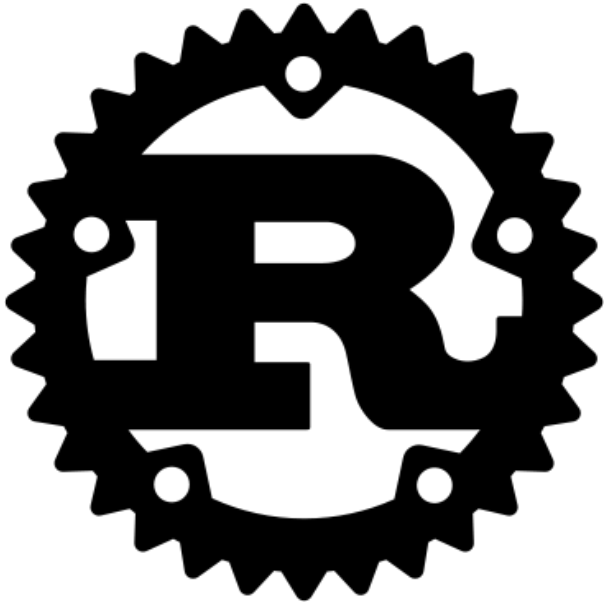
Enumerated data types in Python

Simon Sapin

PyCon UK, 2013-09-22



rust-lang.org

It’s nice!

Type theory

```
int
float
str
NoneType
```

Composition

Product: `A × B`

C, Rust:

```
struct Point { x: float, y: float }
```

Python: tuple, namedtuple, objects, ...

Composition

Sum: `A + B` a.k.a. *enumerated data type*

Type algebra:

```
NoneType = 1
A × 1 = A
bool = 1 + 1 = 2
A + A = A × 2
```

C: *tagged union*

```
enum ShapeKind { Circle, Rectangle };
struct Shape {
  enum ShapeKind kind;
  union {
    struct {Point center; float radius}
      circle;
    struct {Point tl; Point br}
      rectangle
}};
```

Rust: enum

```
enum Shape {
    Circle(Point, float),
    Rectangle(Point, Point)
}
```

```
match shape {
  Circle(center, 0) => {...},
  Circle(center, radius) => {...},
  Rectangle(tl, br) => {...},
}
```

Python?

- PEP 435 Enum: like C, not like Rust
- Dynamic typing
- Object oriented: class hierachy, `isinstance()`, `.type` class attr
- Tuples: `('circle', x, y, r)`

  `('rectangle', x1, y1, x2, y2)`

Can we do better?

Another pattern in current Python?
Adding a `match` statement?

Discuss :)

@SimonSapin