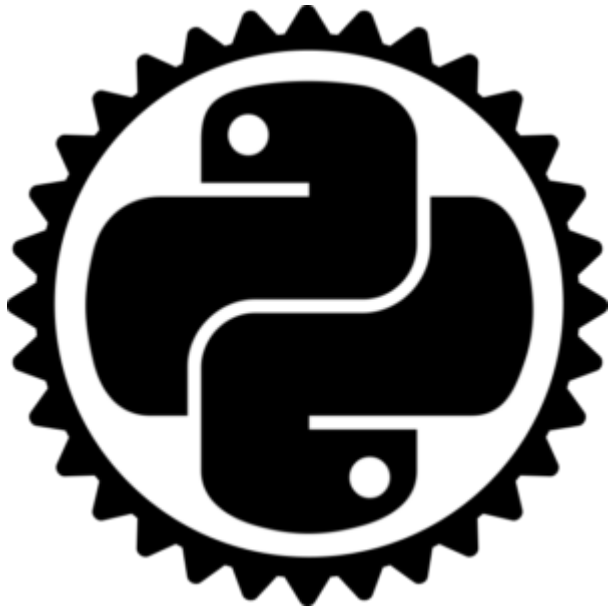
Rust for Pythonistas

Josh Karpel



Part One

What does Rust do?

Part Two

But what does Rust do *for me*?

What problems does Rust solve?

System Programming

- Graphics engines
- Databases
- Low-level utilities (ex: CLI tools)
- Operating systems
- Embedded software (ex: IOT devices)

Why? / How?

- Memory safety
- Rich static typing
- Interoperability with C
- Zero-cost abstractions

Fast

Safe

Expressive

Rust is Pythonic

What is Pythonic?

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one -- and preferably only one -- obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Rust is Pythonic

```rust
fn filtered_sum() -> i64 {
    (0..1000)
        .filter(|x| x % 3 == 0 || x % 5 == 0)
        .sum()
}
```

```python
def filtered_sum() -> int:
    return sum(
        x for x in range(1000)
        if x % 3 == 0 or x % 5 == 0
    )
```

- ✓ 5 lines long
- ✓ Types declared
- ✓ Reads like English

Mutable Assignment and Ownership

```python
def ownership():
    foo = 1
    bar = foo

    bar += 5

    print(foo)
    print(bar)
```

```rust
fn ownership() {
    let foo = 1;
    let mut bar = foo;

    bar += 5;

    println!("{}", foo);
    println!("{}", bar);
}
```

```python
def ownership2():
    foo = [1, 2, 3]
    bar = foo

    bar.append(4)

    print(foo)
    print(bar)
```

```rust
fn ownership2() {
    let foo = vec![1, 2, 3];
    let mut bar = foo;

    bar.push(4);

    println!("{:?}", foo);
    println!("{:?}", bar);
}
```

Rust is **not** Python

or: why didn't that work!?

Ownership

Variables and data structures **own** their data.

Every piece of data has **exactly one owner** at any given time.
A variable can **reference** data that another variable owns.

Assignment can cause data to **move** between owners.
A variable with no data **ceases to exist**.
Data with no variables **ceases to exist**.

Types can implement **`Clone`** to allow some duplication (ex: `Vec<T>`).
Types marked as **`Copy`** can be bit-for-bit copied (ex: `i64`).

Wait...
why did **this** work?

```
fn ownership() {
    let foo = 1;
    let mut bar = foo;

    bar += 5;

    println!("{}", foo);
    println!("{}", bar);
}
```

foo is an **i64**,
which is **Copy**,
So this line
implicitly copies foo

Wait...
why did **this** work!?

```
def ownership():
    foo = 1
    bar = foo

    bar += 5

    print(foo)
    print(bar)
```

bar is an **int**,
which is **immutable**,
So this line
isn't actually
in-place addition

[Ned Batchelder - Facts and Myths about Python names and values]

Why Ownership?

Memory Safety

- Memory Deallocation is Absolutely Predictable
- Compile-Time Detection of Invalid Memory Access

Mutability

- Strict Control of Who Can Modify Data
- Breaking the Rules is Explicit

Runtime Speed

- No Garbage Collector

Structs and Implementations

```rust
struct Position {
    // sort of equivalent of class constructor
    x: f64,
    y: f64,
}

impl Position {
    // equivalent of a method
    fn distance_to_origin(&self) -> f64 {
        (self.x.powi(2) + self.y.powi(2)).sqrt()
    }
}
```

A Contrived Example

Define a
system of subclasses
that represents a
finite, discrete set of values,
and a
function whose output depends on which subclass is passed into it

```
class Direction: pass

class Up(Direction): pass

class Down(Direction): pass

class Left(Direction): pass

class Right(Direction): pass

type_to_arrow = {
    Up: '↑',
    Down: '↓',
    Left: '←',
    Right: '→',
}

def to_arrow(dir: Direction) -> str:
    return type_to_arrow[type(dir)]
```

Enumerable Data

Define an
~~system of subclasses~~
enumeration
~~that represents a~~
~~finite, discrete set of values,~~
does what enumerations do
and a
~~function whose output depends on which subclass is passed into it~~
function with a switch inside

```python
from enum import Enum

class Direction(Enum):
    Up = 'up'
    Down = 'down'
    Left = 'left'
    Right = 'right'

    def to_arrow(self) -> str:
        return type_to_arrow[self]

type_to_arrow = {
    Direction.Up: '↑',
    Direction.Down: '↓',
    Direction.Left: '←',
    Direction.Right: '→',
}
```

Enumerable Data is Extremely Common

but rarely has good support in dynamic languages

Examples

Cardinal Directions
Booleans (`True` and `False`)
IPv4 and IPv6
Programming Language Keywords
Data Types in a Database
Playing Card Suits
Days of the Week

Rust has First-Class Enumerations

(like many statically-typed languages do)

but match gives them superpowers!

```
match = dictionary keyed by type
      + tuple unpacking
      + lambda function
```

Part Two

But what does Rust do *for me*?

Part 2A: Philosophy

How did learning Rust influence my Python code?

Part 2B: Python Integration

But what does Rust do for me, *literally*?

Rust changed how I write Python

Static Typing

- Use Type Annotations
- Think About Data Flow

Ownership

- Always Know Who's in Charge of What

Structs & Traits

- More Struct-like Data
- Favor Composition over Inheritance
- Think Using Interfaces

Enums & Match

- Make Choices Explicit
- Favor Dictionaries over Conditionals

Part 2B: Python Integration

But what does Rust do for me, *literally*?

Goal

Write a **Python extension module** in **Rust**

(like NumPy, but way less useful)

Where To Next?

Official Tutorial

The Rust Book

Tutorial & News Podcast

New Rustacean

An Actual Book

Programming Rust

Talks

Rust Youtube

My Stuff

This Talk
https://github.com/JoshKarpel/rust-for-pythonistas
https://github.com/JoshKarpel/pyext-example

A Befunge-93 Interpreter
https://github.com/JoshKarpel/fungoid

`Some`/`None` and `Ok`/`Err` in Python
https://github.com/JoshKarpel/hypoxia
(please don't use this anywhere real)