Introduction to Haskell

Functional programming in Haskell

Ivan Trepakov

NSU Sys.Pro

What is Haskell?

What is Haskell?

Functional

- Functions as first-class citizens
- Higher order functions
- Declarative style

What is Haskell?

Functional

- Functions as first-class citizens
- Higher order functions
- Declarative style

Pure

- Side-effect separation
- Equational reasoning
- Simplified parallelism

What is Haskell?

Functional

- Functions as first-class citizens
- Higher order functions
- Declarative style

Lazy

- Infinite data structures
- Compositional programming style
- Tricky to evaluate complexity

Pure

- Side-effect separation
- Equational reasoning
- Simplified parallelism

What is Haskell?

Functional

- Functions as first-class citizens
- Higher order functions
- Declarative style

Pure

- Side-effect separation
- Equational reasoning
- Simplified parallelism

Lazy

- Infinite data structures
- Compositional programming style
- Tricky to evaluate complexity

Statically typed

- “If a program compiles, it probably works”
- Expressive type system
- Type inference

Anatomy of declaration

Here is sample Haskell declaration:

```
x :: Int    -- Type declaration
x = 42      -- Value declaration
```

`name = expression` is a *binding* (not assignment)

- `::` reads as “has type”
- `=` reads as “defined to be”

Anatomy of declaration

Here is sample Haskell declaration:

```
x :: Int    -- Type declaration
x = 42      -- Value declaration
```

`name = expression` is a *binding* (not assignment)

- `::` reads as “has type”
- `=` reads as “defined to be”

Multiple declarations with the same name are not allowed!
Compiler will let us know about it with error:

```
Multiple declarations of 'x'
```

Anatomy of declaration

Here is sample Haskell declaration:

```
x :: Int    -- Type declaration
x = 42      -- Value declaration
```

`name = expression` is a *binding* (not assignment)

- `::` reads as “has type”
- `=` reads as “defined to be”

Multiple declarations with the same name are not allowed!
Compiler will let us know about it with error:

```
Multiple declarations of 'x'
```

What does this declaration mean?
And what is its type if any?

```
y = y + 1
```

Anatomy of declaration

Here is sample Haskell declaration:

```
x :: Int    -- Type declaration
x = 42      -- Value declaration
```

`name = expression` is a *binding* (not assignment)

- `::` reads as “has type”
- `=` reads as “defined to be”

Multiple declarations with the same name are not allowed!
Compiler will let us know about it with error:

```
Multiple declarations of 'x'
```

What does this declaration mean?
And what is its type if any?

```
y = y + 1
y :: Int
```

Q&A