

# Python at Twist Bioscience: Improving performance with Rust

Bioinformatics | Twist Bioscience

Kieran Hervold

[hervold@gmail.com](mailto:hervold@gmail.com)

May 17, 2017

# Getting started

## Install Rust

```
curl https://sh.rustup.rs -sSf | sh
```

## Clone today's repo:

```
https://github.com/Twistbioscience/pycon-rust
```

# Performance optimization

- PyPy
- Cython
- C/C++ with SWIG bindings

# Rust

Rust topped the list of most loved programming languages in Stack Overflow's 2016 developer survey.



Mozilla's next-generation browser, code-named Servo, is being written in Rust.

- <https://github.com/servo/servo/tree/master/components>

# Rust: a “safe, concurrent, practical systems language”

Includes a modern type system (no NULL!)

- Algebraic data types similar to Haskell, ML, F#, Scala, etc

Polymorphism via Traits

Decent iterator library

Automatic memory management through the borrow-checker

- no garbage collection overhead
- stack allocation reduces cache misses (Java's missing "Value Types")
- borrow-checker prevents dangling pointers

# Rust ergonomics

- Cargo build system
- `#[test]`
- `#[derive(Debug)]`

ex1-println.rs

```
#[derive(Debug)]
enum Sentence {
    Period,
    Word { text: String, count: usize }
}

fn main() {
    let sentence = vec![ Sentence::Word { text: "Simple".to_string(), count: 1 },
                        Sentence::Word { text: "Rust".to_string(), count: 2 },
                        Sentence::Word { text: "demo".to_string(), count: 1 },
                        Sentence::Period ];

    println!("{:?}", &sentence);
}
```

```
[Word { text: "Simple", count: 1 }, Word { text: "Rust", count: 2 },
 Word { text: "demo", count: 1 }, Period]
```

# Rust bindings in Python, ex 1: Fast Inverse Sqrt

slides/src/lib.rs

```
#[no_mangle]
pub extern fn fast_inv_sqrt(x: f32) -> f32 {
    let i: u32 = unsafe { std::mem::transmute(x) };
    let j = 0x5f3759df - (i >> 1);
    let y: f32 = unsafe { std::mem::transmute(j) };
    y * (1.5 - 0.5 * x * y * y)
}
// credit to https://github.com/itchyny/fastinvsqrt
```

slides/example.py

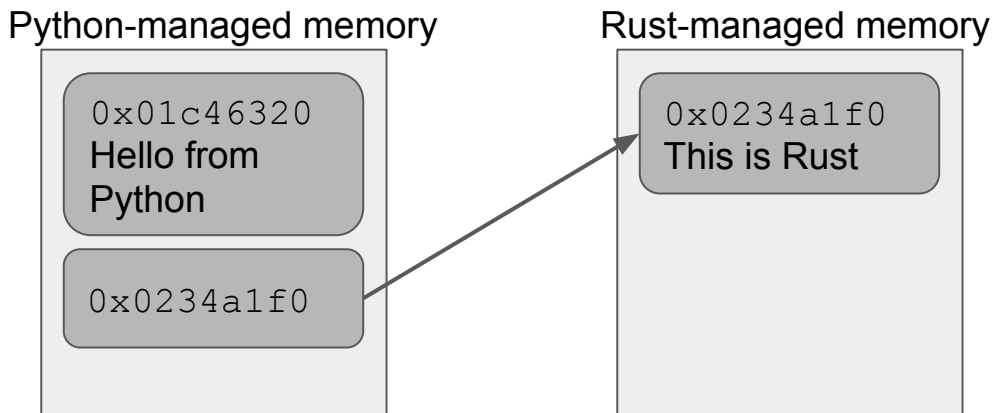
```
import ctypes

ex_lib = ctypes.CDLL("target/release/libtwist_pycon_slides.dylib")
ex_lib.fast_inv_sqrt.restype = ctypes.c_float
ex_lib.fast_inv_sqrt.argtypes = [ctypes.c_float]
print ex_lib.fast_inv_sqrt( 4.0 )
print ex_lib.fast_inv_sqrt( 9.0 )
```

```
0.499153584242
0.332953214645
```

# CFFI bindings: sharing memory

- The problem: Python and Rust maintain separate records of memory allocation



- `ctypes`' default String handling is broken: it wraps a `* char` in a Python string, and attempts to release the memory when the string falls out of scope
- Note that this is a problem for any C Foreign Function Interface library



# Returning pointers from Rust

```
simple_mc/python/simple_mc.py
```

```
import ctypes

lib = ctypes.CDLL("target/release/libsimple_mc.dylib")

lib.read_corpus_file.argtypes = [ctypes.c_char_p]
lib.read_corpus_file.restype = ctypes.c_void_p

lib.ext_generate_sentence.argtypes = [ctypes.c_void_p]
lib.ext_generate_sentence.restype = ctypes.c_void_p

# release really expects the char * from above, but we'll call it a void *
lib.release_str.argtypes = [ctypes.c_void_p]

...
p = lib.ext_generate_sentence( pointer )
sentence = ctypes.cast(p, ctypes.c_char_p).value
```