구조체

2020.3

구조체의 정의

```
struct User {
    username: String,
    email: String,
    sign_in_count: u64,
    active: bool,
}
```

필드가 이름을 갖는 점에서 튜플과 다르다.

구조체의 사용

- 사용하려면 인스턴스를 생성해야함.

```
let user1 = User {
    email: String::from("someone@example.com"),
    username: String::from("someusername123"),
    active: true,
    sign_in_count: 1,
};
```

생략된 초기화

```
fn build_user(email: String, username: String) -> User {
    User {
        email,  // email: email,
        username,  // username: username,
        active: true,
        sign_in_count: 1,
    }
}
```

구조체 갱신법

```
let user2 = User {
    email: String::from("another@example.com"),
    username: String::from("anotherusername567"),
    active: user1.active,
    sign_in_count: user1.sign_in_count,
};
```

```
let user2 = User {
    email: String::from("another@example.com"),
    username: String::from("anotherusername567"),
    ..user1
};
```

튜플 구조체

- 튜플과 다른점: 메소드를 구현 할 수 있다.

```
struct Color(i32, i32, i32);
struct Point(i32, i32, i32);

let black = Color(0, 0, 0);
let origin = Point(0, 0, 0);

Struct MyEmptyStruct;  // trait 을 정의하고 플 때 사용.
```

구조체 데이터의 소유권

```
struct User {
    username: &str, // 이 슬라이스가 가리키는 데이터가 언제까지 살아있는지 컴파일러가 알 수 없음
    email: &str, // &’static str 인 경우 컴파일 됨. 이렇게 선언되면 사용할때 static 만 할당할 수 있음.
    sign_in_count: u64,
    active: bool,
}

fn main() {
    let user1 = User {
        email: “someone@example.com",  // 이 경우는 ‘static이지만 항상 이렇게 사용된다는 보장이 없음.
        username: "someusername123",
        active: true,
        sign_in_count: 1,
    };
}
```

Trait 상속 받기

```
#[derive(Debug)]
struct Rectangle {
    length: u32, // 사각형과 관계된 데이터이므로 struct로 묶어 준다.
    width: u32,
}

fn main() {
    let rect1 = Rectangle { length: 50, width: 30 };

    println!("rect1 is {:?}", rect1); // Debug trait가 필요한 문장.
}
```

{:?}, {:#?}

메소드

- 함수와 비슷, struct, enum, trait에서만 사용됨.
- 첫 파라메터는 self 또는 &self, &mut self

```
#[derive(Debug)]
struct Rectangle {
    length: u32,
    width: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.length * self.width
    }
}

fn main() {
    let rect1 = Rectangle { length: 50, width: 30 };

    println!(
        "The area of the rectangle is {} square pixels.",
        rect1.area()
    );
}
```

자동 참조

object.something()이라고 메소드를 호출했을 때,
러스트는 자동적으로 &나 &mut, 혹은 *을 붙여서
object가 해당 메소드의 시그니처와 맞도록 합니다.

```
p1.distance(&p2); // p1 —> &p1으로 자동 변환 합니다.
                  // 메소드의 시그니처를 참조해서 가능합니다.
(&p1).distance(&p2);
```

```
impl Rectangle {
    fn square(size: u32) -> Rectangle { // self 가 없으면 연관함수 라고 부릅니다.
        Rectangle { length: size, width: size }
    }
}
```