さうすの Rust 勉強会

lesson01

- 目標：Rust の基礎文法をある程度理解する
  - 今日全部理解しなくてももちろん大丈夫 (これからまた同じ概念が出てくるとき聞いても ◎)
- 困ったら The Book (日本語 / English) を読みましょう
  - これ読んだら今回の講義はやらなくていい
- 質問はいつでも Slack でどうぞ (授業中でも授業中じゃなくても)
  - 答えられなくても調べる

前提

- Rust は結構複雑な言語なのですべてを理解しなくてよい
- とりあえず最低限必要なものを習得して、作りたいものを作れるようになろう

環境構築

- `rustup` をインストールする
- たぶんデフォルト `rustup install stable` やってくれるので大丈夫
- VSCode 使うなら Rust Analyzer 入れとこう
    - VSCode 使わなくても入れる方法あれば入れとこう

基礎文法

もう例を示して終わったことにする。

```rust
// bin crate だと main 関数が使える
fn main() {
    println!("Hello, world!");
}

// unit / (): 他の言語の void と大体同じonaji
// unit の場合書かなくてもよい
fn hello_world() -> () {
    // println! ← マクロである
    let world = "world";
    println!("Hello, {}!", world);
}
```

```rust
fn add(a: i32, b: i32) -> i32 {
    // こういうブロックを使って計算することもできる
    let result = {
        let sum = a + b;
        sum
    };
    // 最後の一文は return 書かなくていい
    result
}

// 一番普通の書き方
fn add2(a: i32, b: i32) -> i32 {
    a + b
}
```

Struct, Enum

```
struct S1 {
    a: i32,
    b: i32
}

// フィールドがない struct
struct S2;

struct S3(i32);

fn main() {
    let s1 = S1 { a: 5, b: 6 };
    let s2 = S2;
    let s3 = S3(7);
}
```

```
enum E1 {
    A,
    B
}

enum E2 {
    X(i32),
    Y {
        z: i32
    }
}

// ケースがないので作れない
enum E3 {}

fn main() {
    let e1_1 = E1::A;
    let e1_2 = E1::B;
    let e2_1 = E2::X(32);
    let e2_2 = E2::Y { z: 64 };
}
```

所有権

一番大事な部分：

> - Rust の各値は、所有者と呼ばれる変数と対応している。
> - いかなる時も所有者は一つである。
> - 所有者がスコープから外れたら、値は破棄される。
> - 任意のタイミングで、一つの可変参照か不変な参照いくつでものどちらかを行える。
> - 参照は常に有効でなければならない。

https://doc.rust-jp.rs/book-ja/ch04-00-understanding-ownership.html

```rust
#[derive(Debug)] struct T { inner: i32 }
fn main() {
    let t1 = T { inner: 5 };
    let t2 = t1; // OK: move
    // println!("{:?}", t1); // NG: moved

    let t3 = T { inner: 10 };
    let t4 = &t3; // OK: borrow
    let t5 = &t3; // OK: borrow
    println!("{:?}", *t4); // OK: readonly dereference
    // let t6 = t3; // NG: cannot move

    let mut t7 = T { inner: 15 };
    let t8 = &mut t7; // OK: mutably borrow
    t8.inner = 20; // OK: mutate using mutable reference
    println!("{:?}", &t7);
}
```

```
T { inner: 10 }
T { inner: 20 }
```

match 式

https://doc.rust-jp.rs/book-ja/ch06-02-match.html

```
enum T {
    A(i32),
    B(i64),
}

fn main() {
    let t: T = T::A(5);
    match t {
        T::A(a) => { println!("A: {}", a); },
        T::B(b) => { println!("B: {}", b); },
    }
}
```

impl

https://doc.rust-jp.rs/book-ja/ch05-03-method-syntax.html

```
struct S { a: i32 }

impl S {
    fn double(self) -> S {
        S { a: self.a * 2 }
    }
    fn read(&self) -> &i32 {
        &self.a
    }
    fn add1(&mut self) {
        self.a += 1;
    }
}
```

trait（トレイト）

今回は無視しても大丈夫。https://doc.rust-jp.rs/book-ja/ch10-00-generics.html 読んでね。

```
trait T {
    fn t() -> i32;
}

struct S;

impl T for S {
    fn t() -> i32 {
        42
    }
}
```

ジェネリック

https://doc.rust-jp.rs/book-ja/ch10-01-syntax.html

```
struct S1<T> {
    s: T,
}

struct S2<T: Copy> {
    t: T,
}

fn main() {
    let s11 = S1 { s: "test".to_string() };
    let s12 = S1 { s: 1 };
    let s21 = S2 { t: 2 };
    // let s22 = S2 { t: "test".to_string() };
}
```

Option, Result

めちゃくちゃ使われる enum である。

https://doc.rust-jp.rs/book-ja/ch06-01-defining-an-enum.html#option-enumとnull値に勝る利点

https://doc.rust-jp.rs/book-ja/ch09-02-recoverable-errors-with-result.html

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

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

Vec

可変長配列。

```
fn main() {
    // ここで型を書かないとなんの Vec なのか推論できない
    let empty: Vec<i32> = vec![];

    let numbers = vec![1, 2, 3];
    println!("{}", numbers[1]);

    let mut words = vec!["happy", "birthday"];
    words.push("to");
    words.push("you");
    println!("{:?}", &words);
}
```

Iterator

`.iter()`, `.iter_mut()` または `.into_iter()` で作るもの。

```
fn main() {
    let numbers = [1, 2, 3];
    numbers
        .iter()
        .map(|number| number * 2)
        .filter(|number| number >= 4)
        .for_each(|number| {
            println!("{}", number);
        });
}
```

derive

proc-macro で実装されている。

```
#[derive(Copy, Clone, Debug)]
struct D { a: i32, b: i32 }

struct E { a: i32, b: i32 }

fn main() {
    let d = D { a: 5, b: 6 };
    let e = E { a: 5, b: 6 };
    let d2 = d; // OK: copy
    let d3 = d.clone(); // OK: clone
    println!("{:?}", d); // OK
    let e2 = e; // OK: move
    // let e3 = e.clone(); // NG: Clone trait がない
    // println!("{:?}", e); // NG: Debug trait がない
}
```

pub / pub(crate) / pub(super)

https://doc.rust-jp.rs/rust-by-example-ja/mod/visibility.html

モジュールの話だけど、今はあんまり気にしなくてもよい

```
/// crate 外からアクセスできる
pub fn fn1() {}

/// crate 内からしかアクセスできない
pub(crate) fn fn2() {}

/// 親モジュールからしかアクセスできない
pub(super) fn fn3() {}

/// 今のモジュールからしかアクセスできない
fn fn4() {}
```

serde

serde は serialize-deserialize の略で、ライブラリの名前でもある。オブジェクト ↔️ いろんな形式 (JSON, YAML など) の変換のためのライブラリである。

```
use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize, PartialEq)]
struct S { a: i32, b: i32 }

fn main() {
    let s = S { a: 5, b: 6 };
    println!("{}", serde_json::to_string(&s).unwrap());
    // {"a":5,"b":6}
    let s2: S = serde_json::from_str(r#"{"a":7,"b":8}"#).unwrap();
    println!("{}", &s2);
    assert!(s2 == S { a: 7, b: 8 });
}
```

async-await, futures (tokio)

（ async-await 使うからスレッドあんまり気にしなくていいはず…？ ）

https://rust-lang.github.io/async-book/ を呼んでもよいが、今は `async` 関数はそのあと `.await` と書かないといけなくて、あと `async` 関数からしか呼べないぐらいの理解で大丈夫。

```
async fn async_fn(a: i32, b: i32) -> i32 {
    // async 関数を呼び出すために await を使う
    tokio::time::sleep(tokio::time::Duration::from_millis(500)).await;
    a + b
}

#[tokio::main] // main 関数を async にするための
async fn main() {
    let sum = async_fn(250, 250).await;
    println!("{}", sum);
}
```

課題

> **ポイント**
>
> - 協力してもよい！
> - もう完全に理解してたら他の人に教えよう！
> - わからないときは自由に Slack のチャンネルで聞こう！

- まず `project/` を埋めて、テストが通るようにしよう
  - きれいな解を書くように心がけよう。（時間があれば）書いたあと他の人のコードを比べてみよう。
  - あとで `cargo clippy` 実行するといい。

- 次に Rust で以下の問題を解こう
  - コードは汚くてもいいので、Rust 使える〜という肌感覚をつけてほしい (すぐには無理だが)
  - https://atcoder.jp/contests/language-test-202301/tasks/abc086_a
  - https://atcoder.jp/contests/language-test-202301/tasks/abc081_a
  - https://atcoder.jp/contests/language-test-202301/tasks/abc081_b
  - https://atcoder.jp/contests/language-test-202301/tasks/abc083_b