Intro to Rust for Experienced Developers

Daniel Scherzer

Daniel Scherzer

- Full-stack developer and open source contributor
- https://scherzer.dev
- http://github.com/DanielEScherzer
- 10+ years of software development
- PHP core contributor
  - ext/reflection maintainer
  - 8.5 release manager
- ~6 months of Rust development (origin story)

Setting expectations

- Fast-paced introduction to Rust 1.93.1
- Comparisons to C, C++, Javascript, PHP, and Python
- Does not cover unstable features or unsafe features
- Not 100% accurate, like most introductions
  - Operator overloads (C++, Python)
  - Private visibility (PHP)
- Links to documentation, reference, and books

Rust does not start with a “P”

Like

- C
- C++
- JavaScript

Unlike

- PHP
- Python

- Like = similar to Rust
- Don’t get confused by negatives

Rust has helpful error messages

“As usual, this output tells us exactly what has gone wrong.”

— Book discussion of `Result`

From the development guide page about error codes

> Error explanations should expand on the error message and provide details about why the error occurs. It is not helpful for users to copy-paste a quick fix; explanations should help users understand why their code cannot be accepted by the compiler. Rust prides itself on helpful error messages and long-form explanations are no exception.

Hello, World!

```
fn main () {
        version1();
        version2();
        let conf = "ConFoo";
        version3(&conf);
}

fn version1() {
        println!("Hello, World!");
}
fn version2() { println!("Hi again"); }
fn version3( conf: &str ) -> () {
        println!("Hello, {}!", conf);
}
```

- Uses a `main` function (discussed later)
- Statements end with semicolons
  - Ref: expr.block.statements
- Whitespace insensitive
  - Ref: lex.whitespace.token-sep
- Return type after parameters
  - Ref: items.fn.syntax
- Return optional for unit type
  - Ref: items.fn.implicit-return
- No forward declaration/prototype
  - Like: JavaScript, PHP, Python
  - Unlike: C, C++

Rust is compiled

Like

- C
- C++

Unlike

- JavaScript
- PHP
- Python

- `rustc` command, given file name, produces binary
  - rustc book
- Generally invoked via cargo for crates
  - Book chapter
  - Cargo book

Rust is compiled... and has a main function

Like

- C
- C++

Unlike

- JavaScript
- PHP
- Python

- Initial entry point
  - Ref: crates.main.general
- No parameters, returns `Termination` instance
  - std::env::args()
  - Ref: crate.main.restriction

Rust is compiled… and statically typed

Like

- C
- C++

Unlike

- JavaScript
- PHP
- Python

- Compilation errors for invalid types
- Parameter, return, and variable types
  - Variable types support inference (statement.let.inference)
- Eliminates
  - Runtime checking (PHP)
  - Checking by tools
  - Manual checking

Rust variables can be constant or mutable

Like

- C
- C++
- JavaScript

Unlike

- PHP
- Python

BUT

- Variables are constant **by default**
- Can be marked as `mut` for mutability

```
fn main () {
        let mut changeable = 1;
        changeable = 2;
        let fixed = 1;
        // fixed = 2; <-- would fail compile
        let fixed = "3"; // <-- new variable
}
```

Rust expressions have values

```
fn assignment() {
        let _val;
        let _result: () = _val = 1;
}

fn block_empty() {
        let _result: () = {};
}

fn block_val() -> i32 {
        let result: i32 = { 5 };
        println!("result is: {}", result);
        result
}
```

- Block result is the final expression (or unit type)
- Includes function bodies

Rust has no increment/decrement operator

Like

- Python

Unlike

- C
- C++
- JavaScript
- PHP

```
error: Rust has no postfix increment operator
 --> src/main.rs:3:8
  |
3 |     val++;
  |        ^^ not a valid postfix operator
  |
help: use `+= 1` instead
  |
3 -     val++;
3 +     val += 1;
  |

error: could not compile `playground` (bin
"playground") due to 1 previous error
```

Rust does not have inline ternary with ?:

Like

- Python*

Unlike

- C
- C++
- JavaScript
- PHP

Python:

```
result = 123 if True else 456
```

Rust:

Result of `if` expression is the executed block (expr.if.result)

```
let result = if true { 123 } else { 456 };
```

Rust does not have `switch`

Like

- Python

Unlike

- C
- C++
- JavaScript
- PHP

But, it does have `match`

...but Rust *does* have `match`

Like

- PHP
- Python

Unlike

- C
- C++
- JavaScript

- Must be exhaustive (rust-lang/reference#798)
- Not mutually exclusive (expr.match.scrutinee-value)
- Can be separated with | (expr.match.or-pattern)
- Can have guards (expr.match.guard)
- `_` can be used as a wildcard (patterns.wildcard.intro)

```
match age {
        ..10 => println!("young"),
        18 | 21 => println!("a new adult in some places"),
        13..=19 => println!("a teenager"),
        10..25 => println!("under 25"),
        25..=100 if age % 10 == 0 => println!("{} years old", age),
        _ => println!("an adult"),
};
```

Book chapter

Rust has custom data structures

Like

- C
- C++
- JavaScript
- PHP
- Python

```
struct User {
        name: String,
        id: u64,
        admin: bool,
}
```

- No inheritance from other structures
  - Like: C
  - Unlike: C++, JavaScript, PHP, Python
- No static fields
  - Like: C
  - Unlike: C++, JavaScript, PHP, Python

Rust has custom data structures... with methods

Like

- C++
- JavaScript
- PHP
- Python

Unlike

- C

```
impl User {
        fn new(name: String, id: u64) -> User { User { name, id, admin: false } }
        fn is_admin(&self) -> bool { return self.admin; }
        fn make_admin(&mut self) { self.admin = true; }
}

fn main() {
        let mut instance = User::new("Daniel".to_string(), 1);
        instance.make_admin();
        println!("Is admin? {}", instance.is_admin());
}
```

- Instance is `self` (like Python)
- Instance needs to be specified as parameter (like Python)
- Indicate mutability or not (like C++)

Rust has enums

Like

- C
- C++
- PHP
- Python*

Unlike

- JavaScript

```
enum GameState {
        PAUSED,
        PLAYING,
}
```

With explicit discriminants (items.enum.discriminant.explicit):

```
enum Direction {
        NORTH = 1,
        EAST = 2,
        SOUTH = 3,
        WEST = 4,
}
```

Rust has tagged unions / enums with data

Unlike

- C
- C++
- JavaScript
- PHP
- Python

```
enum GameAction {
        Pause,
        Quit,
        Teleport{ x: i32, y: i32 },
        Move( Direction, u32 ), // u32 is distance
}
```

- Unit-like variants (items.enum.path-expr)
- Struct-like variants (items.enum.struct-expr)
- Tuple-like variants (items.enum.tuple-expr)

Rust has normal unions

Like

- C
- C++

Unlike

- JavaScript
- PHP
- Python

```
union Value {
        boolean: bool,
        int: u64,
        string: &'static str,
}
```

Note that

- Reading is *always* unsafe (items.union.fields.read-safety)
- Types have restrictions (items.union.field-restrictions)

Rust has generics

Like

- C++
- Python*

Unlike

- C
- JavaScript
- PHP

Option:

```
pub enum Option<T> {
        None,
        Some(T),
}
```

Result:

```
enum Result<T, E> {
        Ok(T),
        Err(E),
}
```

Book chapter on generics

Rust does not have exceptions

Like
- C

Unlike
- C++
- JavaScript
- PHP
- Python

- Can panic to abort program entirely
- Use `Result<T, E>` and try propagation (expr.try)

```
#[derive(Debug)] struct NotEven {}
fn divide_by_two( val: u32 ) -> Result<u32, NotEven> {
    if val % 2 == 1 { return Err(NotEven{}); }
    return Ok(val / 2);
}
fn half_plus_one( val: u32 ) -> Result<u32, NotEven> {
    let half: u32 = divide_by_two(val)?;
    return Ok(half + 1);
}
fn main () {
    println!("(10/2)+1 = {:?}", half_plus_one(10));
    println!("(11/2)+1 = {:?}", half_plus_one(11));
}
```

Rust has traits instead of inheritance

Like

- C++ virtual
- PHP*
- Python ABC

Unlike

- C
- JavaScript

```
struct Demo {}
trait AsCustom {
        fn stringify(&self) -> &'static str;
}
trait AsDefault {
        fn stringify(&self) -> &'static str { "default" }
}
impl AsCustom for Demo {
        fn stringify(&self) -> &'static str { "DEMO" }
}
impl AsDefault for Demo {}
fn main() {
        let obj = Demo {};
        println!("Custom: {}", <Demo as AsCustom>::stringify(&obj));
        println!("Default: {}", <Demo as AsDefault>::stringify(&obj));
}
```

Dedicated book chapter - advanced usage

Rust has a borrow checker

- Memory always has a single owner
- References (including mutable references) are allowed
  - 0 references
  - 1 mutable reference, 0 immutable references
  - 0 mutable references, 1+ immutable references
- Reference rules can be delayed to runtime (RefCell<T>)
- Non-reference parameters are moved (or copied)

Dedicated book chapter

Rust has operator overloads

Like

- C++
- Python

Unlike

- C
- JavaScript
- PHP

- std::ops traits
- Most common: std::ops::Drop for destructors
- Can specialize based on right hand side type
  - Like C++ (which uses function overloading)
  - User + Permission => User with permission
  - User + Preference => User with preference

...and more

- Macros that operate on the syntax tree
  - Book chapter - Reference (macros by example) - Reference - (procedural macros)
- Module system (book chapter)
- Visibility qualifiers - everything is private by default (vis.default)
  - Descendants are allowed to access private items (vis.access)
  - Asymmetric visibility planned (RFC 3323)
- Conditional compilation, e.g. `#[cfg(test)]` (cfg.attr)
- Rustdoc comments and tests (book chapter)
- Smart pointers (book chapter) - Box<T>, Rc<T>, RefCell<T>

Conclusion

Rust

- ...helps you learn with clear error messages
- ...is very well documented (book, reference, standard library)
- ...is most similar to C++
- ...has extra features like tagged unions and pattern matching
- ...has eccentricities like a borrow checker

Thank You

Questions?