rust

Chapter 1

Cargo

1.cargo “project name” : 프로젝트 생성

2.cargo build : 프로젝트 빌드

3.cargo run: 빌드 후 실행까지

4.cargo check: 컴파일 되는지 확인하는데 실행 파일은 생성하지 않음

Chapter2

mutable

let x = 5; 이케 하면 나중에 x=6; 이렇게 수정이 안됨

let mut x; 이렇게 선언하거나

[

let x = 10;

let x = 1;

](Shadowing)

이런식으로 해야함.. 기본이 불변;

상수

불변하다. mut를 사용 할 수 없음..

Shadowing

이름을 동일하게 사용하는 것(2장에서 이 기능을 많이 사용하게 될 것이라 하였음)

let x=5; 이후 x=8;이 불가능 하지만

let x=5; 이후 let x=9; 이 가능함(mut 쓰지 않고도 값 변경하는 느낌이나 mut와는 개념이 조금 다른듯 싶다)

변수의 이름을 덮어 버리거나 또는 ***일정 범위가 종료될 때 까지 변수 이름을 사용하지 않는다*** 라는 의미

⇒

fn main() {
let x = 5;

```
let x = x + 1; //x는 6 { let x = x * 2; println!("The value of x in the in
ner scope is: {x}"); x는 12로 출력된다.. }//일정 범위가 종료될 때 까지 변
수 이름 사용 ✗ println!("The value of x is: {x}"); //x는 6으로 출력된다..
```

}

만약 데이터의 타입이 바뀌는

ex)

let spaces = "   ";
let spaces = spaces.len();

shdowing은 가능하지만 mut를 사용한다면 데이터의 타입이 다르기 때문에 컴파일 에러가 난다.

⇒ mut와 비슷해 보이지만 조금 더 편리하다.

데이터의 유형

input은 거의 string으로 받는거 같음. input을 num으로 하고싶을 때 string을 num으로 형 변환을 해야함

let guess: u32 = "42".parse().expect("Not a number!");

let 변수명: 타입 = 변수.p.e;

이 타입을 명시하지 않으면 컴파일 되지 않음.

스칼라

러스트에는 정수, 부동소수점 숫자, 부울, 문자 등 네 가지 기본 스칼라 유형이 있음

▼ 정수

signed 경우 -2^(n - 1) ~ 2^(n - 1) - 1 사이의 수 표현 (n은 bit 수)

unsigned 경우 0에서 2^(n) - 1 사이의 수 펴현

i8 ,16, 32, 64, 128, size ⇒signed
u8 ,16, 32, 64, 128, size ⇒ unsigned

▼ 넘버 리터럴(10진수, 2진수, 8진수 .....)

Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
Byte (`u8` only)	`b'A'`

98_222 는 98222로 _는 큰 수를 읽기 편하도록 구분하는 문자임..

Byte b'문자' 로 아시키 코드 수 의미

▼ 부동소수점

f32, f64 , 기본 64bit

let a: f32=5.5;//32bit로 선언하려면

▼ boolean

let f: bool = false;

▼ char

let a: cahr = 'z';

Compound

▼ 튜플

한 번 선언하면 크기가 커지거나 줄어들 수 없습니다.

let tup: (i32, f64, u8) = (500, 6.4, 1);

let (x, y, z) = tup;

or tup.0, tup.1, tup.2

▼ 배열

튜플과 달리 데이터 유형이 같아야함

let 이름: [타입; 개수] = [];

let 이름 = [초기화 값; 개수];

함수

fn 이름(){ }

▼ 매개 변수

(이름: 타입, 이름: 타입....)

```
fn main() {
print_labeled_measurement(5, 'h');
}

fn print_labeled_measurement(value: i32, unit_label: char) {
println!("The measurement is: {value}{unit_label}");
}
```

??

fn main() {
let x = (let y = 6);
}

이거 컴파일 오류 남.

let은 반환 하는 값이 없어요.

fn main() {
let y = {
let x = 3;
x + 1 //세미콜론 없음, 세미콜론 붙이면 반환값이 없어져서 y 할당이 안됨..
};

y는 4가 됨

반환 값이 있는 함수

fn 이름() → 반환 값 형태{

}

fn five() -> i32 {
5
}

fn main() {
let x = five();

//x는 5가 됨

}

반환 하는 거는 ;이 없어염

제어 흐름

if

```
let number = 3; if number < 5 { println!("condition was true"); } else
{ println!("condition was false"); }
```

if 조건{}

else

이때 조건은 무조건 bool타입이어야 한다.

let x=5;

if x{} 이렇게 하면 컴파일 안됨. (c나 c++과는 다름)

ex)let number = if condition { 5 } else { 6 };

이때 반환 값이 같은 유형이어야 한다..

반복

loop{}

값 반환

break 이후에 반환 값 추가..

```
let result = loop { counter += 1; if counter == 10 { break counter *
2; } };
```

result는 20이 된다

▼ 루프 , 루프

fn main() {
let mut count = 0;
'counting_up: loop {
println!("count = {count}");
let mut remaining = 10;

```
loop { println!("remaining = {remaining}"); if remaining == 9 { break;
} if count == 2 { break 'counting_up; } remaining -= 1; } count += 1;
} println!("End count = {count}");
```

}

If you have loops within loops, `break` and `continue` apply to the innermost loop at that point. You can optionally specify a *loop label* on a loop that you can then use with `break` or `continue` to specify that those keywords apply to the labeled loop instead of the innermost loop. Loop labels must begin with a single quote. Here's an example with two nested loops:??

loop에 이름 부여 가능함.

'이름:loop{

}

break; 하면 그때 가장 안쪽의 loop 종료하는데

break '(loop 이름); 하면 해당 loop을 종료한다.

▼ while

```
let mut number = 3; while number != 0 { println!("{number}!"); number
-= 1; }
```

number 3 → 2 →1 →0일때 종료

while 조건 {}, 조건이 만족하는 동안 실행

▼ for

```
let a = [10, 20, 30, 40, 50]; for element in a { println!("the value is: {element}"); }
```

for number in (1..4).rev(){} 뒤에서 부터 4 → 3→ 2→...

질문쓰

1.println 시에 {:?} 이거 무슨 의미이지?

디버그 형식 → 벡터 출력시에 사용해야 출력이뎀

{:#?} 디버그 형식. 디버그 정보를 더 읽기 쉽게 여러 줄로 출력

{:.소수점 자리}

{:x}, {:X} 이거 16진수 출력 x는 소문자, X는 대문자

{:b} 이진수

{:o}8진수

{:p} 주소값

`{:<width}` / `{:>width}` / `{:^width}`: 출력 너비를 지정하여 좌측, 우측, 가운데 정렬을 합니다.

println!("{:<5}", "hi"); // 출력: "hi   "
println!("{:>5}", "hi"); // 출력: "   hi"
println!("{:^5}", "hi"); // 출력: " hi "

2. 3.변수s 4번

// Fix the error with the use of define_x
fn main() {
println!("{}, world", x);
}

fn define_x() {
let x = "hello";
}

3. 4.1 넘버스 8번

fn main() {
assert!(0.1+0.2==0.3);
}

답

```
fn main() { assert!(0.1_f32+0.2_f32==0.3_f32); } // 이거는 왜 동작하는 거지?? 이게 질문임.
```

bit 수가 줄어들어서 오차가 줄어들어서 그런 것 같다고 했음

`fn main() {`
`assert!((0.1_f64+ 0.2 - 0.3).abs() < 0.001); .abs()는 절댓값`
`}` 0.001 보다 오차가 작으면 을 표현한 것

부동 소수점은 약간의 오차가 발생가능함

4. 4.1 넘버스 10번

use std::ops::{Range, RangeInclusive};
fn main() {
assert_eq!((1.._), Range{ start: 1, end: 5 });
assert_eq!((1.._), RangeInclusive::new(1, 5));
}

assert_eq!((1..5), Range{ start: 1, end: 5 });
assert_eq!((1..=5), RangeInclusive::new(1, 5));

질문쓰..

use std::ops::{Range, RangeInclusive};
fn main() {
assert_eq!((1..5), Range{ start: 1, end: 5 });
assert_eq!((1..6), RangeInclusive::new(1, 5));

```
    println!("Success!");
```

}

이것도 동작해야 하는거 아니야? 질문

Assertion 결과

따라서 두 범위는 실제로 포함된 값은 같지만, 각 범위의 **종료 조건**이 다르기 때문에 타입이 다릅니다.

- `assert_eq!((1..6), RangeInclusive::new(1, 5));`는 **타입이 서로 다르기 때문에** 실패합니다.

어서트 동작 방식 찾아보기

5. 4.4 3번

// Solve it in two ways
// DON'T let `println!` work
fn main() {
never_return();

```
    println!("Failed!");
```

}

fn never_return() -> ! {
// Implement this function, don't modify the fn signatures