Introduction to Haskell

Artem Ohanjanyan

LvivHaskell

24.11.2019

Haskell

Haskell

- Pure functional
- Lazy
- Statically typed
- With strong types and type inference
- Compiled (with an interpreter)

Why learn Haskell

Why learn Haskell

- Probably not to find a job (0 vacancies in Ukraine ☹)

Why learn Haskell

- Probably not to find a job (0 vacancies in Ukraine ☹)
- It's a challenge

Why learn Haskell

- Probably not to find a job (0 vacancies in Ukraine ☹)
- It's a challenge
- Expand your mind with no illegal substances

Why learn Haskell

- Probably not to find a job (0 vacancies in Ukraine ☹)
- It's a challenge
- Expand your mind with no illegal substances
- Familiarize yourself with FP before everyone else

Why learn Haskell

- Probably not to find a job (0 vacancies in Ukraine ☹)
- It's a challenge
- Expand your mind with no illegal substances
- Familiarize yourself with FP before everyone else
- Write programs (ecosystem is surprisingly good)

It's a good language

It's a good language

- Expressive

It’s a good language

- Expressive
- Reliable

It's a good language

- Expressive
- Reliable
- Easy to support

Used for

Used for

- Compilers

Used for

- Compilers
- Code analyzers

Used for

- Compilers
- Code analyzers
- Blockchain ¯_(ツ)_/¯

Used for

- Compilers
- Code analyzers
- Blockchain ¯_(ツ)_/¯
- Web backend

Used for

- Compilers
- Code analyzers
- Blockchain ¯_(ツ)_/¯
- Web backend
- ...

Functional

Functional

- Broadly speaking, functional = Haskell

Functional

- Broadly speaking, functional = Haskell
- Functions are first-class citizens

Functional

- Broadly speaking, functional = Haskell
- Functions are first-class citizens
- Program is an expression rather than a list of instructions

Pure

Pure

- Immutability

Pure

- Immutability
- No side effects

Pure

- Immutability
- No side effects
- Result of the function is always the same

Pure

- Immutability
- No side effects
- Result of the function is always the same

Useful for:

Pure

- Immutability
- No side effects
- Result of the function is always the same

Useful for:

- Equational reasoning

Pure

- Immutability
- No side effects
- Result of the function is always the same

Useful for:

- Equational reasoning
- Refactoring

Pure

- Immutability
- No side effects
- Result of the function is always the same

Useful for:

- Equational reasoning
- Refactoring
- Parallelism

Pure

- Immutability
- No side effects
- Result of the function is always the same

Useful for:

- Equational reasoning
- Refactoring
- Parallelism
- Easier reasoning

Lazy

Lazy

- User-defined control structures

Lazy

- User-defined control structures
- Infinite data structures

Lazy

- User-defined control structures
- Infinite data structures
- Efficient functional programming for free (e.g. efficient higher-order functions)

Lazy

- User-defined control structures
- Infinite data structures
- Efficient functional programming for free (e.g. efficient higher-order functions)
- However, sometimes it makes things more complicated

Types

Types

- Algebraic Data Types

Types

- Algebraic Data Types
- Powerful parametric polymorphism

Types

- Algebraic Data Types
- Powerful parametric polymorphism
- Higher-kinded types

Types

- Algebraic Data Types
- Powerful parametric polymorphism
- Higher-kinded types
- Type classes

Types

- Algebraic Data Types
- Powerful parametric polymorphism
- Higher-kinded types
- Type classes
- Type inference

Types

- Algebraic Data Types
- Powerful parametric polymorphism
- Higher-kinded types
- Type classes
- Type inference
- Many more

Types

- Algebraic Data Types
- Powerful parametric polymorphism
- Higher-kinded types
- Type classes
- Type inference
- Many more
- FP + Types = Win

Let's jump right in

Let's jump right in

- Treap a.k.a. randomized binary search tree
- Regular expression engine

Installation

`https://www.haskellstack.org`

End

End

Thanks for your attention!