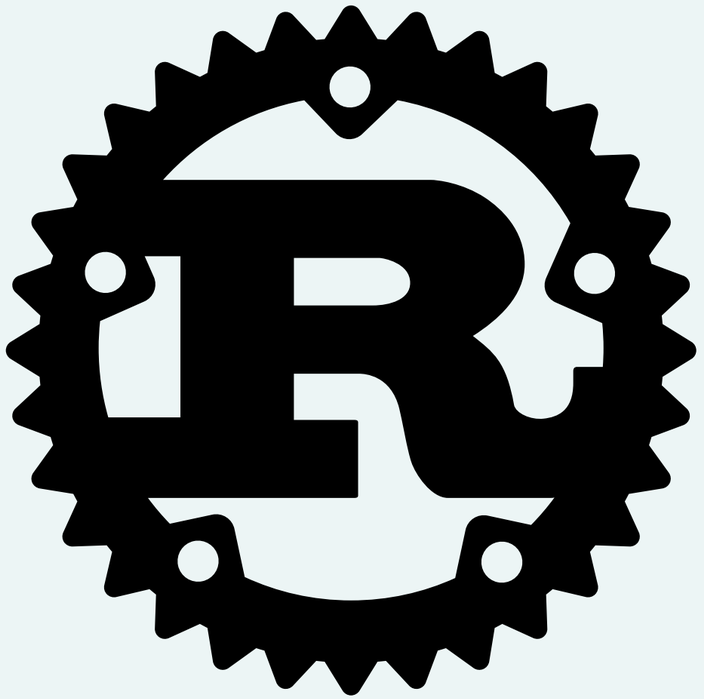
Rust from Python & Ruby



Armin Ronacher

@mitsuhiko

Python Dude

(with Ruby experience)

What is *Rust*?

```
printf("Hello "
"World!\n");
```

Programming Language

UNDER CONSTRUCTION

```
def main():
    for name in ["Peter", "Paul", "Mary"]:
        print "Hello %s!" % name
```

```
def main
    ["Peter", "Paul", "Mary"].each do |name|
        puts "Hello %s!" % name
    end
end
```

```
fn main() {
    for ["Peter", "Paul", "Mary"].each |name| {
        io::println(fmt!("Hello %s!", *name));
    }
}
```

```
fn main() {
    let s = [1, 2, 3, 4].map(|x| (*x * *x).to_str());
    for s.each |item| {
        io::println(*item);
    }
}
```

Soon TM

```
fn main() {
    let s = [1, 2, 3, 4].map(|&x| (x * x).to_str());
    for s.each |&item| {
        io::println(item);
    }
}
```

```
fn main() {
    let x = 42;
    let y = &x;
    let &z = y;
    io::println(fmt!("%d", z + z));
}
```

Compile that

```
$ cat hello.rs
fn main() {
    io::println("Hello World!");
}

$ rustc -g hello.rs

$ ./hello
Hello World!
```

In a Nutshell

- immutable by default
- static, algebraic, locally inferred types
- no dangling pointers, no overflows
- lightweight green tasks
- ahead-of-time compiled
- C compatible

Let's look closer

```
fn main() {
    for ["Peter", "Paul", "Mary"].each |name| {
        io::println(fmt!("Hello %s!", *name));
    }
}
```

```
fn main() {
    ["Peter", "Paul", "Mary"].each(|name| {
        io::println(fmt!("Hello %s!", *name));
        true
    });
}
```

Expressions

```
fn main() {
    let a = [1, 2, 3].map(|x| { *x; });
    let b = [1, 2, 3].map(|x| { *x });
    io::println(fmt!("a=%? b=%?", a, b));
}


/* a=~[ (), (), () ] b=~[ 1, 2, 3 ] */
```

Traits and Implementations

```
trait LengthPrintable {
    fn print_length();
}
```

```
impl<T> &[T]: LengthPrintable {
    fn print_length() {
        io::println(fmt!("length = %u", self.len()));
    }
}
```

```
fn main() {
    ["Peter", "Paul", "Mary"].print_length();
}
```

Memory Ownership

```
fn main() {
    let name = ~"Peter";
    let new_name = name;
    io::println(fmt!("Hello %s!", name));
}
```

Does not compile: string (currently) does not copy

```
fn main() {
    let name = ~"Peter";
    let new_name = move name;
    io::println(fmt!("Hello %s!", name));
}
```

Does not compile: using moved-out value

```
fn main() {
    let name = ~"Peter";
    let new_name = move name;
    io::println(fmt!("Hello %s!", new_name));
}
```

That compiles!

Not Null

(algebraic data types)

```
def first_larger(seq, x):
    for item in seq:
        if item > x:
            return x
```

Where is the NULL?

```
fn first_larger(seq: &[int], x: int) -> Option<int> {
    for seq.each |item| {
        if *item > x { return Some(*item); }
    }
    None
}
```

```
fn main() {
    let rv = first_larger([1, 2, 3, 4, 5], 3);
    io::println(match rv {
        Some(num) => fmt!("Found %d", num),
        None => ~"No number found"
    });
}
```

Pattern Matching!

Enums

(better than C's)

```
enum Shape {
    Point,
    Circle(float),
    Rect(float, float)
}
```

```
impl Shape : ToStr {
    pure fn to_str() -> ~str {
        match self {
            Point => ~"point",
            Circle(r) => fmt!("circle of %f", r),
            Rect(w, h) => fmt!("rect of %f by %f", w, h)
        }
    }
}
```

```
fn main() {
    let p = Point;
    let c = Circle(4.0f);
    io::println(fmt!("p=%s, c=%s",
                     p.to_str(), c.to_str()));
}
```

“Classes”

```
struct Point {
    mut x: float,
    mut y: float,
}

impl Point {
    static fn new(x: float, y: float) -> Point {
        Point { x: x, y: y }
    }
}
```

```
impl Point : ToStr {
    pure fn to_str() -> ~str {
        fmt!("(%f, %f)", self.x, self.y)
    }
}
```

```
fn main() {
    let p = Point::new(0.0f, 0.0f);
    io::println(p.to_str());
}
```

Hold on a second!
INHERITANCE?

trait inheritance
no data inheritance

Tasks

```
fn main() {
    for ["Peter", "Paul", "Mary"].each |name| {
        let name = *name;
        do task::spawn {
            let v = rand::Rng().shuffle([1, 2, 3]);
            for v.each |num| {
                io::print(fmt!("%s says: '%d'\n",
                               name, *num));
            }
        }
    }
}
```

FFI

```
#[link_args="-lcrypto"]
extern {
    fn SHA1(s: *u8, l: libc::c_uint, d: *u8) -> *u8;
}

fn sha1(data: &str) -> ~str {
    unsafe {
        let bytes = str::to_bytes(data);
        let mut buf = [0u8, ..20];
        SHA1(vec::raw::to_ptr(bytes),
             bytes.len() as libc::c_uint,
             vec::raw::to_ptr(buf));
        as_hex(buf)
    }
}
```

```
fn as_hex(data: &[u8]) -> ~str {
    let mut rv = ~"";
    for data.each |b| {
        rv += fmt!("%02x", *b as uint);
    }
    move rv
}

fn main() {
    io::println(sha1("Hello World!"));
}
```

for the lazy:

bindgen

Dear god ...

Why?

Why a new language?

Hello Concurrency

Why is concurrency

messy?

Q: Why are global variables bad?
A: there is only one of them!

Concurrency in web apps:
What do you actually share?

Memory Tracking Enables Cheap Message Passing

Spaghetti Stacks

The Problems with Rust

strings not implicitly copyable

ownership tracking not perfect yet

generics are harder to implement

`Option<T>` is a bit of a pain

directly using C APIs is icky

task scheduler oblivious of libuv

incomplete / work in progress

Thanks, now there is time for ...

@mitsuhiko
http://lucumr.pocoo.org/
http://fireteam.net/