IFrame`
  - View hosted example Notebooks and other files, then edit, save, and download from the browser's `IndexDB` (or `localStorage`)
  - Support for saving settings for JupyterLab/Lite core and federated extensions
  - Basic session and kernel management to have multiple kernels running at the same time
  - Support for Code Consoles

- https://jupyterlite.readthedocs.io
- …

Example



Source: https://jupyterlite.readthedocs.io/en/latest/

5.3 Polyglot

5.3.1 PolyNote

Overview

- **https://polynote.org/**

  Polynote is an experimental polyglot notebook environment. Currently, it supports Scala and Python(with or without Spark), SQL, and Vega.

- **Languages**

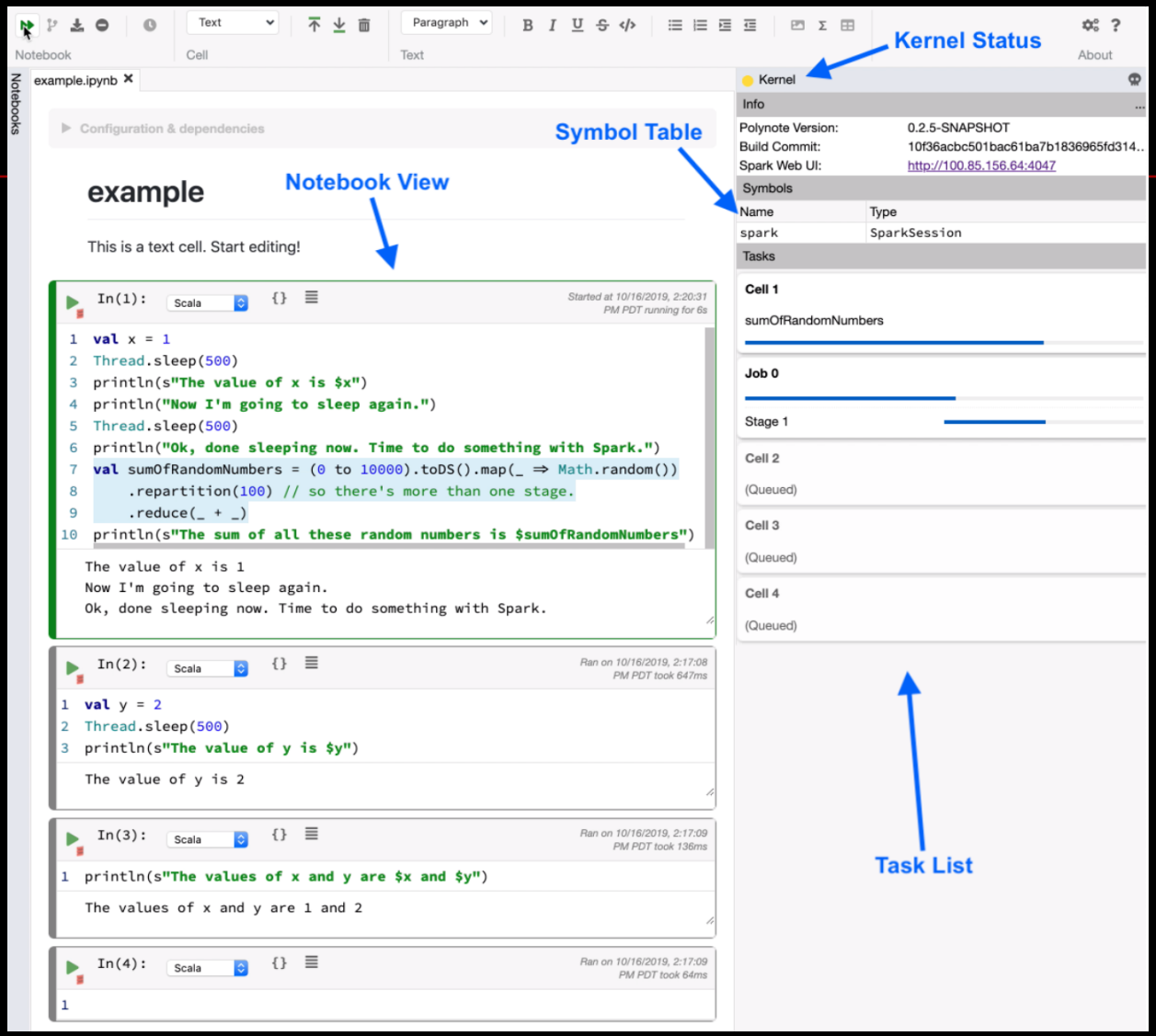
- **Why is it**

Current notebook solutions, like Jupyter and Zeppelin, are lacking in some fundamental features:

- *Code editing* – the code editing capabilities in most notebook tools leave plenty to be desired. Why can't a notebook tool have modern editing capabilities like those you'd find in an IDE? Polynote provides useful autocomplete, parameter hints, and more – we're planning to add even more features, like jump-to-definition.
- *Text editing* – you can use the WYSIWYG editor for composing text cells, so you'll know what the text will look like as you're writing. TeX equations are also supported.
- *Multi-language support* – Polynote allows you to mix multiple languages in one notebook, while sharing definitions seamlessly between them.
- *Runtime insight* – Polynote tries to keep you informed of what's going on at runtime:
  - The tasks area shows you what the kernel is doing at any given time.
  - The symbol table shows you what variables and functions you've defined, so you don't have to scroll around to remind yourself.
  - Compile failures and runtime exceptions are highlighted in the editor (for supported languages), so you can see exactly what's going wrong.

- **...**

Example



Source: https://netflixtechblog.com/open-sourcing-polynote-an-ide-inspired-polyglot-notebook-7f929d3f447

6) Serverless

Serverless

- https://en.wikipedia.org/wiki/Serverless_computing

Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand, taking care of the servers on behalf of their customers. "Serverless" is a misnomer in the sense that servers are still used by cloud service providers to execute code for developers. However, developers of serverless applications are not concerned with capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers. Serverless computing does not hold resources in volatile memory; computing is rather done in short bursts with the results persisted to storage. When an app is not in use, there are no computing resources allocated to the app. Pricing is based on the actual amount of resources consumed by an application.[1] It can be a form of utility computing.

Serverless computing can simplify the process of deploying code into production. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices or monoliths. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.[2] This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P).

Serverless runtimes [edit]

Serverless vendors offer compute runtimes, also known as Function as a Service (FaaS) platforms, which execute application logic but do not store data. Common languages supported by serverless runtimes are Java, Python and PHP. Generally, the functions run under isolation boundaries, such as, Linux containers.

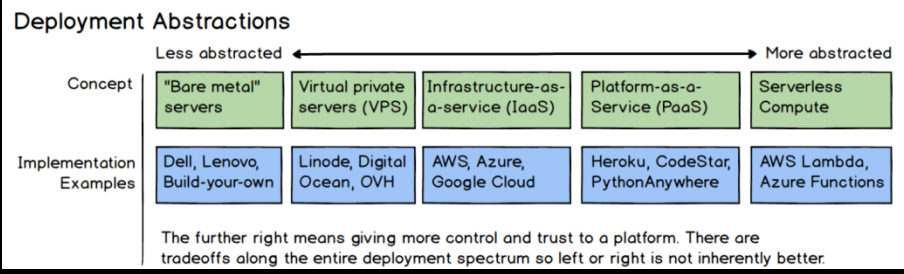
- https://www.fullstackpython.com/serverless.html

Serverless

Serverless is a deployment architecture where servers are not explicitly provisioned by the deployer. Code is instead executed based on developer-defined events that are triggered, for example when an HTTP POST request is sent to an API a new line written to a file.

How can code be executed "without" servers?

Servers still exist to execute the code but they are abstracted away from the developer and handled by a compute platform such as Amazon Web Services Lambda or Google Cloud Functions.



- ...

6.1 Chalice

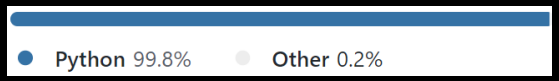
Overview

- https://github.com/aws/chalice

Chalice is a framework for writing serverless apps in python. It allows you to quickly create and deploy applications that use AWS Lambda. It provides:

- A command line tool for creating, deploying, and managing your app
- A decorator based API for integrating with Amazon API Gateway, Amazon S3, Amazon SNS, Amazon SQS, and other AWS services.
- Automatic IAM policy generation

- Languages



- https://aws.github.io/chalice/tutorials/index.html
- https://aws.github.io/chalice/topics/index.html
- https://aws.github.io/chalice/api.html
- https://realpython.com/aws-chalice-serverless-python/
- …

Example

- https://github.com/aws/chalice

You can create Rest APIs:

```
from chalice import Chalice

app = Chalice(app_name="helloworld")

@app.route("/")
def index():
    return {"hello": "world"}
```

Tasks that run on a periodic basis:

```
from chalice import Chalice, Rate

app = Chalice(app_name="helloworld")

# Automatically runs every 5 minutes
@app.schedule(Rate(5, unit=Rate.MINUTES))
def periodic_task(event):
    return {"hello": "world"}
```

You can connect a lambda function to an S3 event:

```
from chalice import Chalice

app = Chalice(app_name="helloworld")

# Whenever an object is uploaded to 'mybucket'
# this lambda function will be invoked.

@app.on_s3_event(bucket='mybucket')
def handler(event):
    print("Object uploaded for bucket: %s, key: %s"
          % (event.bucket, event.key))
```

As well as an SQS queue:

```
from chalice import Chalice

app = Chalice(app_name="helloworld")

# Invoke this lambda function whenever a message
# is sent to the ``my-queue-name`` SQS queue.

@app.on_sqs_message(queue='my-queue-name')
def handler(event):
    for record in event:
        print("Message body: %s" % record.body)
```

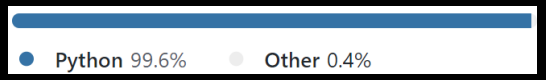
6.2 Zappa

Overview

- https://github.com/zappa/Zappa

Zappa makes it super easy to build and deploy server-less, event-driven Python applications (including, but not limited to, WSGI web apps) on AWS Lambda + API Gateway. Think of it as "serverless" web hosting for your Python apps. That means **infinite scaling**, **zero downtime**, **zero maintenance** - and at a fraction of the cost of your current deployments!

- Languages



- Related Projects

- Mackenzie - AWS Lambda Infection Toolkit
- NoDB - A simple, server-less, Pythonic object store based on S3.
- zappa-cms - A tiny server-less CMS for busy hackers. Work in progress.
- zappa-django-utils - Utility commands to help Django deployments.
- flask-ask - A framework for building Amazon Alexa applications. Uses Zappa for deployments.
- zappa-file-widget - A Django plugin for supporting binary file uploads in Django on Zappa.
- zops - Utilities for teams and continuous integrations using Zappa.
- cookiecutter-mobile-backend - A `cookiecutter` Django project with Zappa and S3 uploads support.
- zappa-examples - Flask, Django, image uploads, and more!
- zappa-hug-example - Example of a Hug application using Zappa.
- Zappa Docker Image - A Docker image for running Zappa locally, based on Lambda Docker.
- zappa-dashing - Monitor your AWS environment (health/metrics) with Zappa and CloudWatch.
- s3env - Manipulate a remote Zappa environment variable key/value JSON object file in an S3 bucket through the CLI.
- zappa_resize_image_on_fly - Resize images on the fly using Flask, Zappa, Pillow, and OpenCV-python.
- zappa-ffmpeg - Run ffmpeg inside a lambda for serverless transformations.
- gdrive-lambda - pass json data to a csv file for end users who use Gdrive across the organization.
- travis-build-repeat - Repeat TravisCI builds to avoid stale test results.
- wunderskill-alexa-skill - An Alexa skill for adding to a Wunderlist.
- xrayvision - Utilities and wrappers for using AWS X-Ray with Zappa.
- terraform-aws-zappa - Terraform modules for creating a VPC, RDS instance, ElastiCache Redis and CloudFront Distribution for use with Zappa.
- zappa-sentry - Integration with Zappa and Sentry
- IOpipe - Monitor, profile and analyze your Zappa apps.

- ...

Example

- https://github.com/zappa/Zappa

```
(env)~/demo $ cat my_app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, from Zappa!\n'

if __name__ == '__main__':
    app.run()
(env)~/demo $ cat zappa_settings.json
{
        "dev": {
                "s3_bucket": "lmbda",
                "app_function": "my_app.app",
                "parameter_depth": 1
        }
}
(env)~/demo $ zappa deploy dev
Packaging project as zip...
Uploading zip (5.8MiB)...
Creating API Gateway routes..
96it [00:06,  6.15it/s]
Deploying API Gateway..
Your Zappa deployment is live!: https://m8atxlc1j9.execute-api.us-east-1.amazonaws.com/dev
(env)~/demo $ curl -l https://m8atxlc1j9.execute-api.us-east-1.amazonaws.com/dev
Hello, from Zappa!
(env)~/demo $
```

7) Interoperability

Overview

- https://
- ...

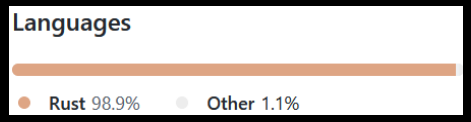
7.1 Rust

7.1.1 PyO3

- https://pyo3.rs/

  Rust bindings for Python, including tools for creating native Python extension modules. Running and interacting with Python code from a Rust binary is also supported.

- https://github.com/PyO3/pyo3



PyO3 supports the following software versions:

- Python 3.7 and up (CPython and PyPy)
- Rust 1.48 and up

You can use PyO3 to write a native Python module in Rust, or to embed Python in a Rust binary. The following sections explain each of these in turn.

- …

Example

- **Using Rust from Python**

PyO3 can be used to generate a native Python module. The easiest way to try this out for the first time is to use `maturin`. `maturin` is a tool for building and publishing Rust-based Python packages with minimal configuration. The following steps install `maturin`, use it to generate and build a new Python package, and then launch Python to import and execute a function from the package.

First, follow the commands below to create a new directory containing a new Python `virtualenv`, and install `maturin` into the virtualenv using Python's package manager, `pip`:

```
# (replace string_sum with the desired package name)
$ mkdir string_sum
$ cd string_sum
$ python -m venv .env
$ source .env/bin/activate
$ pip install maturin
```

Still inside this `string_sum` directory, now run `maturin init`. This will generate the new package source. When given the choice of bindings to use, select pyo3 bindings:

```
$ maturin init
✓ 🤷 What kind of bindings to use? · pyo3
  🎤 Done! New project created string_sum
```

...

Finally, run `maturin develop`. This will build the package and install it into the Python virtualenv previously created and activated. The package is then ready to be used from `python`:

```
$ maturin develop
# lots of progress output as maturin runs the compilation...
$ python
>>> import string_sum
>>> string_sum.sum_as_string(5, 20)
'25'
```

To make changes to the package, just edit the Rust source code and then re-run `maturin develop` to recompile.

To run this all as a single copy-and-paste, use the bash script below (replace `string_sum` in the first command with the desired package name):

```
mkdir string_sum && cd "$_"
python -m venv .env
source .env/bin/activate
pip install maturin
maturin init --bindings pyo3
maturin develop
```

As well as with `maturin`, it is possible to build using `setuptools-rust` or manually. Both offer more flexibility than `maturin` but require more configuration to get started.

Using Python from Rust

To embed Python into a Rust binary, you need to ensure that your Python installation contains a shared library. The following steps demonstrate how to ensure this (for Ubuntu), and then give some example code which runs an embedded Python interpreter.

To install the Python shared library on Ubuntu:

```
sudo apt install python3-dev
```

Start a new project with `cargo new` and add `pyo3` to the `Cargo.toml` like this:

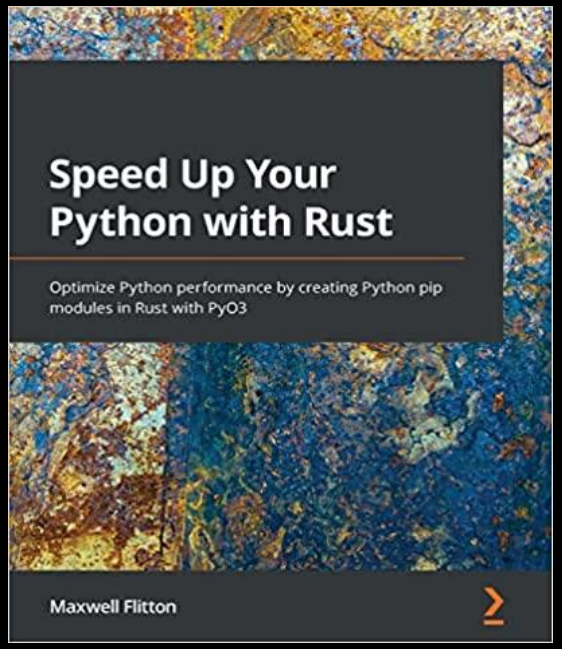
```
[dependencies.pyo3]
version = "0.16.5"
features = ["auto-initialize"]
```

Example program displaying the value of `sys.version` and the current user name:

```rust
use pyo3::prelude::*;
use pyo3::types::IntoPyDict;

fn main() -> PyResult<()> {
    Python::with_gil(|py| {
        let sys = py.import("sys")?;
        let version: String = sys.getattr("version")?.extract()?;

        let locals = [("os", py.import("os")?)].into_py_dict(py);
        let code = "os.getenv('USER') or os.getenv('USERNAME') or 'Unknown'";
        let user: String = py.eval(code, None, Some(&locals))?.extract()?;

        println!("Hello {}, I'm Python {}", user, version);
        Ok(())
    })
}
```

https://pyo3.rs/latest/python_from_rust.html

Good Resources

- https://github.com/ruwanego/RustPython-intro
- https://towardsdatascience.com/nine-rules-for-writing-python-extensions-in-rust-d35ea3a4ec29
- https://pythonspeed.com/articles/rust-cython-python-extensions/
- 

- …

7.2 .Net

7.2.1 Python.NET

- https://github.com/pythonnet/pythonnet

Python.NET is a package that gives Python programmers nearly seamless integration with the .NET Common Language Runtime (CLR) and provides a powerful application scripting tool for .NET developers. It allows Python code to interact with the CLR, and may also be used to embed Python into a .NET application.

- Languages

- https://pythonnet.github.io/
- https://github.com/pythonnet/pythonnet/wiki
- …

Example

- **Calling .NET code from Python**

Python.NET allows CLR namespaces to be treated essentially as Python packages.

```
import clr
from System import String
from System.Collections import *
```

To load an assembly, use the `AddReference` function in the `clr` module:

```
import clr
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Form
```

- **Embedding Python in .NET**

- You must set `Runtime.PythonDLL` property or `PYTHONNET_PYDLL` environment variable starting with version 3.0, otherwise you will receive `BadPythonDllException` (internal, derived from `MissingMethodException`) upon calling `Initialize`. Typical values are `python38.dll` (Windows), `libpython3.8.dylib` (Mac), `libpython3.8.so` (most other Unix-like operating systems).
- All calls to python should be inside a `using (Py.GIL()) {/* Your code here */}` block.
- Import python modules using `dynamic mod = Py.Import("mod")`, then you can call functions as normal, eg `mod.func(args)`.
- Use `mod.func(args, Py.kw("keywordargname", keywordargvalue))` or `mod.func(args, keywordargname: keywordargvalue)` to apply keyword arguments.
- All python objects should be declared as `dynamic` type.
- Mathematical operations involving python and literal/managed types must have the python object first, eg. `np.pi * 2` works, `2 * np.pi` doesn't.

```
static void Main(string[] args)
{
    using (Py.GIL())
    {
        dynamic np = Py.Import("numpy");
        Console.WriteLine(np.cos(np.pi * 2));

        dynamic sin = np.sin;
        Console.WriteLine(sin(5));

        double c = (double)(np.cos(5) + sin(5));
        Console.WriteLine(c);

        dynamic a = np.array(new List<float> { 1, 2, 3 });
        Console.WriteLine(a.dtype);

        dynamic b = np.array(new List<float> { 6, 5, 4 }, dtype: np.int32);
        Console.WriteLine(b.dtype);

        Console.WriteLine(a * b);
        Console.ReadKey();
    }
}
```

Output:

```
1.0
-0.958924274663
-0.6752620892
float64
int32
[  6.  10.  12.]
```

7.3 JVM Languages

7.3.1 Java

Overview

- …

7.3.2 Scala

7.3.2.1 ScalaPy

Overview

- https://scalapy.dev/

Use Python libraries from the comfort of Scala.

- https://scalapy.dev/docs/
- https://github.com/scalapy/scalapy
- …

9) Gaming

Overview

- …

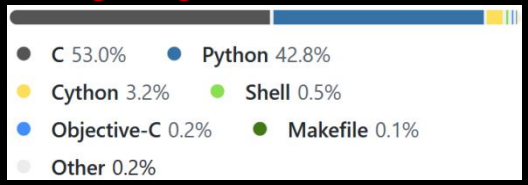
9.1 PyGame

Overview

- https://www.pygame.org
- https://github.com/pygame/pygame

pygame is a free and open-source cross-platform library for the development of multimedia applications like video games using Python. It uses the Simple DirectMedia Layer library and several other popular libraries to abstract the most common functions, making writing these programs a more intuitive task.

- Languages

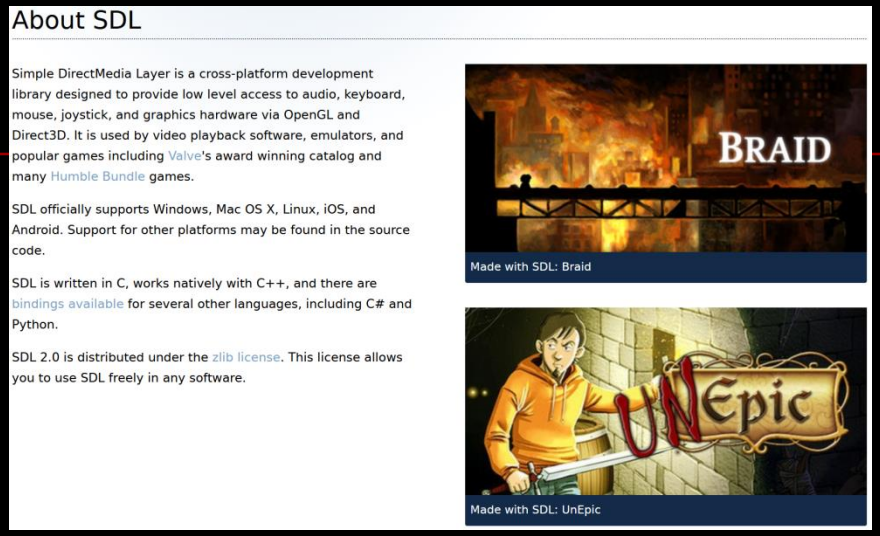


- Dependencies

Pygame is obviously strongly dependent on SDL and Python. It also links to and embeds several other smaller libraries. The font module relies on SDL_ttf, which is dependent on freetype. The mixer (and mixer.music) modules depend on SDL_mixer. The image module depends on SDL_image, which also can use libjpeg and libpng. The transform module has an embedded version of SDL_rotozoom for its own rotozoom function. The surfarray module requires the Python NumPy package for its multidimensional numeric arrays. Dependency versions:

- CPython >= 3.6 or PyPy3
- SDL >= 2.0.1
- SDL_mixer >= 2.0.0
- SDL_image >= 2.0.0
- SDL_ttf >= 2.0.11
- SDL_gfx (optional, vendored in)
- NumPy >= 1.6.2 (optional)

- **SDL**
  https://www.libsdl.org/



Examples

- **https://www.pygame.org/tags/all**
- **https://github.com/pygame/pygame/tree/main/examples**
- **…**

Good Resources

- **https://www.pygame.org/wiki/GettingStarted**
- **https://www.pygame.org/docs/**
- **https://www.pygame.org/wiki/**
- **...**

9.2 Kivy

Overview

- **https://www.kivy.org**

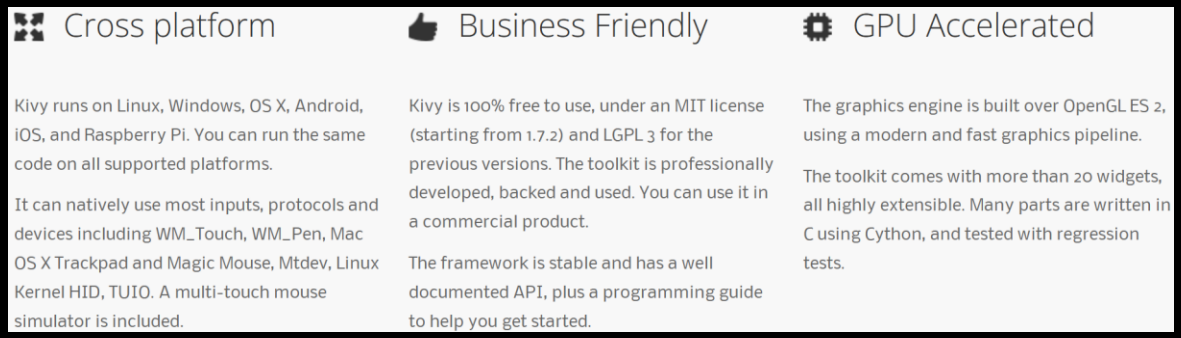
Innovative user interfaces made easy.

Kivy is an open source, cross-platform Python framework for the development of applications that make use of innovative, multi-touch user interfaces. The aim is to allow for quick and easy interaction design and rapid prototyping whilst making your code reusable and deployable.

Kivy is written in Python and Cython, based on OpenGL ES 2, supports various input devices and has an extensive widget library. With the same codebase, you can target Windows, macOS, Linux, Android and iOS. All Kivy widgets are built with multitouch support.

- **Languages**

- **Features**



- **https://kivy.org/doc/stable/**

- **https://github.com/kivy/kivy**

Sister projects

- Buildozer: generic Python packager for Android and iOS.
- Plyer: platform-independent Python wrapper for platform-dependent APIs.
- Pyjnius: dynamic access to the Java/Android API from Python.
- Pyobjus: dynamic access to the Objective-C/iOS API from Python.
- Python for Android: toolchain for building and packaging Python applications for Android.
- Kivy iOS: toolchain for building and packaging Kivy applications for iOS.
- Audiostream: library for direct access to the microphone and speaker.
- KivEnt: entity-based game engine for Kivy.
- Garden: widgets and libraries created and maintained by users.
- Oscpy: a fast and tested python2/3 implementation of OSC.

Examples

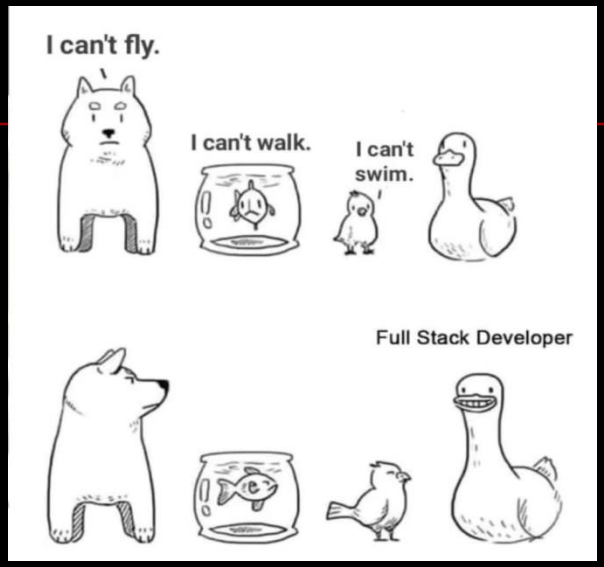
- **https://kivy.org/#gallery**
- **https://kivy.org/doc/stable/examples/**
- **https://github.com/kivy/kivy/tree/master/examples**
- **…**

Good Resources

- **https://github.com/kivymd/KivyMD**
- **...**

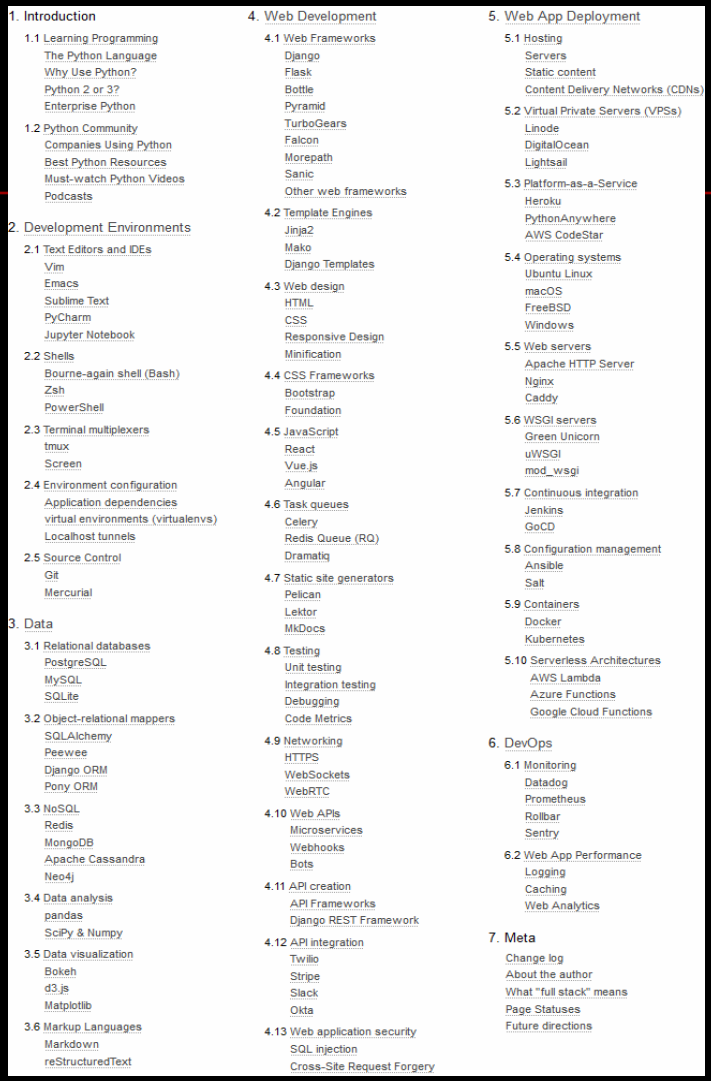
10) Full Stack

Overview

- 

- **https://www.w3schools.com/whatis/whatis_fullstack.asp**

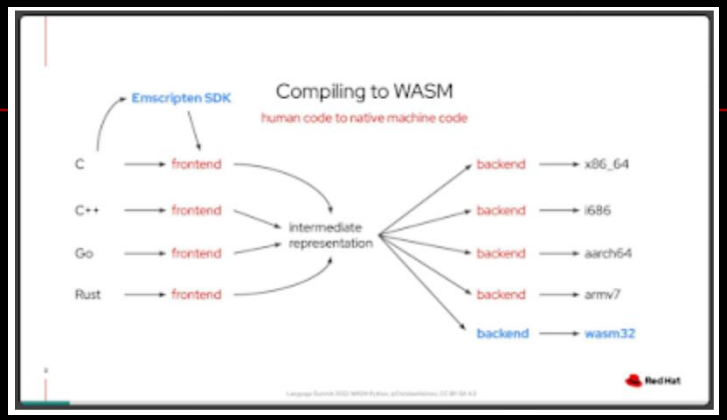
Good Resources



1. Introduction
   - 1.1 Learning Programming
     - The Python Language
     - Why Use Python?
     - Python 2 or 3?
     - Enterprise Python
   - 1.2 Python Community
     - Companies Using Python
     - Best Python Resources
     - Must-watch Python Videos
     - Podcasts
2. Development Environments
   - 2.1 Text Editors and IDEs
     - Vim
     - Emacs
     - Sublime Text
     - PyCharm
     - Jupyter Notebook
   - 2.2 Shells
     - Bourne-again shell (Bash)
     - Zsh
     - PowerShell
   - 2.3 Terminal multiplexers
     - tmux
     - Screen
   - 2.4 Environment configuration
     - Application dependencies
     - virtual environments (virtualenvs)
     - Localhost tunnels
   - 2.5 Source Control
     - Git
     - Mercurial
3. Data
   - 3.1 Relational databases
     - PostgreSQL
     - MySQL
     - SQLite
   - 3.2 Object-relational mappers
     - SQLAlchemy
     - Peewee
     - Django ORM
     - Pony ORM
   - 3.3 NoSQL
     - Redis
     - MongoDB
     - Apache Cassandra
     - Neo4j
   - 3.4 Data analysis
     - pandas
     - SciPy & Numpy
   - 3.5 Data visualization
     - Bokeh
     - d3.js
     - Matplotlib
   - 3.6 Markup Languages
     - Markdown
     - reStructuredText
4. Web Development
   - 4.1 Web Frameworks
     - Django
     - Flask
     - Bottle
     - Pyramid
     - TurboGears
     - Falcon
     - Morepath
     - Sanic
     - Other web frameworks
   - 4.2 Template Engines
     - Jinja2
     - Mako
     - Django Templates
   - 4.3 Web design
     - HTML
     - CSS
     - Responsive Design
     - Minification
   - 4.4 CSS Frameworks
     - Bootstrap
     - Foundation
   - 4.5 JavaScript
     - React
     - Vue.js
     - Angular
   - 4.6 Task queues
     - Celery
     - Redis Queue (RQ)
     - Dramatiq
   - 4.7 Static site generators
     - Pelican
     - Lektor
     - MkDocs
   - 4.8 Testing
     - Unit testing
     - Integration testing
     - Debugging
     - Code Metrics
   - 4.9 Networking
     - HTTPS
     - WebSockets
     - WebRTC
   - 4.10 Web APIs
     - Microservices
     - Webhooks
     - Bots
   - 4.11 API creation
     - API Frameworks
     - Django REST Framework
   - 4.12 API integration
     - Twilio
     - Stripe
     - Slack
     - Okta
   - 4.13 Web application security
     - SQL injection
     - Cross-Site Request Forgery
5. Web App Deployment
   - 5.1 Hosting
     - Servers
     - Static content
     - Content Delivery Networks (CDNs)
   - 5.2 Virtual Private Servers (VPSs)
     - Linode
     - DigitalOcean
     - Lightsail
   - 5.3 Platform-as-a-Service
     - Heroku
     - PythonAnywhere
     - AWS CodeStar
   - 5.4 Operating systems
     - Ubuntu Linux
     - macOS
     - FreeBSD
     - Windows
   - 5.5 Web servers
     - Apache HTTP Server
     - Nginx
     - Caddy
   - 5.6 WSGI servers
     - Green Unicorn
     - uWSGI
     - mod_wsgi
   - 5.7 Continuous integration
     - Jenkins
     - GoCD
   - 5.8 Configuration management
     - Ansible
     - Salt
   - 5.9 Containers
     - Docker
     - Kubernetes
   - 5.10 Serverless Architectures
     - AWS Lambda
     - Azure Functions
     - Google Cloud Functions
6. DevOps
   - 6.1 Monitoring
     - Datadog
     - Prometheus
     - Rollbar
     - Sentry
   - 6.2 Web App Performance
     - Logging
     - Caching
     - Web Analytics
7. Meta
   - Change log
   - About the author
   - What "full stack" means
   - Page Statuses
   - Future directions

https://github.com/mattmakai/fullstackpython.com

10.1 Python & Wasm
10.1.1 Python in the Browser

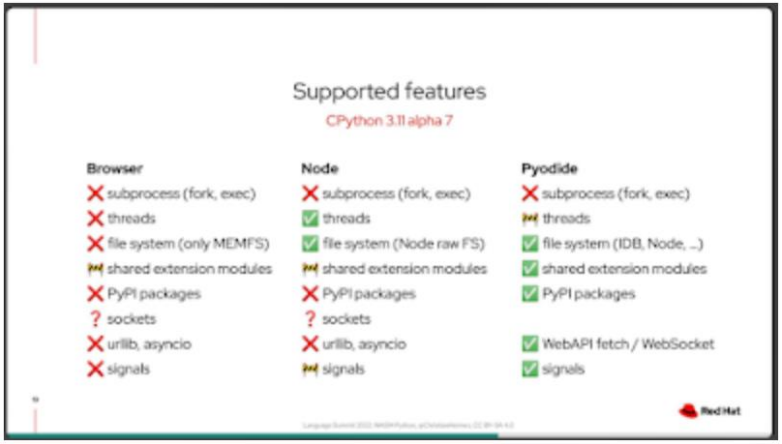


Source: https://pyfound.blogspot.com/2022/05/the-2022-python-language-summit-python.html

- https://github.com/ethanhs/python-wasm
- https://github.com/sagemathinc/wasm-python
- …

Official Support Status

- It should be noted that cross-compiling to WebAssembly is still highly experimental, and not yet officially supported by CPython. Several important modules in the Python standard library are not currently included in the bundled package produced when `--with-emscripten-target=browser` is specified, leading to a number of tests needing to be skipped in order for the test suite to pass.



Source: https://pyfound.blogspot.com/2022/05/the-2022-python-language-summit-python.html

- ...

Good Resources

- https://pythondev.readthedocs.io/wasm.html
- https://testdriven.io/blog/python-webassembly/
- https://www.zdnet.com/article/programming-languages-want-to-run-python-code-in-a-browser-soon-you-might-be-able-to/
- https://speakerdeck.com/tiran/language-summit-2022-webassembly-python-in-the-browser-and-beyond

11) In Action

Overview

- …

11.1 Game Jam

Overview

- https://en.wikipedia.org/wiki/Game_jam

A **game jam** is an event where participants try to make a video game from scratch. Depending on the format, participants might work independently, or in teams. The event duration usually ranges from 24 to 72 hours. Participants are generally programmers, game designers, artists, writers, and others in game development-related fields. Some game jams are contests, others are not.

Traditionally, game jams focus on video games;[1] however, board games have also been the subject of game jams.[2]

- https://globalgamejam.org/
- https://itch.io/jams
- …

PyWeek

- https://pyweek.org/

Welcome to **PyWeek**, a bi-annual game jam to write games in Python.

The PyWeek challenge:

1. Invites entrants to write a game in one week from scratch either as an individual or in a team,
2. Is intended to be challenging and fun,
3. Will hopefully increase the public body of game tools, code and expertise,
4. Will let a lot of people actually finish a game, and
5. May inspire new projects (with ready made teams!)

Entries must be developed in Python during the challenge, and must incorporate some theme decided at the start of the challenge. See the challenge rules for more information.

- https://pyweek.readthedocs.io/en/latest/rules.html
- https://pyweek.org/33/

XI. Wrap-up

- As the gap among different implementations against Python Specs is not small and CPython is the de facto standard, so continuously improve CPython is the most practical solution, especially for bringing JIT to CPython.
- GraalPython is the most attractive candidate in the near future.
- From performance point of view, GraalPython and RustPython probably have the highest growth potential in the future.
- Pyodide have great potential especially when add support for WASI.
- Though the performance improvement of PyPy seems have reached its limit, it comes with some advanced features.
- Pyjion is awesome, it seems provide a best way to keep pace with CPython, and is especially suited to the cloud.

- From the perspective of hardware platform support, RustPython and Pyjion have a built-in" advantage.

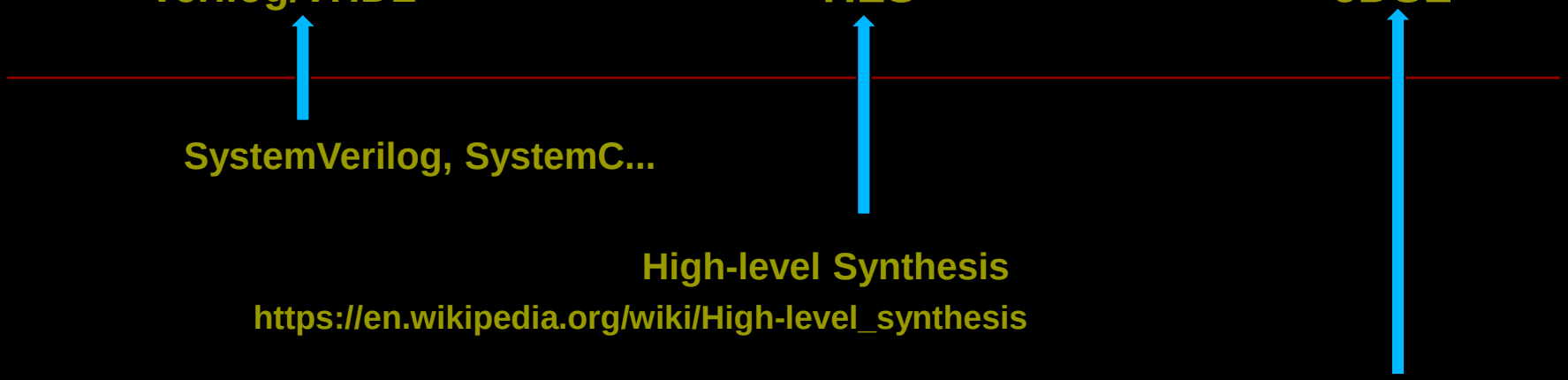
- If you are interested in some good Python-based open source projects, or the technologies like GraalVM, Wasm, .Net, Rust, IoT Edge, Serverless and so on, you may refer to our previous talks (all the slides are put at https://github.com/XianBeiTuoBaFeng2015/MySlides), and some new ones(includes this slides) will be uploaded soon.

- Please look forward to our follow-ups like "Revisiting current Python implementations" and more…

Backup Slides

Evolution of HDL

- https://en.wikipedia.org/wiki/Hardware_description_language
- Verilog/VHDL → HLS → eDSL



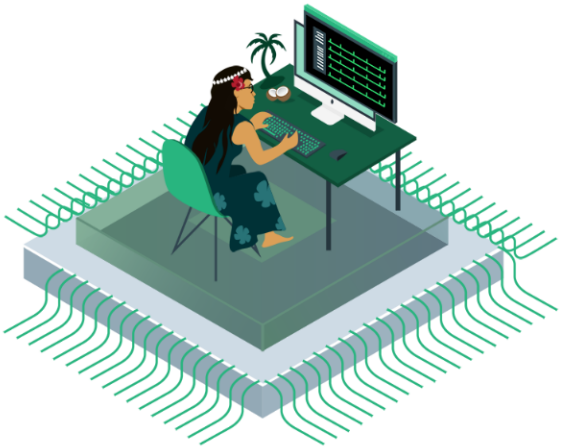
Typical eDSLs

- Haskell as host
  Bluespec, Clash...
- Scala as host
  Chisel, SpinalHDL...
- Python as host
  FHDL, PyGears...
- Finally convert to Verilog or VHDL
  https://en.wikipedia.org/wiki/Source-to-source_compiler

Cocotb

- https://www.cocotb.org/

Use cocotb to test and verify chip designs in Python. Productive, and with a smile.

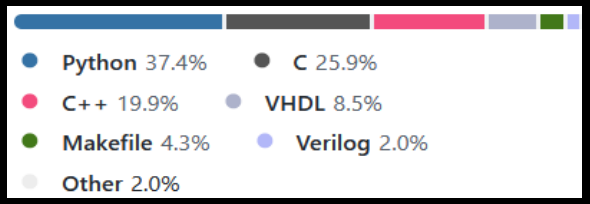


> cocotb is an open source coroutine-based cosimulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.

cocotb lets you verify chips like software: productive, simulator-agnostic, in Python.

- https://github.com/cocotb/cocotb

Languages:



- **https://docs.cocotb.org/en/stable/**

How is it different?

cocotb encourages the same philosophy of design re-use and randomized testing as UVM, however is implemented in Python.

With cocotb, VHDL or SystemVerilog are normally only used for the design itself, not the testbench.

cocotb has built-in support for integrating with continuous integration systems, such as Jenkins, GitLab, etc. through standardized, machine-readable test reporting formats.

cocotb was specifically designed to lower the overhead of creating a test.

cocotb automatically discovers tests so that no additional step is required to add a test to a regression.

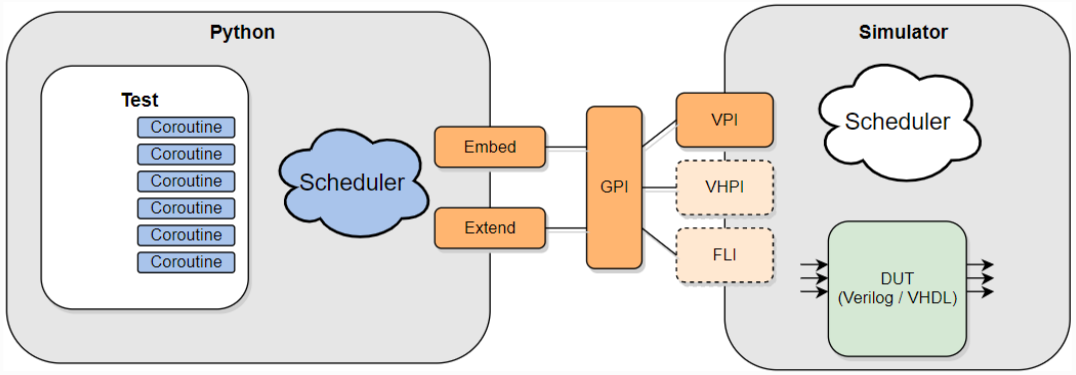
All verification is done using Python which has various advantages over using SystemVerilog or VHDL for verification:

- Writing Python is **fast** - it's a very productive language.
- It's **easy** to interface to other languages from Python.
- Python has a huge library of existing code to **re-use**.
- Python is **interpreted** - tests can be edited and re-run without having to recompile the design or exit the simulator GUI.
- Python is **popular** - far more engineers know Python than SystemVerilog or VHDL.

How it works

Source: https://www.cocotb.org/

A typical cocotb testbench requires no additional RTL code. The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. cocotb drives stimulus onto the inputs to the DUT (or further down the hierarchy) and monitors the outputs directly from Python. Note that cocotb can not instantiate HDL blocks - your DUT must be complete.



A test is simply a Python function. At any given time either the simulator is advancing time or the Python code is executing. The `await` keyword is used to indicate when to pass control of execution back to the simulator. A test can spawn multiple coroutines, allowing for independent flows of execution.

Source: https://docs.cocotb.org/en/stable/

Further Resources

- **https://github.com/cocotb/cocotb/wiki/Further-Resources**

…

Extension modules (cocotbext)

A list of cocotb extensions module packaged as described in documentation.

Please add yours here! If you publish to PyPI, please add the `Framework :: cocotb` classifier.

- cocotbext-eth: Ethernet (GMII, RGMII, XGMII, PTP clock)
- cocotbext-pcie: PCI Express (PCIe), and hard IP core models for UltraScale and UltraScale+
- cocotbext-axi: AXI, AXI lite, and AXI stream
- cocotbext-i2c: I2C interface modules
- cocotbext-uart: UART interface modules
- cocotbext-wishbone: Drive and monitor Wishbone bus
- cocotbext-uart: UART testing
- cocotbext-spi: Drive SPI bus
- cocomod-fifointerface: FIFO testing
- cocotbext-interfaces: "generalization of digital interfaces and their associated behavioral models"; Avalon ST
- cocotbext-ral: A port of the uvm-python RAL to use BusDrivers

…

- **https://github.com/cocotb/cocotb/blob/master/documentation/source/extensions.rst**

Q & A


Florida Python Challenge
2021
FLPythonChallenge.org
Brandon Call
Novice: Longest Python
Dusty Crum
Professional: Longest Python
Charlie Dachton
Ultimate Grand Prize $10,000 Winner:
Most Pythons Overall
Tom Hobbs
Novice: Most Pythons
Donna Kalil
Professional: Most Pythons

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/

- https://awesome-python.com/
- https://pyfound.blogspot.com/2022/05/the-2022-python-language-summit_01678898482.html
- https://devguide.python.org/_/downloads/en/latest/pdf/
- https://realpython.com/python-news-february-2022/
- https://realpython.com/python-news-april-2022/
- https://developer.nvidia.com/cuda-python
- https://www.zhihu.com/question/318030170 //Python 的运行效率在未来是否有可能被优化到跟 C++ 近似？
- https://www.infoq.com/news/2022/03/java-18-so-far/
- https://www.debuggingbook.org/
- https://wesmckinney.com/book/ipython.html
- https://wiki.python.org/moin/PythonDebuggingTools
- …