Make yourself a new lint

Rust Wrocław

mgr inż. Rafał Chabowski, 27.02.2020

About me

- Magister inżynier
- Software developer

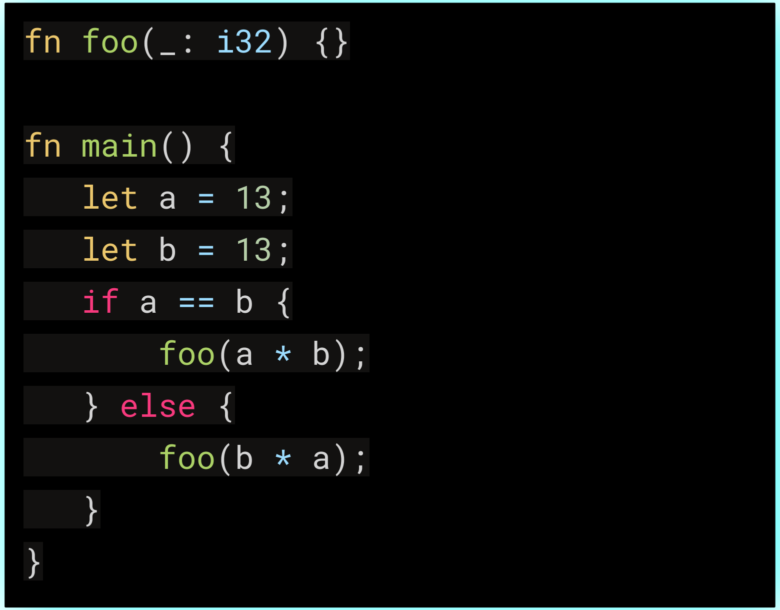
Agenda

- What is Clippy?
- Review some common lints
- Problem with %
- Creating new lint
- Contributing it to the community

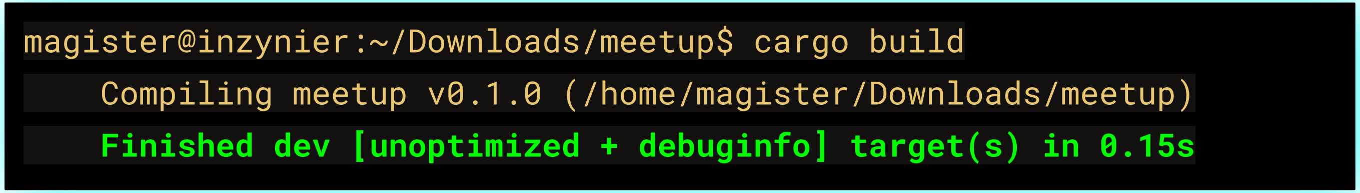
What is Clippy?

- Static code analyzer
- Easily extensible
- More nitpicky than Rust compiler itself
    - `let _ = 2 + 3;`
- 2nd line of support for developers
    - Stays optional, not to drag out the compile times even more
- 350+ lints (and counting)
    - Grouped in several categories

How Clippy can help?

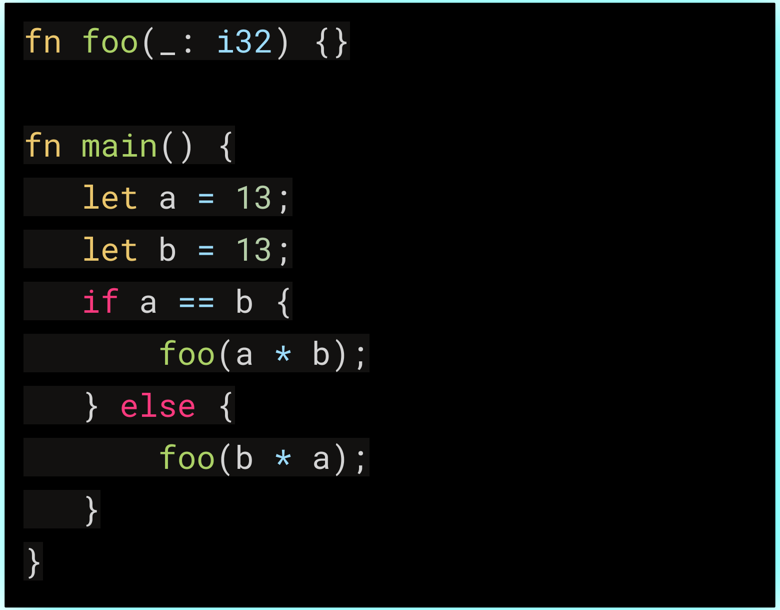


```rust
fn foo(_: i32) {}

fn main() {
    let a = 13;
    let b = 13;
    if a == b {
        foo(a * b);
    } else {
        foo(b * a);
    }
}
```

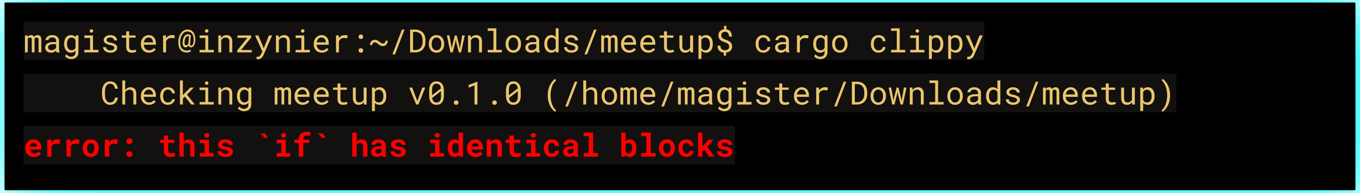


```
magister@inzynier:~/Downloads/meetup$ cargo build
    Compiling meetup v0.1.0 (/home/magister/Downloads/meetup)
    Finished dev [unoptimized + debuginfo] target(s) in 0.15s
```

How Clippy can help?



```rust
fn foo(_: i32) {}

fn main() {
    let a = 13;
    let b = 13;
    if a == b {
        foo(a * b);
    } else {
        foo(b * a);
    }
}
```



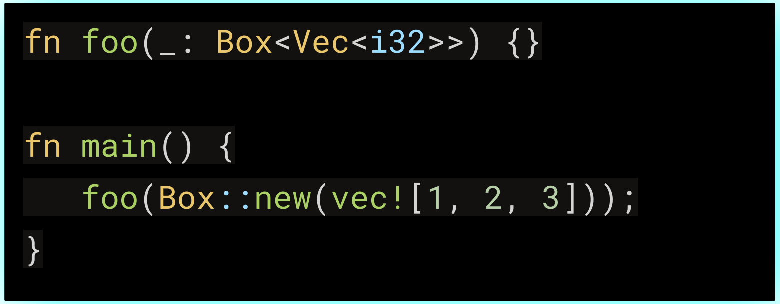
```
magister@inzynier:~/Downloads/meetup$ cargo clippy
    Checking meetup v0.1.0 (/home/magister/Downloads/meetup)
error: this `if` has identical blocks
```

How Clippy can help?

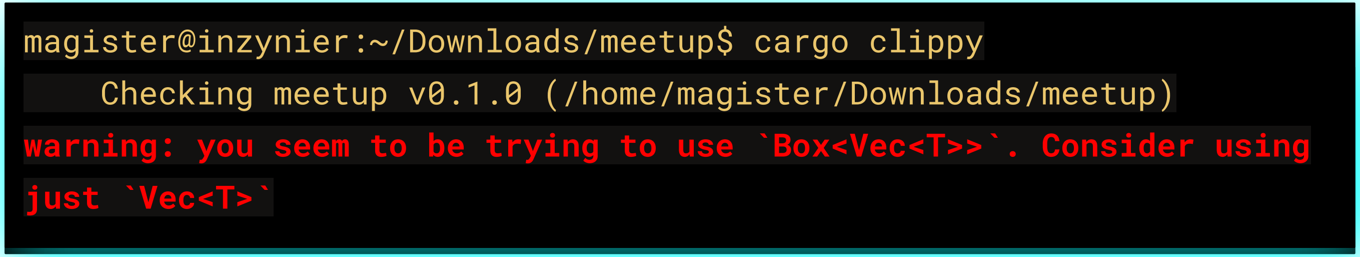
```rust
fn foo(_: Box<Vec<i32>>) {}

fn main() {
    foo(Box::new(vec![1, 2, 3]));
}
```

```
magister@inzynier:~/Downloads/meetup$ cargo build
   Compiling meetup v0.1.0 (/home/magister/Downloads/meetup)
    Finished dev [unoptimized + debuginfo] target(s) in 0.25s
```

How Clippy can help?



```rust
fn foo(_: Box<Vec<i32>>) {}

fn main() {
    foo(Box::new(vec![1, 2, 3]));
}
```



```
magister@inzynier:~/Downloads/meetup$ cargo clippy
    Checking meetup v0.1.0 (/home/magister/Downloads/meetup)
warning: you seem to be trying to use `Box<Vec<T>>`. Consider using
just `Vec<T>`
```

Some “eccentric” lints

- absurd_extreme_comparisons
- blacklisted_name
- integer_arithmetic
- many_single_char_names
- option_option
- suspicious_arithmetic_impl
- trivial_regex
- unsafe_removed_from_name
- cognitive_complexity

```
let _ = Regex::new("^beret");
```

```
let foo = 3.14;
```

```
impl Add for Foo {
    type Output = Foo;

    fn add(self, other: Foo) -> Foo {
        Foo(self.0 - other.0)
    }
}
```

The **%** Problem

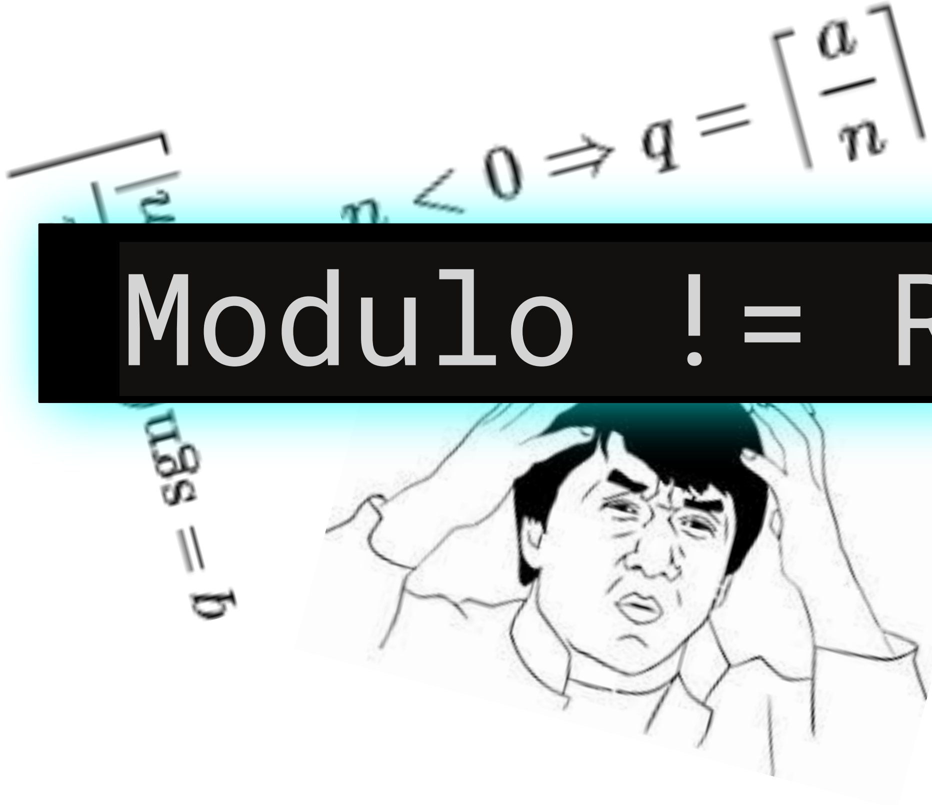
```
>>> print (-17 % 3)
```

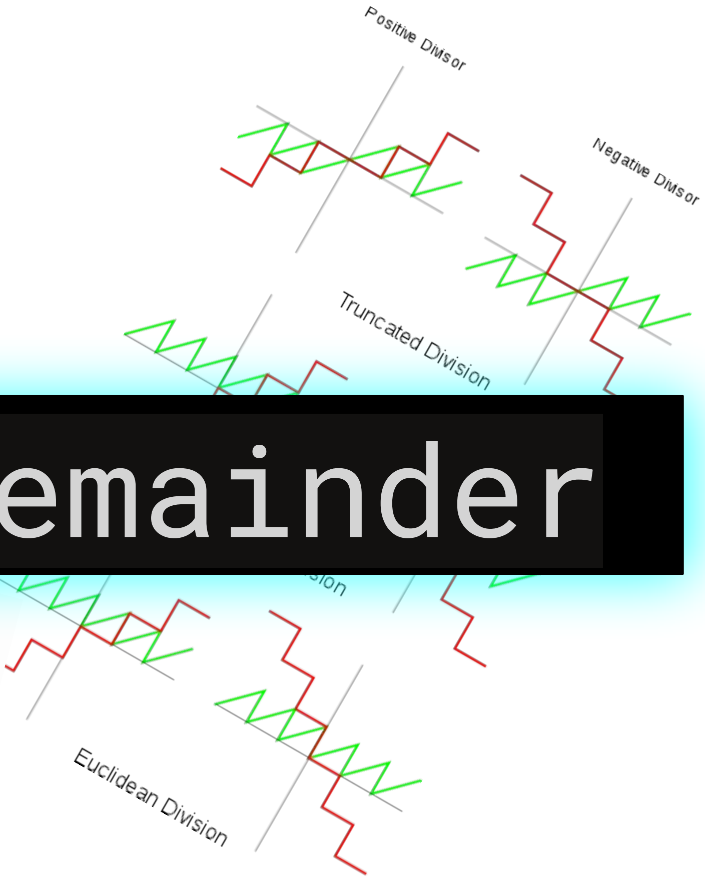
1

```
fn main() {
    println!("{}", -17 % 3);
}
```

-2

The Reason





```
Modulo != Remainder
```

The New Lint

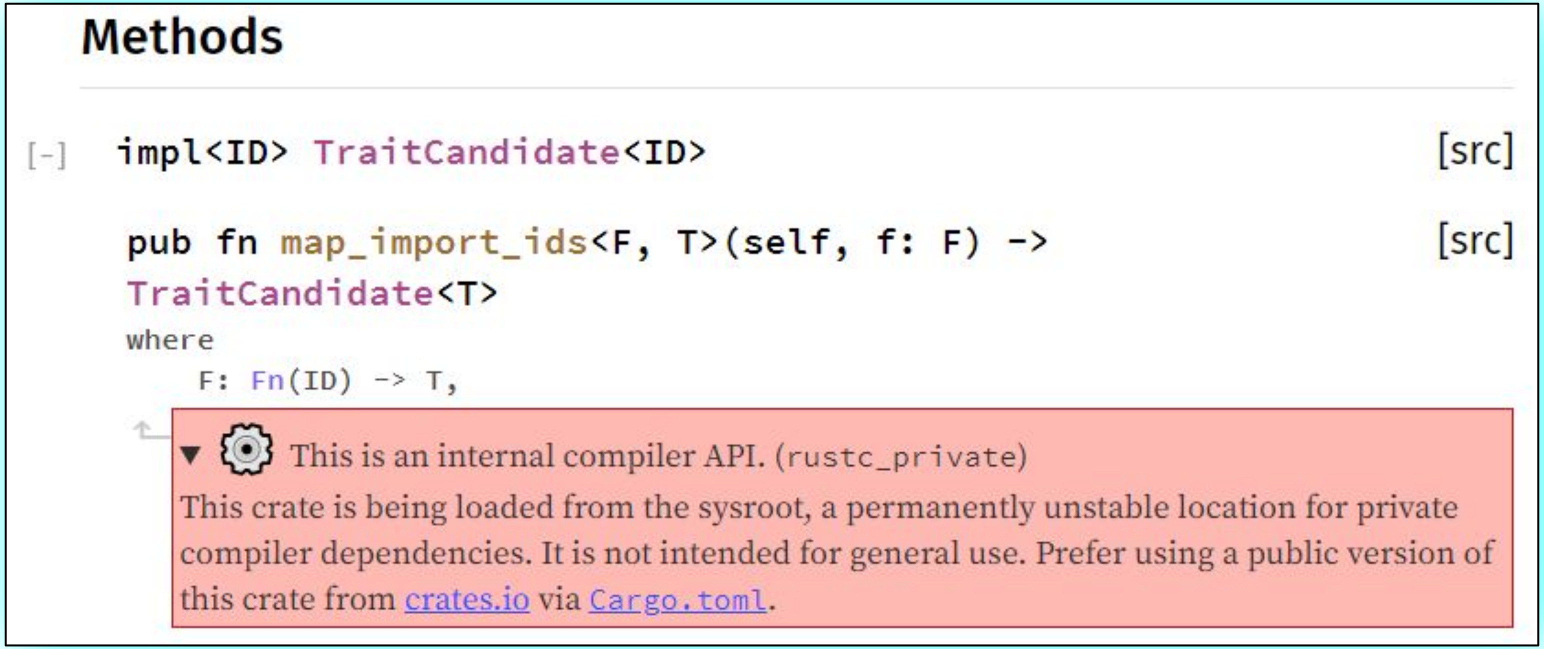
- Why?
  - Learn new things
    - **`rem_euclid()`**, et al
  - Get acquainted with Rust Language internals
  - Contribute
- Is the issue worth it?
  - If it bit you, it can bite others
  - There are plenty of even simpler lints, after all
  - You can make a presentation about it :-)

How to Proceed?

- CONTRIBUTING.md
- Build and test Clippy locally
    - Use Linux
    - Use latest Rust from **`master`** branch
- Get acquainted with “uitests”
- Read source code of other lints
    - …or “The Rust Unstable Book”
- Have patience
- Ask questions
    - Rust community is super supportive

Read source of other lints

Documentation is scanty



Methods

[-] `impl<ID> TraitCandidate<ID>` [src]

```
pub fn map_import_ids<F, T>(self, f: F) ->
TraitCandidate<T>
where
    F: Fn(ID) -> T,
```
[src]

▼ This is an internal compiler API. (`rustc_private`)

This crate is being loaded from the sysroot, a permanently unstable location for private compiler dependencies. It is not intended for general use. Prefer using a public version of this crate from crates.io via `Cargo.toml`.

cargo uitest

- Compile & execute test
- Capture standard output
- Compare it with provided **`.stderr`** file
- Fix the code (cargo fix)
- Compare fixed code with provided **`.fixed`** file

```
error: casting `f32` to `i32` may truncate the value
  --> $DIR/cast.rs:21:5
   |
LL |     1f32 as i32;
   |     ^^^^^^^^^^^
   |
   = note: `-D clippy::cast-possible-truncation` implied by `-D warnings`
```

Clippy engine

- Clippy is a Rust compiler plugin
- Rust compiler calls into Clippy while crunching code
- Calls are made through two alternative lint traits
  - EarlyLintPass
    - AST info only
  - LateLintPass
    - Same as above, but full type information is available
- Provide own implementation of **`check_*()`** functions
  - **`check_param()`**
  - **`check_expr()`**
  - **`check_fn()`**
  - *…a lot more*

Developer utilities

- cargo dev new_lint
  - Creates boilerplate for new lint for you
- cargo dev update_lints
  - Updates Clippy code that references your lint
- cargo dev fmt
  - Formats code (stable formatter might not be available on **`master`**)
- tests/ui/update-all-references.sh
  - Compiles / executes all tests and updates the **`.stderr`** files
- // run-rustfix
  - Yes, it is a comment :)
  - If included in test, the **`.fixed`** file will be created as well
- **`#[clippy::author]`**

The **`modulo_arithmetic`** lint

- First assumption
  - Do a naive implementation inspired by **`integer_arithmetic`** lint
  - i.e. simply check for any modulo operations

```
impl<'a, 'tcx> LateLintPass<'a, 'tcx> for ModuloArithmetic {
   fn check_expr(&mut self, cx: &LateContext<'a, 'tcx>, expr: &'tcx Expr<'_>) {
       match &expr.kind {
           ExprKind::Binary(op, lhs, rhs) => {
               if let BinOpKind::Rem = op.node {
                    LINT_HERE!
               };
           },
           _ => {},}}}
```

The **`modulo_arithmetic`** lint

- but wait, there is an assignment operation as well
- **`x %= -3;`**

```rust
impl<'a, 'tcx> LateLintPass<'a, 'tcx> for ModuloArithmetic {
   fn check_expr(&mut self, cx: &LateContext<'a, 'tcx>, expr: &'tcx Expr<'_>) {
         match &expr.kind {
               ExprKind::Binary(op, lhs, rhs) | ExprKind::AssignOp(op, lhs, rhs) => {
                     if let BinOpKind::Rem = op.node {
                            LINT_HERE!
                     };
               },
               _ => {},}}}
```

The Pull Request

- Lint naming convention
- Lint documentation
- **`cargo test`** passess locally
- All references to lint are updated (developer utilities)
- Code is formatted
- Dogfooding
- Solve conflicts
  - At least two are almost guaranteed :)

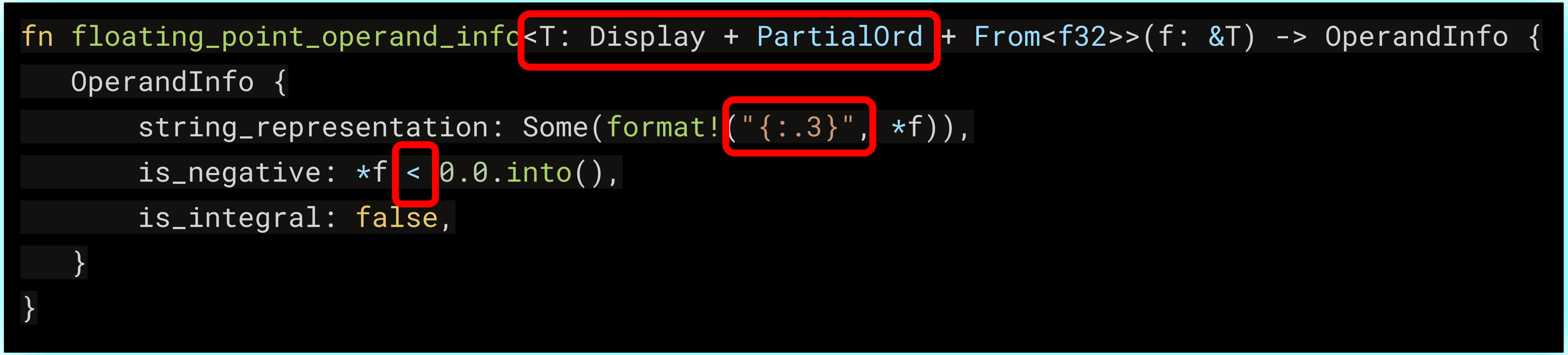
The **`modulo_arithmetic`** lint

- Here comes the review and discussion
  - Consider no linting on constants when both sides are of the same sign
  - Let’s access the values of the operands

```
match constant(cx, cx.tables, operand) {
    Some((Constant::Int(v), _)) => match cx.tables.expr_ty(expr).kind {
        ty::Int(ity) => { let value = sext(cx.tcx, v, ity); },
        ty::Uint(_) => { // Cannot be negative },
        _ => {},
    },
    Some(Constant(floating_point)) => {...}
    _ => {},
}
```

The **`modulo_arithmetic`** lint

- Tackle both floating point types

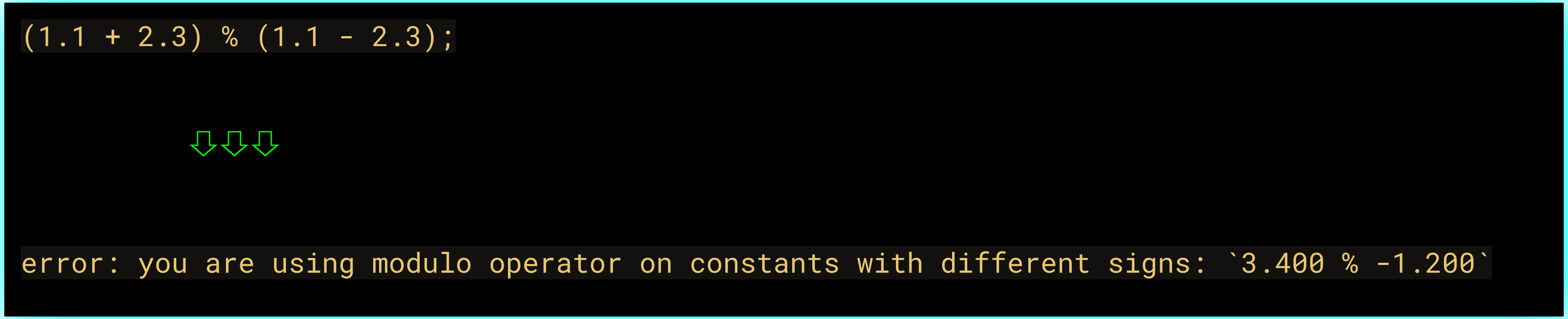


```
fn floating_point_operand_info<T: Display + PartialOrd + From<f32>>(f: &T) -> OperandInfo {
    OperandInfo {
        string_representation: Some(format!("{:.3}", *f)),
        is_negative: *f < 0.0.into(),
        is_integral: false,
    }
}
```

mgr inż. Rafał Chabowski, 27.02.2020

The **`modulo_arithmetic`** lint

- What we have so far
  - Detection of expressions that involve modulo arithmetic
  - Knowledge whether the operands are constants
  - The exact values (in form of human readable strings), in case they are constants
- What we can do
  - Show a nice lint



```
(1.1 + 2.3) % (1.1 - 2.3);

        ⇩⇩⇩

error: you are using modulo operator on constants with different signs: `3.400 % -1.200`
```

The non-const case

- What to do if the value is not known at compile time?
    - Simply check if any of the operands **might** have negative value

```
fn might_have_negative_value(t: &ty::TyS<'_>) -> bool {
    t.is_signed() || t.is_floating_point()}
```

    - And provide less detailed lint message

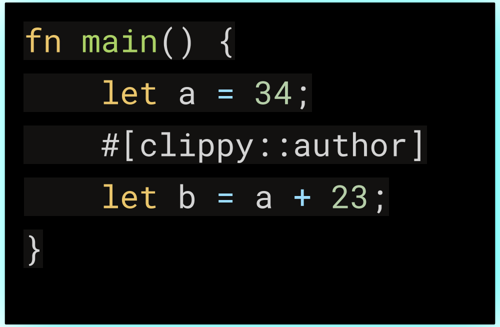
```
b_f32 %= a_f32;

      ⇩⇩⇩

error: you are using modulo operator on types that might have different signs
```

That’s it

- Wait for more review comments
- And eventually get merged into **`master`**
- Get famous :D
- And check this page:

https://thanks.rust-lang.org/

#[clippy::author]



```rust
fn main() {
    let a = 34;
    #[clippy::author]
    let b = a + 23;
}
```

```rust
if_chain! {
    if let StmtKind::Local(ref local) = stmt.kind;
    if let Some(ref init) = local.init;
    if let ExprKind::Binary(ref op, ref left, ref right) =
init.kind;
    if BinOpKind::Add == op.node;
    if let ExprKind::Path(ref path) = left.kind;
    if match_qpath(path, &["a"]);
    if let ExprKind::Lit(ref lit) = right.kind;
    if let LitKind::Int(23, _) = lit.node;
    if let PatKind::Binding(BindingAnnotation::Unannotated, _,
name, None) = local.pat.kind;
    if name.as_str() == "b";
    then {
        // report your lint here
    }
}
```

Additional considerations

- Lint **span**
  - Descriptor of the offending part of code
- Lint **sugg**estion (applicability)
  - A way to solve the issue
- Cargo fix
  - A tool that will automatically apply **sugg** to **span**

THE END