



Haskell

Elena Schmitt

WHAT IS HASKELL?

Purely functional language

- No side-effects

Statically typed

- Types of expressions are checked before the program runs and do not change during execution

Higher-order Functions

- Can take in other functions as parameters and return functions

Lazy Evaluation

- Expressions not evaluated until needed



History of Haskell

- Named after mathematician Haskell Brooks Curry
- In 1987 a committee was formed to create a common functional programming language
- Haskell 98 released and a standard library was published
- In 2006 work on a successor to Haskell 98 began
- Haskell 2010 is released
- GHC Haskell is the implementation widely used today

1987

FPCA'87 CONFERENCE

Haskell development committee was organized there.

1990

THE FIRST HASKELL REPORT

Document, which describes the motivation for creating the language, the nature of the language and the process of its creation, for which the committee is responsible.

1992

GLASGOW HASKELL COMPILER

The creation of GHC – open-source and the most commonly-used native code compiler for Haskell language.

1994

HASKELL.ORG

The main information page about Haskell basics was created.

1996

THE HASKELL VER. 1.3 REPORT

In terms of technical changes, it was the most significant release of Haskell after 1.0.

2005

HASKELL WORKSHOP

First ideas on how the new standard should be designed.

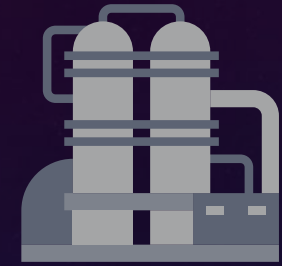
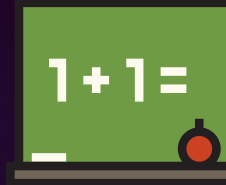
2010

THE HASKELL 2010 RELEASE

The most important point in the modern history of Haskell development and also it is the current standard for most Haskell developers.

START

WHERE IT'S USED



Pure Functional

Strong Type System

Parallel Programming

Reasoning

Efficiency

- **TEACHING**
 - Good for learning programming concepts.
- **RESEARCH**
 - Strength in reasoning, useful for statistics.
 - Ability to handle large datasets.
- **INDUSTRY**
 - Used in areas like aerospace, finance, and healthcare.
 - Good at handling real-time data streams.

Monads



RETURN
 $\text{RETURN} :: A \rightarrow M A$

BIND
 $(\gg=) :: M A \rightarrow (A \rightarrow M B) \rightarrow M B$

THEN
 $(\gg) :: M A \rightarrow M B \rightarrow M$



```
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    (>>)  :: m a -> m b -> m b
    return :: a -> m a
```

- m monadic type constructor (Maybe, IO, Reader, etc.).
- a, b type variables- a is the type input to the monad, b is the type input you get out of the monad

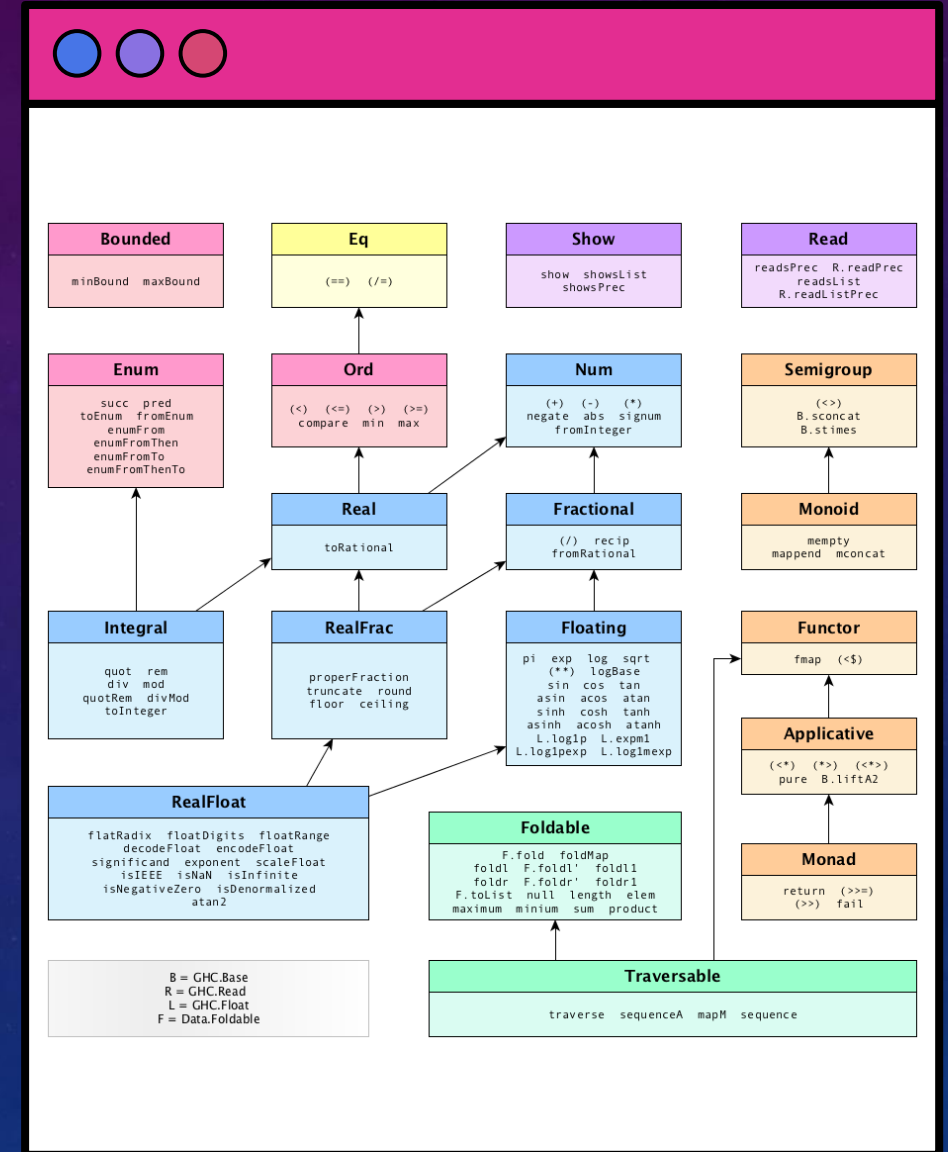
Return: Wraps a value in a monad

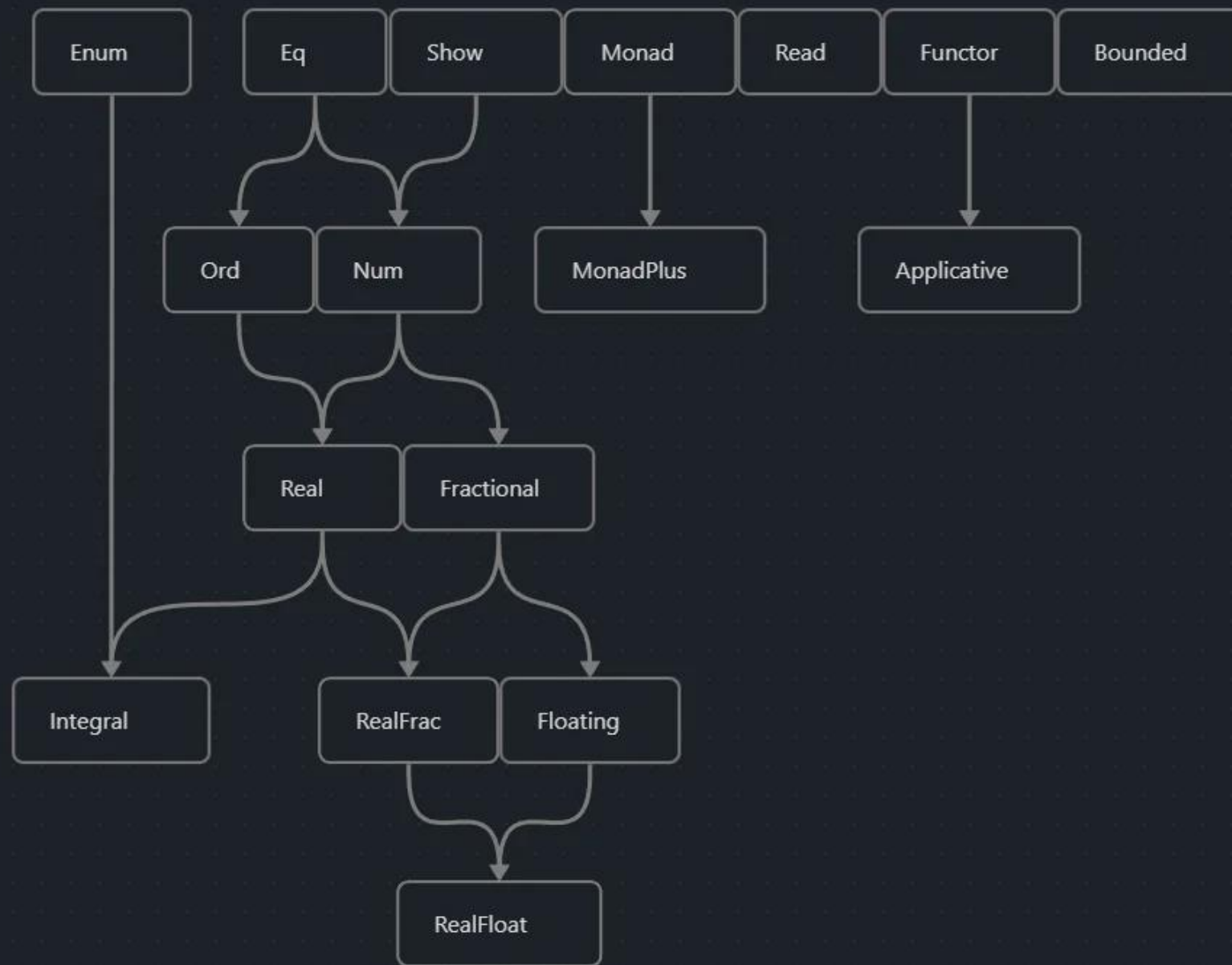
Bind: Allows for chaining together monadic operations

Then: Similar to bind, but ignores the result of the first monadic operation

TYPECLASSES

- Define a set of methods that are shared between multiple types
 - Only able to accept certain types for a function
- Compile-time type restraints
 - Types are checked before running
- Have set operations so that they behave consistently
- Inheritance allows for creating more specific types
- Polymorphism





Demonstration

SOURCES

<https://cs.lmu.edu/~ray/notes/introhaskell/>

<https://scrapingrobot.com/blog/haskell-programing-language/#programming-in-haskell>

<https://serokell.io/blog/haskell-history>

<https://medium.com/sourcescribes/what-is-haskell-853b6949a800>

<https://wiki.haskell.org/Monad>

