Functional Programming

Introduction

Tae Geun Kim

- Table of Contents

- 뭐
- 왜
- 그래서 뭐

뭐

함수형 프로그래밍

- 객체지향 프로그래밍이 객체 위주라면 함수형 프로그래밍은 당연하게도 함수 위주
- 함수는 일급 시민 - 반환 가능하고 인수로 사용 가능함
- 예를 들면

```haskell
-- Haskell
hello :: String -> (String -> String)
-- String을 받아서 String에서 String으로 가는 함수를 반환함
hello text = (text++)

test = hello "Hi"
test " Axect"
```

- Haskell은 이해가 안되는데..
- 그래서 준비했습니다!

```julia
# Julia
function hello(text::String)
  return function hi(text2::String)
  return text * text2
  end
end

hi = hello("Hi ")
# hi (generic function with 1 method)

hi("Axect")
# "Hi Axect"
```

니가 알던 내가 아니야

- 함수형 프로그래밍의 함수는 기존 프로그래밍의 함수들과 다름
- 함수형 프로그래밍의 함수는 순수하다. (= 수학에서의 함수와 동치이다.)
- 함수의 순수함은 참조투명성 (referentially transparent) 으로 정의됨
- 쉽게 말해서 다음과 같은 함수는 수학에서 쓰지만

$$f(x) = x^2$$

- 이런 함수는 안 쓰지 않나

$$f(x) = print(\text{“}helloworld\text{”})\ and\ x^2$$

참조 투명성

- 참조투명성은 간단하게 "치환 가능성"을 의미
- 예를 들어 다음 C++ 코드를 보자.

```
#include <iostream>

using namespace std;

int Inner(int a, int b) {
    return a * b;
}

int main() {
    cout <<Inner(3, 4) + Inner(1,2)<< endl;

    return 0;
}
```

- `Inner(3,4)`는 12, `Inner(1,2)`는 2로 치환해도 결과는 같다. 따라서 `Inner`는 참조투명한 함수이며 순수한 함수이다.

순수하지 않은 함수?

- 치환 불가능한 함수도 있나? - 객체지향의 기본함수들은 대개 치환 불가능

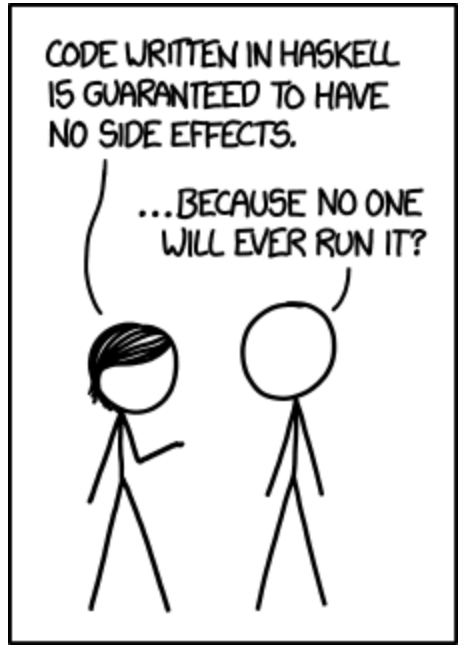
```julia
# Julia
struct Circle
    radius :: Float64
end

function extend(C::Circle)
    C.radius += 1
end
```

- 값의 변경이 일어나는 경우나 print, 파일 읽기 쓰기 등등의 현실과 맞닿은 기능들은 순수하지 않음 -> 부작용 **(Side Effect)!**
- 현재는 부작용을 없애거나 거의 없앤 프로그래밍 패러다임을 함수형 프로그래밍이라고 말함.

"거의 없앤**?**"

- 말했듯이 순수함수로는 `print` 하는 것도 어려우니 순수함수만 외치면 사람들이 외면 (다음 짤을 참조)



- 따라서 부작용의 도입에 따라 함수형 언어들이 분류 된다.
  i. **완전 불허용**: Haskell - Monad라는 대수적 구조로 요리조리 피해감, Elm, Idris, Coq (4색정리!)
  ii. **대체적으로 불허용**: Erlang, Elixir, F#, Clojure, Ocaml
  iii. **완전 허용**: Scala - FP + OOP (~~아단이다!~~)