Abusing Type Annotations

Zach Mitchell
tinkering.xyz

Rust is *really* nice

- Fast
- Useful type system
- Memory safe

but my favorite feature is...

Rust Procedural Macros

Code -> AST -> [Your Macro Here] -> New AST

Great for:

- Code generation
- Reducing boilerplate
- Custom syntaxes or shorthands
- etc.

Rust Procedural Macros

Generate CLI parsing code by defining a struct

```
#[derive(StructOpt, Debug)]
#[structopt(name = "basic")]
struct Opt {
    // Enable debug mode
    #[structopt(short = "d", long = "debug")]
    debug: bool,

    // Set verbosity
    #[structopt(short = "v", long = "verbose", parse(from_occurrences))]
    verbose: u8,

    // Output file
    #[structopt(short = "o", long = "output", parse(from_os_str))]
    output: PathBuf,
}
```

Rust Procedural Macros

...or do something slightly less useful

tinkering.xyz/introduction-to-proc-macros

```rust
#![feature(proc_macro)]

extern crate wickerman;
use wickerman::wickerman;

#[wickerman]
struct Foo(i32);

#[wickerman]
struct Bar {
    baz: i32,
    bees: String,
error:
        🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝
        🐝 not the bees!!! NOT THE BEEEEEES!!! 🐝
        🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝🐝
}

fn main() {
    println!("Hello, world!");
}
```

Can I Do This In Python?

Can I Do This In Python?

- Decorators get you half-way there
- The **ast** module lets you do more
- But how do I attach things to variables?

Type Annotations

```
class MyClass:
    foo: SomeRealType
    bar: "this can be anything"
```

Type? No!
That's an arbitrary string attached to your variable!

Great Learning Experience!

- Diving into the data model
- Learning about descriptors
- Abstract syntax trees
- 🤓

First CPython Contribution!

```
771 -   :attr:`__self__` attributes is :class:`C`.  When it would yield a static
    771 +   :attr:`__self__` attribute is :class:`C`.  When it would yield a static
```

Ex.) Generate Properties

```
@inrange
class MyClass:
    foo: "0 < foo < 3"  # <- generates a property
    bar: int  # <- normal class variable
```

Each ***instance*** gets a property that only accepts values in the specified range

Ex.) Notify On Write

```
@notify
class MyClass:
    def __init__(self, x):
        self.x: "this one" = x  # will be notified
        self.y = True  # won't be notified
```

Notifies the user when **MyClass.x** is set

Live Demo!