파이썬 프로그래머를 위한 러스트 입문

윤인도

freedomzero91@gmail.com

CH3. 함수

이번 챕터에서는 함수의 선언과 사용에 대해서 자세히 다루겠습니다.

함수 선언

파이썬

```
def add(num1: int, num2: int) -> int:
    return num1 + num2
```

러스트

함수의 선언에 `fn` 키워드를 사용하고, 함수에서 실행할 코드를 중괄호로 묶어줍니다. 그리고 파이썬과 비슷하게 파라미터에는 `:i32`로 타입을 표기하고, 리턴값에는 `-> i32`처럼 화살표를 사용해 타입을 명시했습니다.

> 이때 주의해야 하는 점은 반드시 파라미터와 리턴 타입을 명시해야 한다는 것입니다.

```
fn add(num1: i32, num2: i32) -> i32 {
    return num1 + num2;
}
```

러스트는 코드 마지막에서 `return` 키워드를 생략할 수 있습니다. 이때 세미콜론이 없다는 점에 주의하세요.

```
fn add(num1: i32, num2: i32) -> i32 {
    num1 + num2
}
```

이제 `add` 함수를 메인 함수에서 호출하고 값을 프린트해 보겠습니다.

```
fn add(num1: i32, num2: i32) -> i32 {
    num1 + num2
}

fn main() {
    println!("{}", add(1, 2));
}
```

실행 결과

```
3
```

파이썬에서 `swap` 이라는 함수를 아래와 같이 구현합니다.

```python
def swap(num1: int, num2: int) -> tuple[int, int]:
    return num2, num1


num1, num2 = swap(1, 2)
print(f"{num1}, {num2}")
```

실행 결과

```
2, 1
```

러스트도 여러 개의 값을 리턴하는 경우, 값들이 튜플로 묶이게 됩니다.

따라서 함수의 리턴 타입도 튜플로 `(i32, i32)` 표기합니다.

```
fn swap(num1: i32, num2: i32) -> (i32, i32) {
    (num2, num1)
}

fn main() {
    let (num1, num2) = swap(1, 2);
    println!("{num1}, {num2}");
}
```

실행 결과

```
2, 1
```

스코프

스코프(scope)란 변수에 접근할 수 있는 범위를 의미합니다. 먼저 파이썬에서는 스코프를 기본적으로 함수 단위로 구분합니다.

> 실제로는 파이썬은 LEGB 룰이라고 불리는 좀더 복잡한 스코프 규칙을 가지고 있지만, 여기서는 단순화해서 함수 기준으로 설명합니다.

```python
def hello(name: str):
    num = 3
    print(f"Hello {name}")


if __name__ == '__main__':
    my_name = "buzzi"

    if True:
        print("My name is", my_name)
        my_name = "mellon"

    hello(my_name)

    # print(num) # error
```

실행 결과

```
My name is buzzi
Hello mellon
```

```
fn hello(name: String) {
    let num = 3;
    println!("Hello {}", name);
}

fn main() {
    let my_name = "buzzi".to_string();

    {
        println!("My name is {}", my_name);
        let my_name = "mellon";
    }

    hello(my_name);

    // println!("{}", num); // error
}
```

실행 결과

```
My name is buzzi
Hello buzzi
```

익명 함수

익명 함수란 이름이 없는 함수라는 뜻으로, 프로그램 내에서 변수에 할당하거나 다른 함수에 파라미터로 전달되는 함수입니다. 따라서 익명 함수를 먼저 만들어 놓고 나중에 함수를 실행할 수 있습니다.

파이썬에서는 `lambda` 키워드를 사용합니다.

```
my_func = lambda x: x + 1
print(my_func(3))
```

러스트에도 람다 함수와 비슷한 개념이 있는데 바로 클로저(Closure)입니다.
클로저는 파라미터를 `| |` 의 사이에 선언하고, 그 뒤에 함수에서 리턴하는 부분을 작성합니다.

```
fn main() {
    let my_func = |x| x + 1;
    println!("{}", my_func(3));
}
```

이때 컴파일러가 클로저의 파라미터와 리턴값의 타입을 `i32` 로 추측해서 보여줍니다.
하지만 타입을 명시하는 것도 가능합니다.

```
fn main() {
    let my_func = |x: i32| -> i32 { x + 1 };
    println!("{}", my_func(3));
}
```

Quiz

1. 두 개의 정수를 인자로 받아 두 정수의 곱을 반환하는 클로저를 작성해 보세요.

```
fn main() {
    let multiply_numbers = |?| -> ? {  };

    let result = multiply_numbers(3, 4);
    println!("The product of 3 and 4 is: {}", result); // 12
}
```

정답1

```rust
fn main() {
    let multiply_numbers = |a: i32, b: i32| -> i32 { a * b };

    let result = multiply_numbers(3, 4);
    println!("The product of 3 and 4 is: {}", result);
}
```

2. 두 개의 정수를 인자로 받아 두 정수 중 더 큰 값을 반환하는 함수 `find_max` 를 작성합니다.

```
fn find_max(?) ? {
}

fn main() {
    let result = find_max(3, 4);
    println!("The larger number is: {}", result); // 4
}
```

정답2

```rust
fn find_max(a: i32, b: i32) -> i32 {
    if a > b {
        a
    } else {
        b
    }
}

fn main() {
    let result = find_max(3, 4);
    println!("The larger number is: {}", result);
}
```