



Haskell01

함수형 패러다임, 그리고 그 이상

차례

- 시작하기 앞서서
 - 왜 함수형이냐?
 - 왜 Haskell을 해?
- 준비과정
 - 컴파일은?
 - 편집기는?
 - 잠깐! 뭘 어떻게 할건데?

차례

- Haskell Basic
 - Haskell에 묶이기 - Binding
 - 함수형이면 함수를 써야지 - Function
 - 니 뭐 그래 특이하나? - Currying & Purity
 - Haskell모양으로 잘라줄게요 - Type Basic
 - 이런 모양은 어떨까? - Complex Type Basic



시작하기 앞서서

대체 왜 하는데?

왜 함수형이냐?

- 의문들
 - “함수형은 이해하기 어렵지 않아요?”
 - “함수형은 주류 언어도 아니잖아요?”
 - “실제로 쓰긴 하나요?”

왜 함수형이냐?

- “함수형은 이해하기 어렵지 않나요?”
- (이과) 사람들은 수학과 프로그래밍 중 어떤 것을 먼저 배울까?
- (이과) 사람들은 수학과 프로그래밍 중 어떤 것을 더 많이 했을까?

왜 함수형이냐?

- 함수형 = 수학적인 프로그래밍 언어
- 수학에서 생각할 수 있는 많은 것들이 그대로 적용된다.
= 많은 (이과) 사람들의 생각 방식을 따를 수 있다.

왜 함수형이냐?

- “함수형은 주류 언어도 아니잖아요?”
- 주류언어는 무엇일까?
 - C, C++, Python, Java, ...

왜 함수형이냐?

- C++98~
 - <algorithm> header의 확장, lambda, fold, concepts 도입
- Java 8
 - Lambda 식 도입
- Python
 - Lambda 식 지원, List Comprehension

왜 함수형이냐?

- 주류 언어들? 함수형의 발자취를 좇는 언어들!

왜 함수형이냐?

- “실제로 쓰긴 쓰나요?”
- Bluespec
 - Haskell 기반의 하드웨어 디자인 언어 (고수준 언어이기 때문에 빠른 디자인에 좋음)
- Leksah
 - Haskell로 만들어진 Haskell IDE

왜 함수형이냐?

- “실제로 쓰긴 쓰나요?”
- Coq
 - OCaml로 쓰여진 증명 보조도구 (4색 문제 증명에 사용됨)
- Pandoc
 - 문서변환 프로그램. 수많은 프로그램이 의존한다. 리눅스를 쓰다 보면 의존성 목록에서 자주 보게 되는 프로그램.

왜 함수형이냐?

- “실제로 쓰긴 쓰나요?”
- Haxl
 - Haskell로 쓰여진 데이터 액세스 도구 (Facebook에서도 사용 중)

왜 함수형이냐?

- “함수형은 이해하기 어렵지 않아요?” => No!
- “함수형은 주류 언어도 아니잖아요?” => 주류언어들이 함수형을 좇아온다!
- “실제로 쓰긴 하나요?” => 물론 Yes!

왜 하스켈을 해?


- 성능
 - 고수준 언어이면서 Java 수준에서 C++ 수준의 성능을 낼 수 있다.
- 순수 함수
 - 다른 함수형 언어들에서도 보기 드물 정도로 순수 함수를 잘 지원한다.

왜 하스켈을 해?

- 짧은 Code
 - Prelude(기본 지원 기능)에 있는 수많은 함수들을 사용하여 간략한 코드를 만들 수 있다.
- 느긋함
 - 필요할 때까지 최대한 늦장 부린다.

왜 하스켈을 해?

- 순수 함수?? 느긋함??
 - Haskell을 배우면서 다루자!



준비과정

뭐 깔거 많냐?

컴파일은?

- 프로그램을 돌리려면 컴파일을 할 수 있어야 한다
- 게다가 **Haskell**은 인터프리팅도 지원하는 언어이다.
- 인터프리터던 컴파일러던 필요하다.

컴파일은?

- 가장 대표적인 컴파일러 GHC(Glasgow Haskell Compiler)
 - GHC <https://www.haskell.org/ghc/>
 - Haskell Platform with GHC (권장) <https://www.haskell.org/platform/>

편집기는?

- 소스코드를 쓰려면 편집기가 있어야 한다.

편집기는?

- 자기가 익숙한 편집기
 - atom, sublime, emacs, vi, ...
- Haskell IDE
 - Leksah
- General IDE
 - IntelliJ IDEA, Eclipse IDE, ...
- 설정법 := <https://wiki.haskell.org/IDEs>

잠깐! 뭘 어떻게 할건데?

- **GHCI**를 사용해 **소스코드**를 치고 바로 확인해보는 것으로 시작한다
 - Window의 경우 **Haskell Platform**을 깔았다면 **WinGHCI**를 사용하는 것이 좋다.
- 기본적인 문법에 익숙해지면 **소스 파일**을 작성하고 **GHCI**에 로드해서 테스트 해 본다
- 마지막으로, **소스 파일**을 작성하고 **GHC**로 컴파일하여 프로그램을 실행시켜 본다.



Haskell Basic

대체 어떻게 생겨먹은 녀석인데?

Haskell에 묶이기 - Binding

- 수학에서 말하는 정의하기
- `let val = 5`
 - `val`는 5이다.
- `let fun x = 2 * x`
 - `fun(x)` 는 $2x$ 이다.

Haskell에 묶이기 - Binding

- 정의를 확인해보자
- `val`
- `fun 2`
- `fun val`

Haskell에 묶이기 - Binding

- 만일 똑같은 이름을 바꾸려 한다면?
- `let val = 3`
- 갱신X 새로 정의하기O

Haskell에 묶이기 - Binding

- 갱신은 기본적으로 불가능하다!

함수형이면 함수를 써야지 - Function Basic

- 앞에서 이미 함수를 하나 살펴보았다.
- `let fun x = 2 * x`

함수형이면 함수를 써야지 - Function Basic

- 조금 더 복잡한 함수를 보자.
- `let funComp x y = x^2 + y^2`

함수형이면 함수를 써야지 - Function Basic

- 사용은 어떻게 할까?
- funComp 3 4

함수형이면 함수를 써야지 - Function Basic

- 조금 더 복잡하게 써보자
- `funComp 3 2 + 2`
- `funComp 3 (2 + 2)`
- 차이는 무엇일까?

함수형이면 함수를 써야지 - Function Basic

- 온갖 ‘연산자’들....
 - $+$, $-$, $*$, $^$, ...
- 사실은 모두 함수다!
- $(+)$ 3 4
- $(*)$ 2 1
- $(^)$ 6 3

함수형이면 함수를 써야지 - Function Basic

- 연습!
- 직육면체의 높이, 너비, 깊이를 받아 직육면체의 겉넓이를 구하는 함수를 짜보자

니 뭐 그래 특이하나? - Currying & Purity

- Currying & Purity
 - Haskell의 특징들

니 뭐 그래 특이하냐? - Currying & Purity

- Currying

- 다변수 함수를 일변수 함수들 여러 개로 나타낼 수 있다

???

니 뭐 그래 특이하냐? - Currying & Purity

- $f(x, y, z) = xyz$
- $a(x) = b_x(y) = c_{x,y}(z) = xyz$
 - $a(5) = 5yz$
 - $b_5(3) = 5 * 3z$
 - $c_{5,3}(4) = 5 * 3 * 4$

니 뭐 그래 특이하나? - Currying & Purity

- $f(x, y, z) = xyz$
- 삼변수 함수

니 뭐 그래 특이하나? - Currying & Purity

- $a(x)$
- $b_x(y)$
- $c_{x,y}(z)$
- 모두 일변수 함수

니 뭐 그래 특이하냐? - Currying & Purity

- 실제 Haskell 코드
- `let funCurryA x y z = x + y + z`
- `let funCurryB = funCurryA 3`
- `let funCurryC = funCurryB 2`
- `let result = funCurryC 1`
- `result`

니 뭐 그래 특이하나? - Currying & Purity

- Purity
 - 부작용(Side-effect)이 없다
- Side-effect
 - 매개변수와 결과값 외의 다른 부분에 영향을 미치는 것

니 뭐 그래 특이하나? - Currying & Purity

- C, C++, ...
 - 동일한 함수를 똑같은 매개변수를 넘겨서 두 번 불렀다고 하자. 결과값이 같을까?
 - `rand();`
- Haskell
 - 동일한 함수를 똑같은 매개변수를 넘겨서 두 번 불렀다고 하자. 결과값은 항상 같다!

니 뭐 그래 특이하냐? - Currying & Purity

- 이전에 부른 함수는 똑같은 결과를 돌려준다? → **Caching**이 가능하다!
- 그러면 입출력을 못하지 않냐???? 함수가 아니면 되지!

Haskell모양으로 잘라줄게요 - Type Basic

- 지금까지 써온 코드에서 타입은?? 없었다!
- 그렇다면
- `fun 'd'`
- `funComp 'm' 'y'`
- 도 동작할까?

Haskell모양으로 잘라줄게요 - Type Basic

- 타입 추정
 - 타입을 쓰지 않아도 알아서 가장 일반적인 타입을 찾아준다!
- 강력한 타입
 - 타입을 멋대로 바꾸지 않는다.
 - 약한 타입 언어에서는 문자가 숫자로, 숫자가 문자로 멋대로 바뀔 수 있다.

Haskell모양으로 잘라줄게요 - Type Basic

- 타입을 확인하는 방법
- `:t 'b'`
- `:t fun`
- `:t (+)`
- `::` 뒤에 있는 것들이 타입이다.

Haskell모양으로 잘라줄게요 - Type Basic

- a
 - 타입에 등장하면서 대문자로 시작하지 않는 모든 것 = 타입 변수
- Num a
 - a라는 타입 변수가 Num 이라는 클래스에 포함된다.
 - 클래스?? 뒤!에!서!
 - 지금은 +, -, * 같은 것들이 가능한 모든 타입 a들을 Num a라고 부른다고 하자.

Haskell모양으로 잘라줄게요 - Type Basic

- 'b'의 타입 :: Char
 - Char 타입
- fun의 타입 :: Num a => a -> a
 - Num에 속하는 a 타입의 값을 받아서 a타입의 결과값을 돌려주는 함수 타입

Haskell모양으로 잘라줄게요 - Type Basic

- **(+)**의 타입 :: Num a => a -> a -> a
 - -> 는 오른쪽부터 괄호를 묶어서 보면 된다
 - a -> (a -> a)
 - a 타입의 값을 받아서 (a -> a) 타입의 결과값을 돌려주는 함수 타입
 - (a -> a) 타입???? a 타입의 값을 받아서 a 타입의 결과값을 돌려주는 함수 타입!
 - ???
 - (+) 는 a 타입 값 2 개를 받아서 a 타입의 결과값을 돌려주는 함수 타입

Haskell모양으로 잘라줄게요 - Type Basic

- Currying으로 돌아가보자
- $a(x) = b_x(y) = c_{x,y}(z) = xyz$
- $a(x)$ 의 결과값은? $b_x(y)$ 라는 함수!
- Currying := ‘모든 함수들을 함수를 결과값으로 가지는 일변수 함수 여러 개로 쪼개자!’

Haskell모양으로 잘라줄게요 - Type Basic

- Quiz!
- `let quizFun1_1 = (+)`의 타입은?
 - `Num a => a -> a -> a`
- `let quizFun1_2 = (+) 5`의 타입은?
 - `Num a => a -> a`
- `let quizFun1_3 x = 'i'`의 타입은?
 - `t -> Char`

이런 모양은 어떨까? - Complex Type Basic

- 프로그래밍을 할 때에는 조금 더 복잡한 값들을 다루어야 할 필요가 있다.
 - C/C++/Java/...의 배열, 구조체, 클래스, ...
 - Ex) 날짜별 강우량을 저장해놓고 싶을 때

이런 모양은 어떨까? - Complex Type Basic

- Haskell에선...
 - Tuple
 - List
 - User Define Type

이런 모양은 어떨까? - Complex Type Basic

- Tuple :: (a, b) 또는 (a,b,c) 또는 ...
 - 여러 타입의 값들을 묶어서 다루고 싶을 때
- ('a', 5)
- :t ('a', 5)
- (57, 'a', 22)
- :t (57, 'a', 22)

이런 모양은 어떨까? - Complex Type Basic

- **Tuple**은 길이에 따라 서로 다른 타입이 된다.
- 2개짜리 **Tuple**을 특별히 **Pair**라고 부른다.
- 그 이상의 **Tuple**은 **n-ple**로 부른다. (triple, quadruple, pentuple, ...)

이런 모양은 어떨까? - Complex Type Basic

- Pair를 위한 함수들
 - **fst**
 - Pair의 첫번째 값을 가져온다.
 - **snd**
 - Pair의 두번째 값을 가져온다.
 - **swap**
 - Pair의 첫번째 값과 두번째 값을 바꾼다.

이런 모양은 어떨까? - Complex Type Basic

- List :: [t]
 - 같은 타입의 값 여러 개를 다루고 싶을 때
- [1, 2]
- [1, 3, 4]
- :t [1, 5, 6, 7, 8]
- [1..5]

이런 모양은 어떨까? - Complex Type Basic

- List :: [t]
 - 다른 표현법도 있다! 이게 원래 정의이다.
- 1:(2:[])
- 1:(3:(4:[]))
- :t 1:5:6:7:8:[]
- enumFromTo 1 5

이런 모양은 어떨까? - Complex Type Basic

- 왜 이렇게 정의하지?

이런 모양은 어떨까? - Complex Type Basic

- 가장 간단하게 정의할 수 있어서!
 - `[]`
 - 빈 리스트 :: `[t]`
 - `(:)`
 - 원소를 리스트 앞에다 추가하는 함수 :: `a -> [a] -> [a]`

이런 모양은 어떨까? - Complex Type Basic

- List를 위한 수많은 함수들
 - head, last, tail, init
 - length
 - map, reverse
 - foldl, foldr
 - any, all, ...

이런 모양은 어떨까? - Complex Type Basic

- 나중에 보게 될 겁니다.

이런 모양은 어떨까? - Complex Type Basic

- Tuple VS List
 - 타입
 - Tuple은 원소의 개수마다 다른 타입! :: (a, b), (a, b, c), ...
 - List는 원소의 개수에 상관없이 같은 타입! :: [t]
 - 길이 바꾸기
 - Tuple은 원소를 연달아 추가하는 것을 하나의 함수로 할 수 없다! 타입이 다르기 때문
 - List는 (:)를 사용해 원소를 연달아 추가할 수 있다!

이런 모양은 어떨까? - Complex Type Basic

- Quiz!
- `let l = 'c':l` 에서 `l` 의 타입은?
 - [Char]