

# 프로그래머를 위한 하스켈



# 이 발표는...

- 프로그래밍 기초를 다룹니다 ✗
- 함수형 프로그래밍을 다룹니다 ✗
- 모나드를 설명합니다 ✗

# 이 발표는...

- 프로그래밍 언어, 하스켈을 설명합니다 ●
- 기초적인 문법과 개념들을 배웁니다 ●

# 배경

# 하스켈은 무엇인가?

- 함수형 언어
- 자연 평가
- 순수 함수형 프로그래밍 패러다임
- 정적/강 타입

# 왜 하스켈인가?

- 고차원의 안전한 추상화
- 수준 높은 타입 시스템
- ~~HIP~~

# 더 많은 배경지식은...

[Downloads](#) [Community](#) [Documentation](#) [Donate](#)

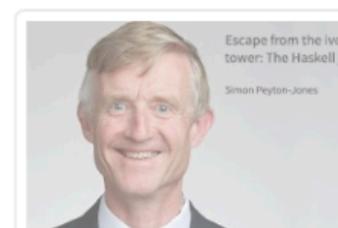


Declarative, statically typed code.

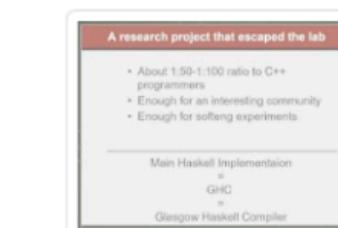
```
primes = filterPrime [2..]
where filterPrime (p:xs) =
      p : filterPrime [x | x <- xs, x `mod` p /= 0]
```

The screenshot shows a Haskell code editor interface. On the left, under 'Try it!', there is a text input field with the placeholder 'Type Haskell expressions in here.' and a λ symbol. On the right, under 'Got 5 minutes?', there is a text input field with the placeholder 'Type help to start the tutorial.' and a snippet of Haskell code: `3 * 3`. Below the code editor is a large blue button with the URL <https://www.haskell.org>.

## Videos



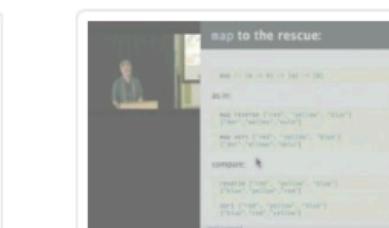
[Escape from the ivory tower: The Haskell journey, by Simon Peyton-Jones](#)



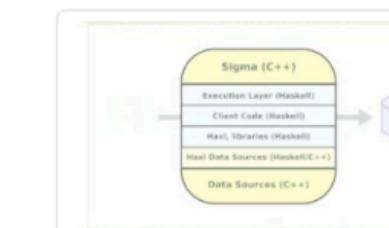
[Haskell taketh away: limiting side effects for parallel programming, by Ryan Newton](#)



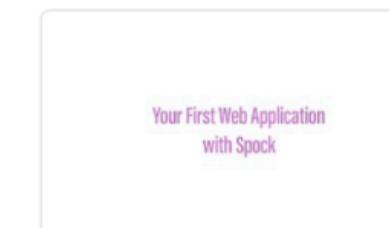
[Production Haskell, by Reid Draper](#)



[Haskell Amuse-Bouche, by Mark Lentczner](#)



[Haskell is Not For Production and Other Tales, by Katie Miller](#)



[Your First Web Application with Spock, by Oskar Wickström](#)

# 하스켈 기초

# 코드의 세 가지 유형

- 표현식(Expression): 계산 결과로 값이 도출되는 문장  
ex: 수식, 함수 호출
- 구문(Statement): 컴퓨터의 상태 따위를 조작하는 문장  
ex: 조건문, 반복문, goto
- 선언문(Declaration): 개념 따위를 정의하는 문장  
ex: 함수 선언, 변수 선언, 타입 선언

# 코드의 세 가지 유형

- **표현식(Expression)**: 계산 결과로 값이 도출되는 문장  
ex: 수식, 함수 호출
- **구문(Statement)**: 컴퓨터의 상태 따위를 조작하는 문장  
ex: 조건문, 반복문, goto
- **선언문(Declaration)**: 개념 따위를 정의하는 문장  
ex: 함수 선언, 변수 선언, 타입 선언

# 표현식

## 최소 단위, 리터럴



`1` -- 수는 숫자를 통해 표현합니다.

`'a'` -- 글자는 따옴표로 감싸서 표현합니다.

`"abc"` -- 문자열은 쌍따옴표로 감싸서 표현합니다.

`(1, 2, 'a')` -- 튜플은 소괄호로 시작하고 완결지으며, 각 요소는 쉼표로 구분합니다.

`[1, 2, 3, 4]` -- 리스트는 대괄호로 시작하고 완결지으며, 마찬가지로 각 요소는 쉼표로 구분합니다.

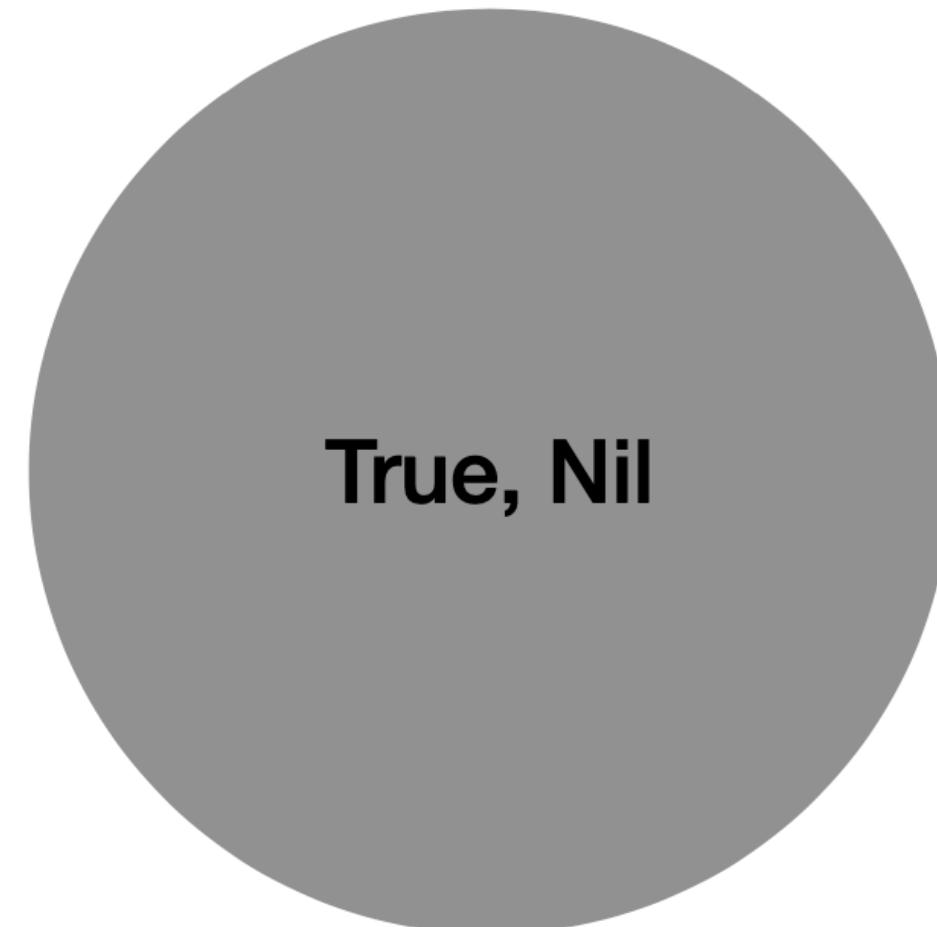
`\x -> x` {- 람다 함수(익명 함수 표현식)은 역슬래쉬로 시작하여,  
그 뒤로 매개변수를 나열한 다음 화살표 뒤에 식을 작성합니다. -}

`\x y -> x + y` -- 매개변수는 빈칸으로 구별합니다.

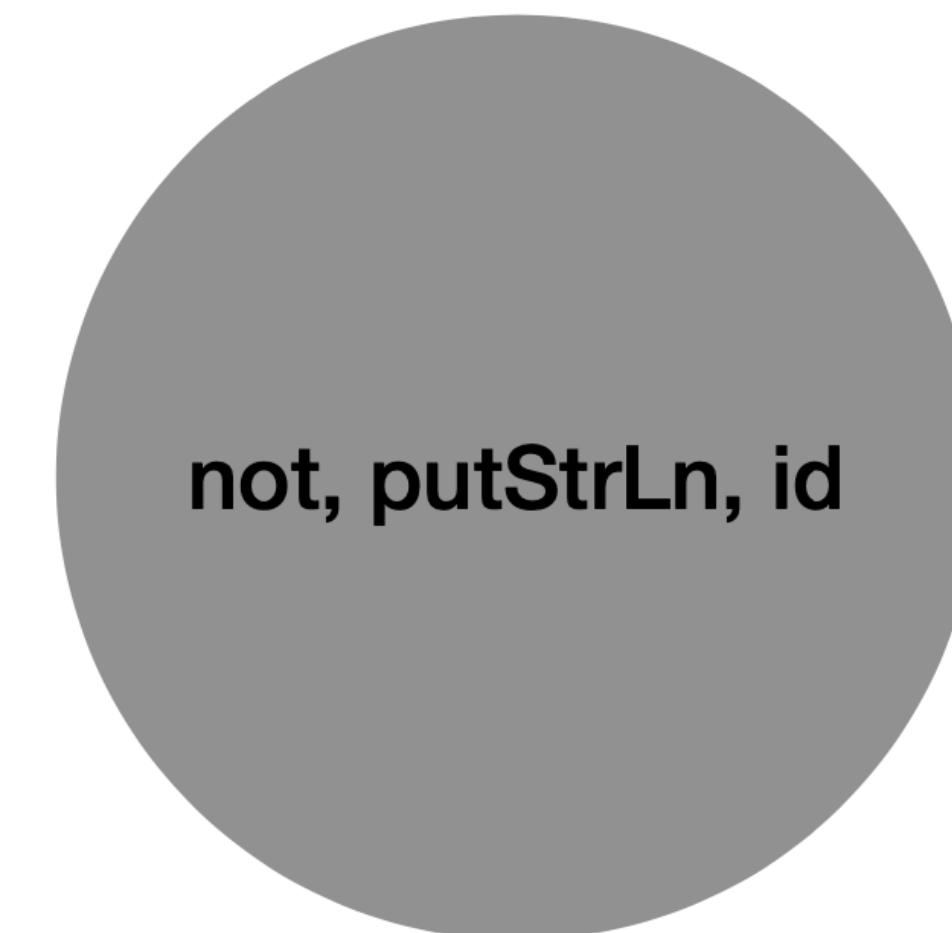
# 표현식

또 다른 최소 단위, 참조

값 생성자에 대한 참조



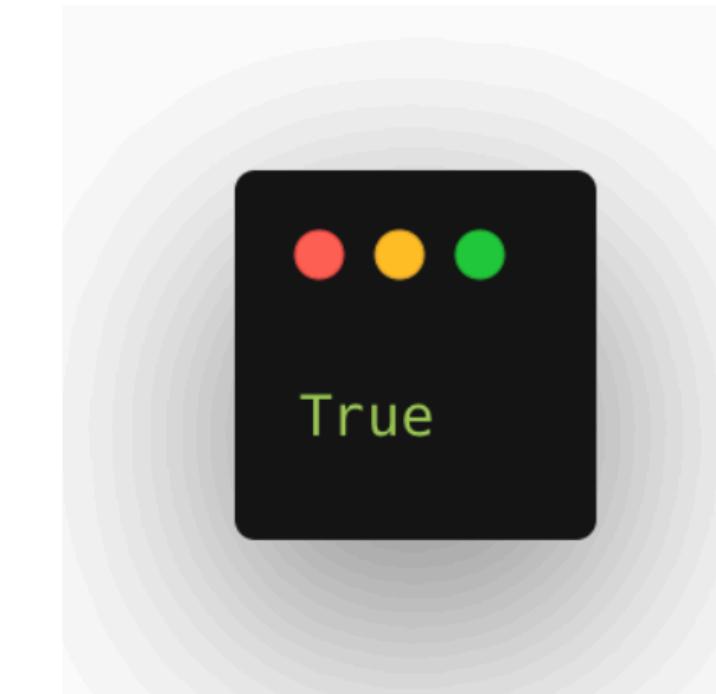
그 외 일반적인 값에 대한 참조



# 표현식

## 값 생성자

- 클래스의 생성자 함수에 대응하는 함수
- 값을 가지고 새로운 값을 생성하는 함수
- 값을 받지 않는 경우도 존재. 이 경우에는 리터럴 처럼 보임.



# 표현식

## 함수 호출



```
not True
id 1
subtract 3 2
map (\x -> x + 1) [1, 2, 3, 4]
-- [ 함수 ] [ 전달인자n ] [ 전달인자n ] ... [ 전달인자n ]
```

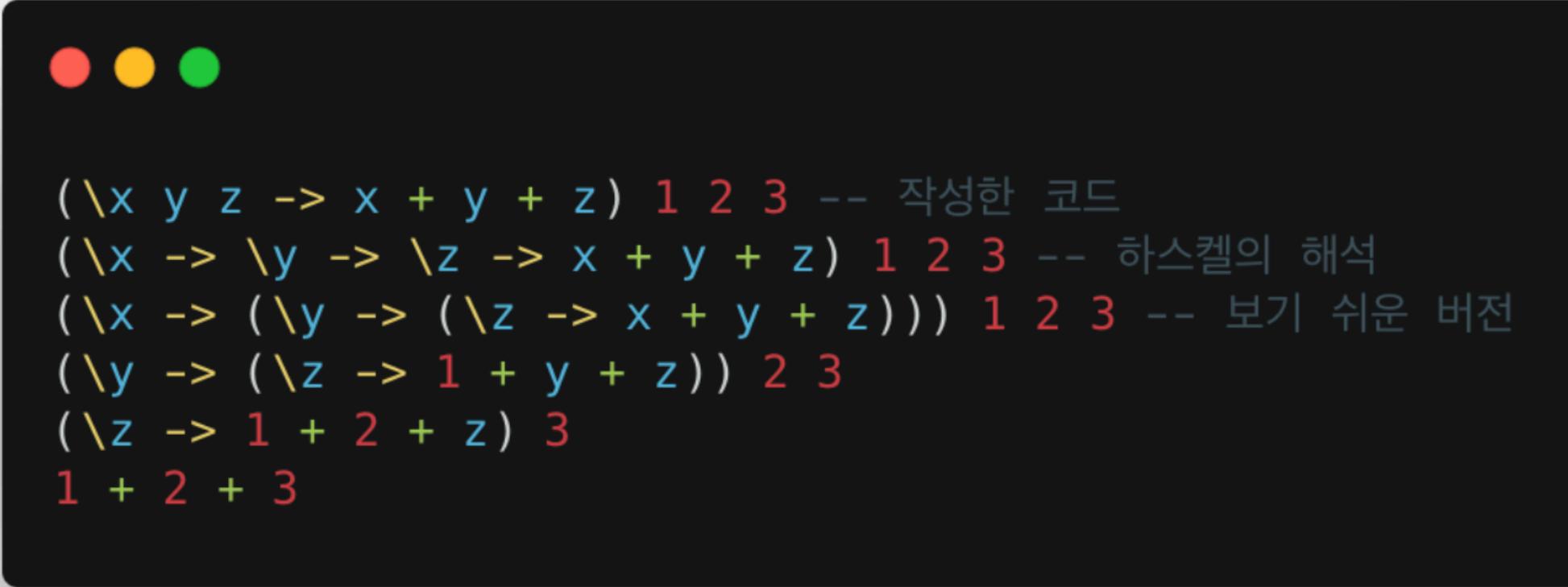
# 표현식

## 함수



# 표현식

## 함수 호출 해석 과정



```
(\x y z -> x + y + z) 1 2 3 -- 작성한 코드
(\x -> \y -> \z -> x + y + z) 1 2 3 -- 하스켈의 해석
(\x -> (\y -> (\z -> x + y + z))) 1 2 3 -- 보기 쉬운 버전
(\y -> (\z -> 1 + y + z)) 2 3
(\z -> 1 + 2 + z) 3
1 + 2 + 3
```

# 표현식

## 기본적인 타입들



[값] :: [타입] -- 타입은 콜론을 사용하여 지정한다.

```
1 :: Int
'a' :: Char
"abc" :: String
True :: Bool
(1, 'a') :: (Int, Char)
[1, 2, 3] :: [Int]
(\x -> x + 1) :: Int -> Int
(\x -> y -> x + y) :: Int -> Int -> Int
-- Int -> Int -> Int는 Int -> (Int -> Int)와 같습니다.
```

# 선언문

## supercombinator



# 선언문

## 데이터 타입 정의

```
● ● ●  
data [타입명] = [값 생성자] | [값 생성자] | ...  
  
data Bool = True | False  
  
data IntPair = IntPair Int Int  
  
data IntList = Nil | Push Int IntList
```

# 선언문

데이터 타입 정의

# 선언문

## 데이터 타입 정의

실수

# 선언문

## 데이터 타입 정의

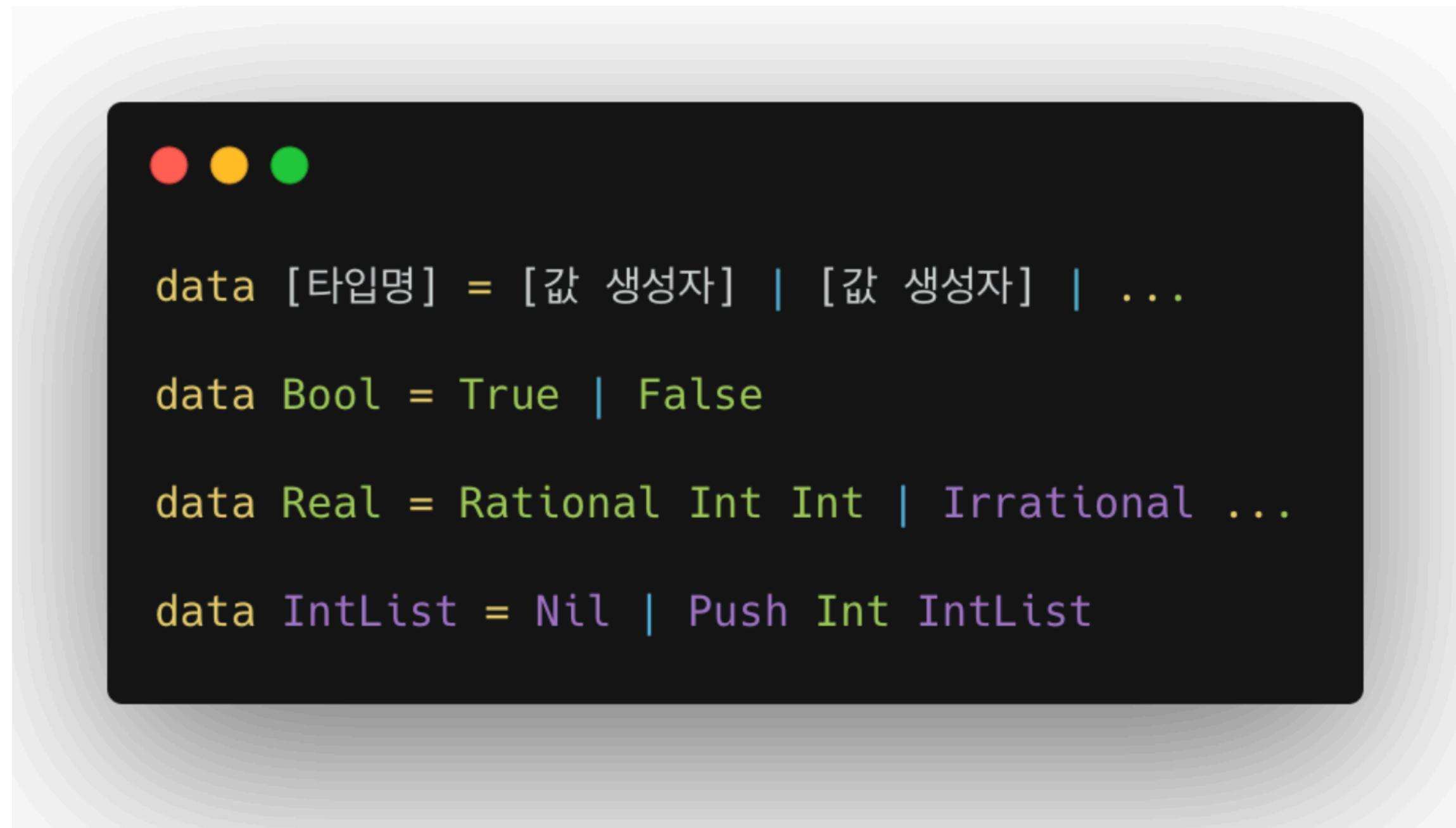
실수

유리수

무리수

# 선언문

## 데이터 타입 정의



# 표현식

## case 식

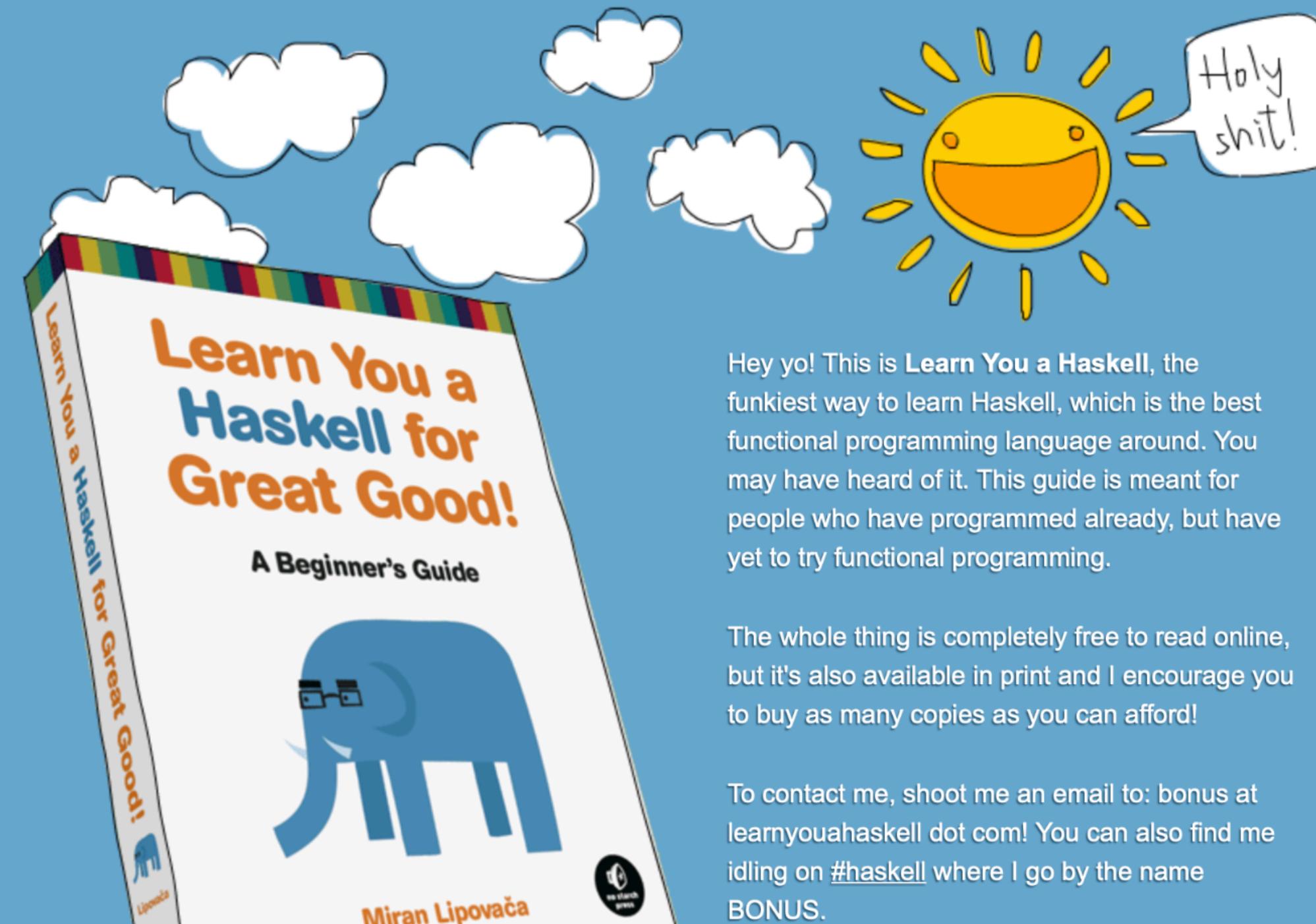
```
case [조사할 타입의 값] of
    [패턴] -> [표현식]
    [패턴] -> [표현식]
    ...
data IntList = Nil | Push Int IntList

sum :: IntList -> Int
sum xs = case xs of
    Nil -> 0
    Push x xs' -> x + sum xs'
```

더 나아가기

# Learn You a Haskell for Great Good!

a.k.a 코끼리책



Hey yo! This is **Learn You a Haskell**, the funkiest way to learn Haskell, which is the best functional programming language around. You may have heard of it. This guide is meant for people who have programmed already, but have yet to try functional programming.

The whole thing is completely free to read online, but it's also available in print and I encourage you to buy as many copies as you can afford!

To contact me, shoot me an email to: bonus at learnyouahaskell dot com! You can also find me idling on [#haskell](#) where I go by the name BONUS.

<http://learnyouahaskell.com>

# Thinking With Types

## Thinking with Types

Type-Level Programming in Haskell

Buy Now

About

Contents

Contact

The book that takes you from a competent Haskell programmer to one whose compiler does the work for you.

### About the Book

<https://thinkingwithtypes.com>

Hi there. My name is Sandy Maguire and you might know me from [my blog](#) where I write about type-level programming in Haskell.

This book came to be when I realized that learning type-level programming was harder than it needed to be. While the information was available, it wasn't organized. The best way to learn this stuff was to read through dozens of blog posts—each of which described a small piece of the puzzle—and attempt to synthesize it for yourself.

I knew we could do better.

Without further ado, I'd like to introduce **Thinking with Types: Type-Level Programming in Haskell**—the comprehensive guide to unleashing the power of Haskell's type-system. In it, we'll cover topics like erasing boilerplate with GHC.Generics, building extensible data-types, and how to generate error messages meant for humans when things go wrong.

Buy Now

# 기타 문서

- 하스켈 위키: <https://wiki.haskell.org/Haskell>
- GHC 확장: <https://ghc.readthedocs.io/en/8.0.2/lang.html>
- Hoogle: <https://hoogle.haskell.org>
- Stack: <https://docs.haskellstack.org/en/stable/README/>
- HLS(Haskell Language Server):  
<https://github.com/haskell/haskell-language-server>
- Category Theory for Programmers:  
<https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/>

# 감사합니다!

