

```
$ python -m this | sed -n '4p'
```

Explicit is better than implicit.

Rust for Pythonistas

Vinay Keerthi *@stonecharioteer*
Data @ Merkle Science

2022-03-26
BangPypers



\$ whoami

- Hi! I'm Vinay Keerthi, I go by [@stonecharioteer](#) everywhere.
- I blog at stonecharioteer.com
- Self-taught
- I have been using Python for 8 years.
- I build web applications (flask), command line tools (click), sometimes a GUI (PyQt5/Plotly Dash), and automate pipelines (airflow, flask-scheduler).
- Currently Data Engineering @ Merkle Science, a cryptocurrency analytics firm.
- Previously @ Visa, GKN Driveline, Flipkart
- I've been writing Rust code for ~30 days. Beware the errors.

What's with the title?

- The Zen of Python states “Explicit is better than implicit”
- Rust embodies this to a fault, and is a natural successor to the ideal.

What is this talk about?

Note: This is not a Rust tutorial.

- Why Rust?
- I already know Python.
- Then why would I learn Rust?
- Why not X?
- What does it look like?
- How do I learn?
- Numbers?
- I've heard Strings are hard.

Why Rust?

- I've been writing Python code my entire career.
- I wanted to try something new.
- I'd tried some go-lang resources, but I'd kept coming back to Rust.
- I first heard about Rust on news.ycombinator.com (HackerNew/HN).
- What did I know?
 - Rust is *fast*.
 - Rust has a steep learning curve.
 - It's *low level*.
 - It's got something called the *borrow checker*.
 - *Strings are weird?*

But I already know Python!

- And that's ***good***.
- Use what you know for the problems that you want to solve quickly.
- Python can do almost anything. If you need to scale, think about distributing your tasks across workers.
- Use Pandas, Numpy, and other libraries designed for speed over native data types.
- Use Cython or PyCuda to write faster code.
- Try PyPy for JIT - it's faster.

Then why would I learn Rust?

- Shipping python applications is getting *harder*.
 - There is a paradox of choice: poetry, pyenv, virtualenvwrapper, flit, conda.
 - There are ways to ship a single binary, but they ship the python interpreter.
- If you're self-taught, like me, you should learn a lower level language.
 - Python doesn't teach you about memory management.
 - What *are* threads, really?
 - How do you implement a truly safe multi-threaded application?
 - How does the memory model work?
- Rust gives you simple and safe concurrency.
- To become a better programmer.
 - More on this later.

Why not **X**?

- X is usually Golang, Java, C++ or C.
- Follow your instinct. Learn what you want to.
- Performance matters, and it *should*.
- Ownership and borrowing is *interesting*.
- Have you seen *really* small binaries?
- Cargo is an amazing package manager that does everything for you.

What does it look like?



```
1 fn main() {  
2     println!("ನಮಸ್ಕಾರ, BangPypers!");  
3 }  
4
```

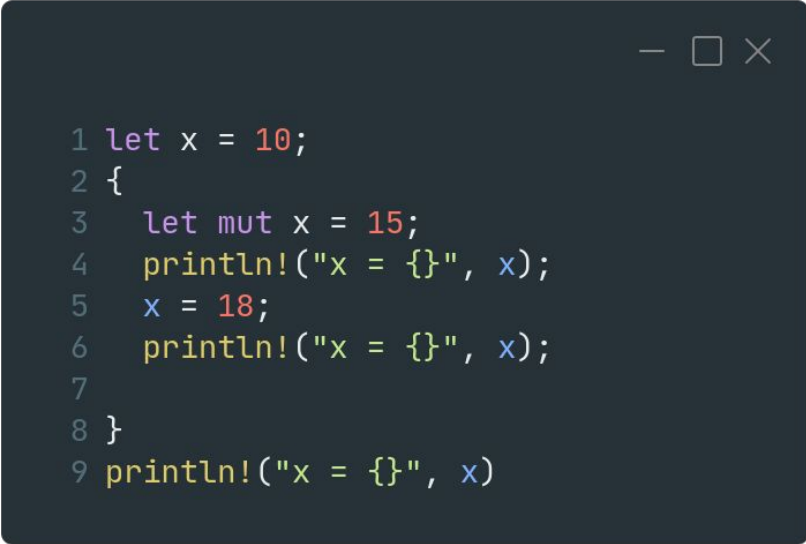
What does it look like? Real Code this time.

```
1 /// This store is having a sale where if the price is an even number, you get
2 /// 10 Rustbucks off, but if it's an odd number, it's 3 Rustbucks off.
3
4 /// This is the function that is triggered when you're building a binary, a CLI or a server
5 /// instance for example.
6 fn main() {
7     let original_price = 51;
8     println!("Your sale price is {}", sale_price(original_price));
9 }
10
11 /// This function takes an i32 integer and returns an i32 integer.
12 /// Rusts typing is extremely strict. If it looks like a duck, don't trust it.
13 /// If you want something that quacks, then don't ask for a duck. Ask for something
14 /// that quacks.
15 fn sale_price(price: i32) -> i32 {
16     if is_even(price) {
17         price - 10
18     } else {
19         price - 3
20     }
21 }
22
23 /// This function takes an i32 and returns a boolean value.
24 /// Even the number size really matters. Rust showed me that we use inconsistent
25 /// database schemas in some places because I was using u32 (unsigned integer 32 bit),
26 /// and one particular database schema returned a u64 instead. That's an error you wouldn't get
27 /// with Python.
28 fn is_even(num: i32) -> bool {
29     num % 2 == 0
30 }
31
```

What does it look like? More Real Code.

```
1 #[derive(Debug)]
2 struct Package {
3     sender_country: String,
4     recipient_country: String,
5     weight_in_grams: i32,
6 }
7
8 impl Package {
9     fn new(sender_country: String, recipient_country: String, weight_in_grams: i32) → Package {
10         if weight_in_grams ≤ 0 {
11             panic!("uh-oh! what do you mean that the weight is negative?");
12         } else {
13             return Package {
14                 sender_country,
15                 recipient_country,
16                 weight_in_grams,
17             };
18         }
19     }
20
21     fn is_international(&self) → bool {
22         self.sender_country ≠ self.recipient_country
23     }
24
25     fn get_fees(&self, cents_per_gram: i32) → i32 {
26         self.weight_in_grams * cents_per_gram
27     }
28 }
```

Memory Management in Rust: Scope and Mutability



```
1 let x = 10;
2 {
3     let mut x = 15;
4     println!("x = {}", x);
5     x = 18;
6     println!("x = {}", x);
7
8 }
9 println!("x = {}", x)
```

Memory Management in Rust: Movement

```
1 fn main() {  
2     let x: String = "hello".to_string();  
3     println!("x = {}", x);  
4     let y = x;  
5     println!("y = {}", y);  
6     println!("x = {}", x); // ERROR: This won't even compile because x has *moved* into y.  
7 }
```

- You can't keep passing variables around and copying them over without thinking.
- Rust's memory model is centered around Ownership and Borrowing.

Memory Management in Rust: Movement - Compiler errors

```
1> main.rs
6 fn main() {
H 5   let x: String = "hello".to_string();    ■ move occurs because `x` has type `String`, which does not implement the `Copy` trait
4   println!("x = {}", x);
H 3   let y = x;    ■ value moved here
2   println!("y = {}", y);
E 1   println!("x = {}", x); // ERROR: This won't even compile because x has *moved* into y.    ■ borrow of moved value: `x` value borrowed here after move
7   }
Not Committed Yet
```

- The Rust analyzer tells us that we can't use the value of `x` because it has *moved*.
- And it also tells us that `String` doesn't implement the Copy trait.
- Traits are *abilities* types and structs can have in Rust.
- You can implement a trait on any datatype. Some are *installable* while some need to be implemented manually.
- You can override traits to mess around, or to give your datatypes not-so-obvious features.
- For instance, you want to use subtraction to remove a substring from a string, you can do that.
- Think of it like overriding or implementing `__dunder__` methods in Python. **Note that this is a gross trivialization of what traits are.**

Memory Management in Rust: Borrowing

```
1 fn borrow_a_string(x: &str) {  
2     println!("I've only borrowed a string. The value is {}", x);  
3 }  
4  
5 fn move_a_string(x: String) {  
6     println!("I've taken ownership of a string. The value is {}", x);  
7 }  
8  
9  
10 fn main() {  
11     let v = String::from("The cake is a lie!");  
12     borrow_a_string(&v); // v is still in scope because this is just a borrow.  
13     move_a_string(v); // v is a move, so it cannot be used after this.  
14     println!("v={}", v); // if you try this, the compiler will complain.  
15 }  
16
```

Memory Management in Rust: Mutable Borrowing

```
1 fn append_to_a_vector(v: &mut Vec<u32>, a: u32) {  
2     v.push(a);  
3 }  
4  
5 pub fn run() {  
6     let mut x = vec![10, 20, 30, 40];  
7     println!("Initial Vector: {:?}", x);  
8     append_to_a_vector(&mut x, 10);  
9     println!("Final Vector: {:?}", x);  
10 }
```


Strings in Rust

- Strings are UTF-8 encoded in Rust. So a string length might not be what you think it is.
- [The docs explain it well.](#) (They use the devanagiri script as an example of why this is not straightforward)
- Ex: **नमस्कार** would not be just 4 characters.



```
1 pub fn run() {  
2     let x = "नमस्कार";  
3     println!("{}", x, x.as_bytes());  
4     println!("{}", x, x.chars().collect::<Vec<char>>());  
5 }
```

Traits & Implementations

```
trait CanWave {
    fn get_name(&self) → &str;
    fn wave(&self) → String;
}

struct Adult {
    name: String,
}

impl CanWave for Adult {
    fn get_name(&self) → &str {
        &self.name
    }
    fn wave(&self) → String {
        String::from("Hey look, I'm an adult waving!")
    }
}

struct Child {
    name: String
}

impl CanWave for Child {
    fn get_name(&self) → &str {
        &self.name
    }
    fn wave(&self) → String {
        String::from("gaga, googoo. *waves hands*")
    }
}
```

```
struct TrainedDog {
    name: String
}

impl CanWave for TrainedDog {
    fn get_name(&self) → &str {
        &self.name
    }
    fn wave(&self) → String {
        String::from("Woof! *waves paw*")
    }
}

struct Dog {
    name: String
}

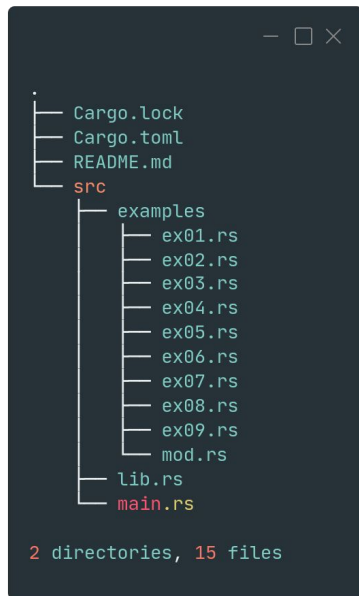
fn wave_for_me(x: &impl CanWave) {
    println!("{}", waves: '{}', x.get_name(), x.wave());
}

pub fn run() {
    let kumar = Adult { name: "Kumar".to_string() };
    let junior = Child { name: "Junior".to_string() };
    let tiny = TrainedDog { name: "Tiny".to_string() };
    let chotu = Dog { name: "Chotu".to_string() };
    wave_for_me(&kumar);
    wave_for_me(&junior);
    wave_for_me(&tiny);
    // ERROR IF UNCOMMENTED
    // wave_for_me(&chotu);
    // This will not compile because 'Dog' doesn't implement CanWave, and thus cannot
    // be used with 'wave_for_me'.
    println!("{}", hasn't been trained to wave. Maybe you could teach him?", chotu.name);
}
```

Strings in Rust

```
1 // rustlings quiz2.rs
2 // This is a quiz for the following sections:
3 // - Strings
4
5 // Ok, here are a bunch of values-- some are `String`'s, some are `&str`'s. Your
6 // task is to call one of these two functions on each value depending on what
7 // you think each value is. That is, add either `string_slice` or `string`
8 // before the parentheses on each line. If you're right, it will compile!
9
10 fn string_slice(arg: &str) {
11     println!("{}", arg);
12 }
13
14 fn string(arg: String) {
15     println!("{}", arg);
16 }
17
18 pub fn run() {
19     string_slice("blue");
20     string("red".to_string());
21     string(String::from("hi"));
22     string("rust is fun!".to_owned());
23     string("nice weather".into());
24     string(format!("Interpolation {}", "Station"));
25     string_slice(&String::from("abc")[0..1]);
26     string_slice(" hello there ".trim());
27     string("Happy Monday!".to_string().replace("Mon", "Tues"));
28     string("mY sHiFt KeY iS sTiCkY".to_lowercase());
29 }
```

The Module System



```
[package]
name = "rust_for_pythonistas"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at
https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
structopt = "0.3.26"
log = "0.4.16"
```

How does Rust make me a better programmer?

- Types are not suggestions anymore.
 - Datatypes really matter.
 - Trait or interface oriented programming teaches you to think in terms of what an object *can do* and not what it *is*.
- I've never thought about how my variables move in and out of scopes before.
 - Closures are *fun*.
 - Having control over which parts of your code *can* and *cannot* modify your variables lets you think in a way you haven't before.
- I need to account for everything I've written.
 - When I use an enum variant in a match statement, I need to match for each and every case.
 - I need to account for errors in the same way.
 - I can *define* functions as having a return value that *must be used and not ignored*.

How do I...

<u>Create a GUI?</u>	<u>Create a CLI?</u>	<u>Write ML code?</u>
<u>Connect to a database?</u>	<u>Create a game?</u>	<u>Write a linked list?</u>
<u>Create a web application?</u>	<u>Connect to the Ethereum Blockchain?</u>	<u>Configure my IDE?</u>
<u>Write async code?</u>	<u>Write web assembly code?</u>	<u>Start learning?</u>

Resources

1. [The Rust Book](#) (Official Docs)
2. [CS 4414 - Operating Systems - Using Rust for an Undergrad OS Course](#)
(Reasons to Use Rust)
3. [A Half-Hour to Learn Rust](#) (Blog Article)
4. [Rustlings - Interactive Exercises](#) (Official companion exercises)
5. [Let's Get Rusty - The Rust Lang Book \(Videos\)](#)
6. [Rust in Action \(Book\)](#)
7. [Rust for Rustaceans \(Book\)](#)
8. [Zero to Prod \(Book\)](#)
9. [Example Code & Project Structure](#)