# Haskell02

함수형 패러다임, 그 이상

# 차례

- 되새김질
  - 내가 방금 뭘 했더라? - Review
- Haskell Elementary
  - 진짜 함수 만들기 - Pattern Matching And Others
  - 계산되거나 말거나 - Some Expressions And Statement
  - 한 번만 다시 생각해봐요... - Functional Thinking

# 차례

- Haskell Elementary
  - List에 당도한 것을 환영하네 낯선 이여 - List Type
  - 차 마시면서 즐기는 느긋함 - Lazy Evaluation
  - Nothing or not - Maybe Type
  - 일급 객체랑 놀기 - Function As First Class Object
  - 바보야 니 이름은 개발자가 버렸어 - Lambda Function
  - 따르릉 따르르릉 - Simple Example

# 차례

- Practice

# 되새김질

내가 소냐

# 내가 방금 뭘 했더라?

- Binding
- Function Definition
- Currying
- Purity
- Type
- Tuple
- List

# Haskell Elementary

우리 연변에서는 이 정도는 해야 아 고거이 Haskell 좀 시작했다 그럼다

# 진짜 함수 만들기
# Pattern Matching And Others

- 지금까지는 매우 간단한 함수들을 만들어 보았다.
- 조금 더 복잡한 함수를 만들어보자.

# 진짜 함수 만들기
# Pattern Matching And Others

- 지금까지는 매우 간단한 함수들을 만들어 보았다.
- 조금 더 복잡한 함수를 만들어보자
- 복잡한 함수를 위해 일단 원하는 편집기를 켜자.

# 진짜 함수 만들기
# Pattern Matching And Others

- Factorial을 계산하는 다음과 같은 타입의 함수를 만들어보자
- factorial :: Num a => a -> a

# 진짜 함수 만들기
# Pattern Matching And Others

- Factorial을 정의해보자.
- $0! = 1$
- $n! = n * (n-1)!$

# 진짜 함수 만들기
# Pattern Matching And Others

- factorial을 정의해보자.
- 적당한 경로에 First.hs파일을 만들고 아래와 같이 친다.
- ```
  factorial 0 = 1
  factorial n = n * factorial (n-1)
  ```
  ```

# 진짜 함수 만들기
# Pattern Matching And Others

- let은 어디로??
- 똑같은 함수가 두 개??
- 함수의 매개변수가 0??

# 진짜 함수 만들기
# Pattern Matching And Others

- let은 어디로??
  - 소스 파일을 쓸 때에는 Binding에는 쓸 필요가 없다.
  - 어디에 쓰는지는 조금 뒤에

# 진짜 함수 만들기
# Pattern Matching And Others

- 똑같은 함수가 두 개??
- 함수의 매개변수가 0??
  - 패턴 매칭 Pattern Matching!
  - 입력 값을 정해진 Pattern과 각각 맞춰본다

# 진짜 함수 만들기
# Pattern Matching And Others

- 이제 소스 파일을 GHCI에서 불러서 확인해보자.
- :load <파일경로>
- ex) :show paths
  :cd haskell_example
  :load First.hs
- WinGHCI의 경우 File 메뉴의 Load를 사용하면 된다.

# 진짜 함수 만들기
# Pattern Matching And Others

- factorial 0
- 은 factorial 0 = 1에 매칭되고,
- factorial 2
- 은 factorial n = n * factorial (n-1)에 매칭된다.

# 진짜 함수 만들기
# Pattern Matching And Others

- factorial 2가 계산되는 순서를 좀 더 자세히 보면,
- 은 factorial 2 = 2 * factorial (2-1)가 되고,
- factorial (2-1)는 factorial 1이므로,
- factorial 1 = 1 * factorial (1-1),
- factorial (1-1)는 factorial 0 = 1이 된다.
- 따라서 factorial 2 = 2 * 1 * 1 = 2

# 진짜 함수 만들기
# Pattern Matching And Others

- factorial에 음수가 들어가면 어떻게 될까?

# 진짜 함수 만들기
# Pattern Matching And Others

- factorial (-2)가 계산되는 순서를 좀 더 자세히 보면,
- 은 factorial (-2) = (-2) * factorial ((-2)-1)가 되고,
- factorial ((-2)-1)는 factorial (-3)이므로,
- factorial (-3) = (-3) * factorial ((-3)-1),
- factorial ((-3)-1)는 factorial (-4),
- …

# 진짜 함수 만들기
# Pattern Matching And Others

- 매개변수가 음수일 때는 0을 결과값으로 돌려줄 수는 없을까?

# 진짜 함수 만들기
# Pattern Matching And Others

- 다시 Factorial을 정의해보자
- $$f(n) = \begin{cases} 0, & n < 0 \\ 1, & n = 0 \\ n * f(n-1), & n > 0 \end{cases}$$

# 진짜 함수 만들기
# Pattern Matching And Others

- 다시 Factorial을 정의해보자
- 
```
factorial n | n < 0 = 0
            | n == 0 = 1
            | otherwise = n * factorial (n-1)
```

# 진짜 함수 만들기
# Pattern Matching And Others

- | b = a???
  - Guard
  - 현재의 Pattern에 조건을 추가한다
  - b이 성공했을 때만 a를 결과값으로 준다.

# 진짜 함수 만들기
# Pattern Matching And Others

- | otherwise = e???
  - 이전의 모든 Guard들이 실패했을 때 e를 결과값으로 돌려준다.
  - | True = e라고 쓴 것과 마찬가지이다.

# 진짜 함수 만들기
# Pattern Matching And Others

- 다른 언어에서 익숙할 방식으로 써보자
- factorial n =
  if n < 0 then 0
  else if n == 0 then 1
  else n * factorial (n-1)

# 진짜 함수 만들기
# Pattern Matching And Others

- 참고/함수 정의 내에서 Pattern Matching하기
- ```
  factorial n = case (compare n 0) of
    LT -> 0
    EQ -> 1
    GT -> n * factorial (n-1)
  ```
  ```

# 진짜 함수 만들기
# Pattern Matching And Others

- 의문
  - “Pattern Matching이 If-Else보다 나은게 뭐에요?”

# 진짜 함수 만들기
# Pattern Matching And Others

- 리스트와 함께하는 Pattern Matching
- sumList [] = 0
  sumList (x:xs) = x + sumList xs

# 진짜 함수 만들기
# Pattern Matching And Others

- Pattern Matching없이 짜면?
- ```
  sumList l =
      if length l == 0 then 0
      else
         let h = head l
             t = tail l in
          h + sumList t
  ```
  ```

# 진짜 함수 만들기
# Pattern Matching And Others

- C비슷하게 짜면?
- 길이확인하고
- 리스트의 첫번째 원소를 나누고
- 리스트의 첫번째를 제외한 리스트를 나누고
- 재귀하고

# 진짜 함수 만들기
# Pattern Matching And Others

- Pattern을 사용하면 훨씬 간단하고 이해하기 쉽게 짤 수 있다!

# 진짜 함수 만들기
# Pattern Matching And Others

- 일부 패턴의 이름이 필요없는 경우
- ```
  dropThree (_:_:_:xs) = xs
  dropThree _ = []
  ```
  ```

# 진짜 함수 만들기
# Pattern Matching And Others

- Pattern을 사용하면 필요한 이름들만 부분적으로 정의할 수 있다

# 계산되거나 말거나
# Some Expressions And Statements

- 다음과 같은 함수를 생각해보자.
- bmiChecker h w | w / h^2 <= 18.5 = “Bone”
  | w / h^2 <= 26.5 = “Human. Whatever”
  | w / h^2 <= 34.5 = “Fat ball”
  | otherwise = “Horrible”

# 계산되거나 말거나
# Some Expressions And Statements

- 다음과 같은 함수를 생각해보자.
- w / h^2가 계속 반복된다. 어떻게 하면 좋을까?

# 계산되거나 말거나
# Some Expressions And Statements

- 잠깐 사용하는 것들에 이름을 부여하는 방법
- let expression
- where statement

# 계산되거나 말거나
# Some Expressions And Statements

- let expression
  - 우리가 지금껏 ghci 위에서 써왔던 let의 정체
  - expression, 즉 결과값이 있는 식
  - let ... in 사이에서 정의된 것들은 in 이후에서 잠깐만 사용할 수 있다.

# 계산되거나 말거나
# Some Expressions And Statements

- where statement
  - guard에서도 쓸 수 있는 임시 정의
  - 결과값을 가지지 않기 때문에 식 중간에 끼워넣을 수는 없다
  - where ... 에 정의된 것들은 함수 안에서 그 앞의 모든 부분에 쓸 수 있다.

# 계산되거나 말거나
# Some Expressions And Statements

- 앞의 예제를 where statement를 사용해서 고쳐보자
- bmiChecker h w | bmi <= 18.5 = “Bone”
  | bmi <= 26.5 = “Human. Whatever”
  | bmi <= 34.5 = “Fat ball”
  | otherwise = “Horrible”
  where bmi = w / h^2

# 계산되거나 말거나
# Some Expressions And Statements

- 동일한 식을 반복하지 않을 수 있다.
- 만일 bmi에 대한 계산식이 바뀐다면 식 하나만 바꾸어 쓰면 된다.

# 한 번만 다시 생각해봐요…
# Functional Thinking

- 함수형적으로 생각한다는 건 어떤 것일까?
- 1. 우리가 해결해야할 문제를 생각한다.
- 2. 해결해야할 문제를 비슷하지만 더 작은 문제로 쪼개거나 풀기 쉬운 다른 문제들로 쪼갠다.
- 3. 적절한 타입을 정한다.
- 4. 소스코드로 변환한다.

# 한 번만 다시 생각해봐요…
# Functional Thinking

- 학생 명단에 “Jesus”가 있는지 확인하기

# 한 번만 다시 생각해봐요…
# Functional Thinking

- 1. 이름들의 List에서 “Jesus”라는 값이 있는지 찾기
  => List에 a라는 값이 있는지 찾기

# 한 번만 다시 생각해봐요…
# Functional Thinking

- 2. List의 머리가 a인지 확인하기
  List의 뒷 부분이 a를 포함하는지 확인하기
- List의 뒷 부분도 List이다!!

# 한 번만 다시 생각해봐요…
# Functional Thinking

- 3. List와 List의 원소 하나를 받아 비교한 뒤, 참 거짓을 결과값으로 가지므로

  (Eq a) => [a] -> a -> Bool

- (Eq a)는 a가 비교할 수 있다는 뜻이다.

# 한 번만 다시 생각해봐요…
# Functional Thinking

- 4.

```
inList [] a = False
inList (x:xs) a | a == x = True
                | otherwise = inList xs a
```

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- List의 함수들을 제대로 살펴보도록 하자.

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- List의 간단한 함수들은 다음과 같다.

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- head
  - List의 첫번째 원소를 돌려주는 함수. []의 경우에는 에러.
- tail
  - List의 첫번째 원소를 제외한 List를 돌려주는 함수. []의 경우에는 에러.
- init
  - List의 마지막 원소를 제외한 List를 돌려주는 함수. []의 경우에는 에러.
- last
  - List의 마지막 원소를 돌려주는 함수. []의 경우에는 에러.

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- length
  - 리스트의 길이를 알려주는 함수.
- (++)
  - 두 리스트를 하나로 합치는 함수.
- (!!)
  - 리스트의 n번째 원소를 찾는 함수
- reverse
  - 리스트를 뒤집는 함수.

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- myHead, myTail, myInit, myLast, myLength, myApp, myAt, myReverse를 구현해보자

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- List의 약간 복잡한 함수들은 다음과 같다.

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- map
  - 리스트의 처음부터 끝까지 똑같은 함수를 적용하기
  - map (*2) [1,2,3,4]

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- filter
  - 리스트에 조건을 검사해서 일부만 남기기
  - filter (>2) [1,2,3,4]

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- List의 좀 많이 복잡한 함수들은 다음과 같다.

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- foldl
  - 리스트를 왼쪽부터 하나의 값으로 접는 함수
  - foldl (-) 0 [1,2,3,4]

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- foldr
  - 리스트를 오른쪽부터 하나의 값으로 접는 함수
  - foldr (-) 0 [1,2,3,4]

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- 이외에도 다양한 함수들이 있다.
- and, or, any, all, concat, concatMap, scanl, scanl1, scanr, scanr1, iterate, repeat, replicate, cycle, take, drop, span, break, lookup, zip, zipWith, …
- https://hackage.haskell.org/package/base-4.8.2.0/docs/Prelude.html#g:13
- http://hackage.haskell.org/package/base-4.8.2.0/docs/Data-List.html

# List에 당도한 것을 환영하네 낯선 이여
# List Type

- Quiz!
- map함수를 직접 짜 보자!

# 차 마시면서 즐기는 느긋함
# Lazy Evaluation

- let v = 5 + 3을 ghci에 쳤을 때, v에 들어있는 값은 무엇일까?
- let h = head []를 ghci에 쳤을 때, h에 들어있는 값은 무엇일까?

# 차 마시면서 즐기는 느긋함
# Lazy Evaluation

- 값을 요구할 때까지 계산을 최대한 미룬다!
- 에러를 실제로 사용할 때까지 발생시키지 않는다!

# 차 마시면서 즐기는 느긋함
# Lazy Evaluation

- let l = ‘c’:l로 돌아오자
  - (:) :: a -> [a] -> [a]
  - l :: [Char]
  - l의 정체는???

# 차 마시면서 즐기는 느긋함
# Lazy Evaluation

- let l = ‘c’:l
  = ‘c’:(‘c’:l)
  = …
  = ‘c’:(‘c’:(‘c’: …))
- ‘c’로 이루어진 무한리스트!

# 차 마시면서 즐기는 느긋함
# Lazy Evaluation

- 컴퓨터에서 무한이 어떻게 가능할까?
  - 필요할 때 필요한 부분까지만 계산한다!
  - l!!5
  - head l
  - …

# 차 마시면서 즐기는 느긋함
# Lazy Evaluation

- Fibonacii 수열을 계산해보기
- fib a b = a : fib b (a+b)
- fibonacii n = (fib 0 1) !! (n+1)

# Nothing Or Not
# Maybe Type

- 이전에 정의했던 myHead를 생각해보자.
- myHead (x:xs) = x

# Nothing Or Not
# Maybe Type

- myHead []는 에러를 부른다.
- 프로그램이 돌아가는 중간의 에러? 극혐

# Nothing Or Not
# Maybe Type

- 뭔가 '아무것도 아닌 값' 같은 것을 리턴할 수 없을까?

# Nothing Or Not
# Maybe Type

- Haskell은 함수의 결과값이 딱 하나의 타입만 가질 수 있다.
- 따라서
- myHead [] = NULL
  myHead (x:xs) = x
- 같은 함수는 정의할 수 없다

# Nothing Or Not
# Maybe Type

- 어떻게 하면 좋을까?
- 바로 Maybe Type을 쓰면 된다!

# Nothing Or Not
# Maybe Type

- Maybe Type
- myHead [] = Nothing
  myHead (x:xs) = Just x
- Nothing이나 Just x를 포함하는 타입!

# Nothing Or Not
# Maybe Type

- List의 타입이 [a]와 같이 List 안에 들어있는 타입에 따라 결정되듯이
- Maybe 역시 Maybe a와 같이 Just 안에 들어있는 타입에 따라 결정된다

# Nothing Or Not
# Maybe Type

- Just 안에 있는 값을 어떻게 쓸 수 있을까??
- (Just 5) + 3
- (Just 5) + (Just 3)
- 에러!
- Num (Maybe a)는 성립하지 않는다!

# Nothing Or Not
# Maybe Type

- Just 안에 있는 값을 어떻게 쓸 수 있을까??
- Pattern Matching을 사용하면 된다!
- ```
  headAndInc l = case (myHead l) of
    Nothing -> Nothing
    Just x -> Just (x + 1)
  ```
  ```

# Nothing Or Not
# Maybe Type

- Just 안에 있는 값을 어떻게 쓸 수 있을까??
- 매 번 이렇게 써줘야 할까?
- NO! 더 세련된 방법은 Funtor, Applicative, Monad를 통해서

# 일급 객체랑 놀기
# Function As First Class Object

- First class object (일급 객체)?
  - First class citizen (일급 시민) 이라고도 부른다.
  - ALGOL에서 유래한 말
  - 변수에 대입하거나, Function의 인자로 넘기거나, Function이 리턴하는 등 그 언어에서 객체가 할 수 있는 모든 일들을 다 할 수 있는 것들
  - C의 일급 객체는 int, float 등의 primitive type 과 struct, enum 등이 포함된다.
  - 참고 : 이급 객체 (Second class object, Non-first class object)

# 일급 객체랑 놀기
# Function As First Class Object

- Haskell의 First class object
  - 함수!
  - Haskell은 상수나 매개변수가 1개인 함수를 First class object로 취급한다
  - 매개변수가 2개인 함수는요? Currying에 의해 매개변수가 1개인 함수가 된다!

# 일급 객체랑 놀기
# Function As First Class Object

- 함수가 First class면 무엇이 가능할까?
- map 함수와 같이 함수를 인자로 받기!
- Currying에서 해왔던 것처럼 함수를 리턴하기!

# 일급 객체랑 놀기
# Function As First Class Object

- map 함수와 같이 함수를 인자로 받기!
  - map sum [[1,2,3],[4,5],[6]]

# 일급 객체랑 놀기
# Function As First Class Object

- Currying 에서 해왔던 것처럼 함수를 리턴하기!
  - :t (+)
  - :t (+) 5
  - :t ((+) 5) 4
  - :t map sum

# 일급 객체랑 놀기
# Function As First Class Object

- map에 넘길 함수를 Curring으로 만들어내기
- map :: (a->b) -> [a] -> [b]
- map은 인자 하나짜리 함수를 받는다!

# 일급 객체랑 놀기
# Function As First Class Object

- map에 넘길 함수를 Curring으로 만들어내기
- pow x y = x^y
- 라는 함수가 있다고 하자.

# 일급 객체랑 놀기
# Function As First Class Object

- map에 넘길 함수를 Curring으로 만들어내기
- 어떻게 pow를 map에게 넘길 수 있을까?
- map (pow 2) [1,2,3]
- 과 같이 Curring으로 넘기면 된다.

# 일급 객체랑 놀기
# Function As First Class Object

- map에 넘길 함수를 Curring으로 만들어내기
- 그냥 넘기면 어떻게 될까?
- map pow [1,2,3]
- pow :: (Integral b, Num a) => a -> b -> a
- pow :: (Integral b, Num a) => a -> (b -> a)
- map pow :: (Integral b, Num a) => [a] -> [b->a]

# 일급 객체랑 놀기
# Function As First Class Object

- map에 넘길 함수를 Curring으로 만들어내기
- 그냥 넘기면 어떻게 될까?
- 함수의 리스트가 된다!
- 함수는 출력할 수 없기 때문에 에러가 뜬다

# 바보야 니 이름은 개발자가 버렸어
# Lambda Function

- map같은 함수들에게 넘길 인자를 매 번 정의해야 할까?
- mult2AndAdd1 a = 2*a + 1
- 이런 간단한 함수도 일일이 길게 이름을 붙여야 할까?

# 바보야 니 이름은 개발자가 버렸어
# Lambda Function

- 일회용 간단한 함수를 이름을 붙이지 않고 사용하는 방법
- Lambda Function

# 바보야 니 이름은 개발자가 버렸어
# Lambda Function

- Haskell에서 Lambda Function을 정의하는 방법
- `\x y -> 2*x + y`
- `\x -> x^2 - x`
- …

# 따르릉 따르르릉
# Simple Example

- Haskell을 사용해 간단한 전화번호부를 만들어보자
- phone = [(“JunYoung”, “010-1111-1111”),
  (“SungHun”, “011-2345-6789”),
  (“MiNyoung”, “032-0000-3333”)]

# 따르릉 따르르릉
# Simple Example

- Haskell을 사용해 간단한 전화번호부를 만들어보자
- 전화번호부에서 번호를 어떻게 찾을 수 있을까?
- List의 첫번째 원소에서 이름을 빼내 찾으려는 이름과 비교한다.
  List의 나머지 번호들에서 이름을 찾는다.

# 따르릉 따르르릉
# Simple Example

- Haskell을 사용해 간단한 전화번호부를 만들어보자
- 찾았을 경우에는 폰 번호를 주면 되는데, 못 찾았을 경우에는?
- Maybe Type을 사용하자
- 못 찾았을 경우에는 Nothing, 찾았을 경우에는 Just 전화번호

# 따르릉 따르르릉
# Simple Example

- Haskell을 사용해 간단한 전화번호부를 만들어보자
- findPhone :: [(String, String)] -> String -> Maybe String

# 따르릉 따르르릉
# Simple Example

- Haskell을 사용해 간단한 전화번호부를 만들어보자
- ```
  findPhone pb n = case pb of
    (k, v):xs | k == n      -> Just v
              | otherwise -> findPhone xs n
    []                        -> Nothing
  ```
  ```

# 따르릉 따르르릉
# Simple Example

- Haskell을 사용해 간단한 전화번호부를 만들어보자
- Haskell이 가진 함수들을 사용해 다른 방식으로 짜보자
- ```
  findPhone pb n | (_, p):_ <- pbFound = Just p
                 | otherwise           = Nothing
    where pbFound = filter (\(k, v) -> k == n) pb
  ```
  ```

# Practice

- 1. Quick Sort와 Merge Sort를 Haskell로 짜보자
  - quickSort :: (Ord a) => [a] -> [a]
  - mergeSort :: (Ord a) => [a] -> [a]
    merge :: (Ord a) => [a] -> [a] -> [a]
- 2. List가 회문 (앞에서부터 읽어도 뒤에서부터 읽어도 똑같은 글)인지 확인하는 함수를 짜보자
  - palindromeChecker :: (Eq a) => [a] -> Bool

# Practice

- 3. List 두 개를 받아서 첫번째 리스트가 두번째 리스트에 포함되는지 확인하는 함수를 짜 보자.
  이를테면 [2,3]은 [2,3,4]에 포함되지만, [2,4]는 [2,3,4]에 포함되지 않는다. ([2,4]는 [2,4,2,3]에는 포함된다.)
  - isPrefixOf :: (Eq a) => [a] -> [a] -> Bool
    isIn :: (Eq a) => [a] -> [a] -> Bool

# What’s Next?

- Haskell의 Type Class
- Haskell의 User Defined Type
- Functor – Applicative – Monad 상속관계
- Haskell의 Module
- Haskell의 Input-Output
- Haskell로 컴파일된 프로그램 만들기