세상에서 제일 쉬운 러스트 프로그래밍

4장. 조건문과 반복문

윤인도

freedomzero91@gmail.com

if/else

`if` 문은 어떤 조건을 만족하는 경우, 그에 해당하는 코드를 실행하도록 논리적 분기를 만드는 방법입니다.

```
x = 1.0
y = 10

if x < y:
    print("x is less than y")
elif x == y:
    print("x is equal to y")
else:
    print("x is not less than y")
```

실행 결과

```
x is less than y
```

```rust
fn main() {
    let x = 1.0;
    let y = 10;

    if x < (y as f64) { // casting
        println!("x is less than y");
    } else if x == (y as f64) {
        println!("x is equal to y");
    } else {
        println!("x is not less than y");
    }
}
```

실행 결과

```
x is less than y
```

let if

러스트에서는 if문의 각 분기를 변수에 바로 할당하는 것이 가능합니다.

```
fn main() {
    let x = 1.0;
    let y = 10;

    let result = if x < (y as f64) {
        "x is less than y"
    } else if x == (y as f64) {
        "x is equal to y"
    } else {
        "x is not less than y"
    };

    println!("{}", result);
}
```

> 각 분기에서 할당하는 값들이 모두 동일한 타입이어야만 사용 가능!

`if`를 함수에서 바로 리턴한다면, 다음과 같은 코드도 가능합니다. `let if` 문법이 `if` 문의 결과를 바로 얻을 수 있는 문법이기 때문입니다.

```
fn check_password(password: i32) -> bool {
    if password == 1234 {
        true
    } else {
        false
    }
}
fn main() {
    let password = 1234;
    let result = check_password(password);
    println!("Result: {}", result);
}
```

for

값들을 차례로 순회할때 사용하는 문법

```
for i in range(6, 10):
    print(i, end=",")
```

실행 결과

```
6,7,8,9,
```

- 러스트에서 특정 범위의 정수를 만들려면 `a..b` 와 같은 문법을 사용
- 마지막 값은 생략

```
fn main() {
    for i in 6..10 {
        print!("{},", i);
    }
}
```

실행 결과

```
6,7,8,9,
```

정수 범위를 변수에 할당했다가 나중에 사용하는 것이 가능

```python
num_range = range(6, 10)

for i in num_range:
    print(i, end=",")
```

```rust
fn main() {
    let num_range = 6..10;
    for i in num_range {
        print!("{},", i);
    }
}
```

러스트에서는 마지막 숫자를 포함할 수 있음

```
fn main() {
    let num_range = 6..=10;
    for i in num_range {
        print!("{},", i);
    }
}
```

while

조건이 만족되는 동안 코드가 계속 반복해서 실행

```
x = 0
while x < 5:
    print(x, end=",")
    x += 1
```

실행 결과

```
0,1,2,3,4,
```

참고로, 러스트는 파이썬과 마찬가지로 증감 연산자(`++, --`)가 없습니다.

```
fn main() {
    let mut x = 0; // 가변!
    while x < 5 {
        print!("{},", x);
        x += 1; // 증감 연산자가 없어서 x++와 같이 쓸 수 없습니다.
    }
}
```

실행 결과

```
0,1,2,3,4,
```

loop

파이썬에서 무한 루프를 구현할 때는 `while True`를 사용합니다.

```python
x = 0
while True:
    x += 1
    if x == 5:
        break
    print(x, end=",")
```

실행 결과

```
0,1,2,3,4,
```

- 러스트의 `loop`는 무한 루프를 만들 때 사용되는 문법
- `break`를 사용해 루프를 종료하고 다음으로 진행할 수 있습니다.

```
fn main() {
    let mut x = 0;
    loop {
        x += 1;
        if x == 5 {
            break;
        }
        print!("{},", x);
    }
}
```

실행 결과

```
0,1,2,3,4,
```

`loop`에서 특정 값을 리턴할 수 있습니다. `break` 뒤에 리턴할 값을 넣어주면 됩니다. `x`가 5가 됐을 때 `x`를 리턴하도록 코드를 고치면 다음과 같습니다.

```
fn main() {
    let mut x = 0;
    let y = loop {
        x += 1;
        if x == 5 {
            break x;
        }
        print!("{},", x);
    };

    println!("{}", y);
}
```

루프 안에서 1부터 4까지가 출력되고, 그 뒤에 `y`의 값 5가 출력됩니다.

`break`와 `continue`

파이썬과 동일하게 작동함

```
fn main() {
    for i in 0..10 {
        if i % 2 == 0 {
            continue; // 짝수일 때는 스킵
        } else if i == 7 {
            break; // 7일 때는 루프 종료
        }
        print!("{}, ", i);
    }
}
```

실행 결과

```
1, 3, 5,
```

match

다음은 다른 언어에서는 `switch ... case` 로 많이 사용되는 `match` 입니다.

아래 코드는 `name` 변수에 값에 따라서 서로 다른 결과를 출력하는 코드입니다. 현재 `name` 변수의 값이 `"John"` 이므로 `"Hello, John!"`가 출력됩니다.

```
name = "John"
if name == "John":
    print("Hello, John!")
elif name == "Mary":
    print("Hello, Mary!")
else:
    print("Hello, stranger!")
```

실행 결과

```
Hello, John!
```

`match`에서 나머지 경우를 나타내기 위해서 매칭할 값을 생략하는 `_`을 사용합니다. 여기서 `name` 변수의 값이 `"John"`이기 때문에 `"Hello, John!"`이 출력됩니다.

```
fn main() {
    let name = "John";
    match name {
        "John" => println!("Hello, John!"),
        "Mary" => println!("Hello, Mary!"),
        _ => println!("Hello, stranger!"),
    }
}
```

실행 결과

```
Hello, John!
```

`loop`와 마찬가지로 `match` 문도 값을 리턴할 수 있습니다. `let <변수명> = match ...`와 같이 선언하면 됩니다. 이때 컴파일러가 `match` 문의 리턴값으로부터 변수 `greet`의 타입을 추론합니다. 또한, 각 조건마다 리턴하는 값들의 타입이 반드시 동일해야 합니다.

```
fn main() {
    let name = "John";
    let greet = match name {
        "John" => "Hello, John!",
        "Mary" => "Hello, Mary!",
        _ => "Hello, stranger!",
    };

    println!("{}", greet);
}
```

파이썬에서는 최신 버전인 3.10 이후부터 `match ... case`가 추가되었습니다.

```
name = "John"

match name:
    case "John":
        print("Hello, John!")
    case "Mary":
        print("Hello, Mary!")
    case _:
        print("Hello, stranger!")
```

연습문제

1. 정수를 인자로 받아 정수가 양수이면 "positive"를, 그렇지 않으면 "negative"를 프린트하는 함수 `check_sign`을 작성합니다.

```
fn check_sign(n: i32) {
    ...
}

fn main() {
    check_sign(3);
}
```

정답

```rust
fn check_sign(n: i32) {
    if n > 0 {
        println!("positive")
    } else {
        println!("negative")
    }
}

fn main() {
    check_sign(3);
}
```

2. 다음과 같은 피라미드 패턴을 만드는 코드를 작성해보세요.

```
*
**
***
```

정답

```rust
fn main() {
    for i in 1..=3 {
        for _ in 1..=i {
            print!("*");
        }
        println!();
    }
}
```