Rustに触れる１

この資料はRustについて言語的に解説しています。

変数の作成

まずはじめに変数についてです。Rustでは基本、変数を宣言する際、以下のように型を指定しません。

```
fn main()
{
    let int_value = 100;
    let float_value = 3.14;
    let str_value = "Hello, 力士"
}
```

Rustで変数を宣言するときは***let***というワードを用います。これをすることによってその変数が定数であることを定義します。つまりその変数は不変となるのです。

```
fn main()
{
    let value = 5;
    value = 6;
}
```

例えば上のコードを動かそうとするとコンパイルエラーとなります。Rustでは通常変数を不変としているのです。

可変可能な変数は以下のように宣言します。

```
fn main()
{
    let mut value = 5;
    println!("hello, {}", value); // 「hello, 5」
    value = 6;
    println!("hello, {}", value); // 「hello, 6」
}
```

関数の作成

度々見えていたのですが関数の作成方法です。

```
fn main()
{

}
```

ずっと登場していたメイン関数です。ソースコードを実行したときにこの関数が呼び出されます。

次にユーザ関数です。

```
fn main()
{
    let print_value = 123.456;

    original_print(print_value);
    //  「I'm 123.456 years old.」
}

fn orignal_print(value : f32)
{
    println!("I'm {} years old.", value);
}
```

これは引数があるタイプで「***引数の名前 : 引数の型***」という形で宣言します。

```
fn main()
{
    let prev = 10;

    let result = plus_one(prev);

    println!("10 + 1 = {}", result);  //  「10 + 1 = 11」
}

fn plus_one(value : i32)->i32
{
    value + 1
    // return value + 1;  と同じ
}
```

Rustでは基本「；（セミコロン）」をつけないといけないのですが、つけなかった場合にはその関数や演算の結果を***return***するという仕組みがあります。

よって***plus_one***関数ではreturnがないのにも関わらず返り値として***i32***を返せている。

返り値は上述したように関数の引数を書いたあとに「->」をつけたあとに型を指定してあげることでOKです。

Rustには**タプル**というものがあります。まずはコードをご覧ください

```
fn main()
{
    let new_value = 100;

    let (b, a) = before_and_after(new_value);

    println!("before is {}, after is {}", b, a);
    // 「before is 99, after is 101」
}

fn before_and_after(value : i32)->(i32, i32)
{
    let before = value - 1;
    let after = value + 1;

    (before, after)
}
```

このように（）を用いることで複数の値を用いることができます。返り値と受ける変数があっていれば使うことができます。また同じ内容を以下のように書き換えることも可能です。

```
fn main()
{
    let new_value = 100;

    let result = before_and_after(new_value);

    println!("before is {}, after is {}", result.0, result.1);
    // 「before is 99, after is 101」
}

fn before_and_after(value : i32)->(i32, i32)
{
    let before = value - 1;
    let after = value + 1;

    (before, after)
}
```

タプルの要素の数が増えても２，３，４というように増やすだけで扱うことができます。