

Reader Monad (11A)

Copyright (c) 2016 - 2018 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

[Haskell in 5 steps](https://wiki.haskell.org/Haskell_in_5_steps)

https://wiki.haskell.org/Haskell_in_5_steps

Reader, ReaderT, MonadReader

MonadReader is not a **monad**.

MonadReader is the **class** of all monads that can read from some **environment**.

Reader is an **instance** of **MonadReader**, but doesn't do anything else, except that it only acts as a **reader**,

ReaderT is the **reader monad transformer**, which adds a read-only environment (**r**) to the given **monad** (**m**).

Generally, **instance monads** are defined as a **monad transformer stack** with a **ReaderT** in somewhere

MonadReader – class

Reader – an **instance**

ReaderT – another **instance**

<https://stackoverflow.com/questions/39366069/reader-and-monadreader>

Reader v.s. ReaderT

Reader has ReaderT only applied to the trivial identity monad:

```
type Reader r = ReaderT r Identity
```

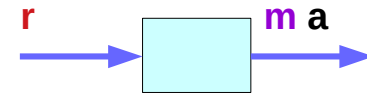
MonadReader includes Reader itself,
but also MaybeT (ReaderT Int (ListT IO)).

```
newtype ReaderT r m a = ReaderT { runReaderT :: r -> m a }
```

```
type Reader r a = ReaderT r Identity a (1)
```

```
= Reader { runReader :: r -> Identity a }
```

```
newtype Reader r a = Reader { runReader :: r -> a } (2)
```



<https://stackoverflow.com/questions/39366069/reader-and-monadreader>

Reader Monad

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

Reader Monad type definition

A **Reader** is a data type that encapsulates an **environment**.

The **runReader** function

takes an **environment** (a reader of type **r**)

and runs it (**r -> a**),

producing the **result** of type **a**.

```
newtype Reader r a = Reader { runReader :: r -> a }
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

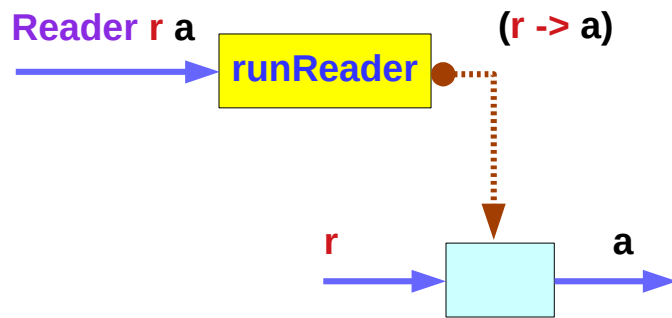
Implementing **ReaderT**

Reader Monad runReader Function

```
newtype Reader r a = Reader { runReader :: r -> a }
```

With the **record** syntax, the type of **runReader** is actually

```
runReader :: Reader r a -> (r -> a)
```



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

Reader Monad Record Syntax

```
newtype Reader r a = Reader { runReader :: r -> a }
```



```
Reader function r -> a
```

```
Reader $ r -> r + 1
```

```
Reader $ r -> "reader2: " ++ (show r)
```

```
data Car = Car { company :: String, model :: String, year :: Int }
```

```
(1) Car { company = "Ford", model = "Mustang", year = 1967 }
```

```
(2) Car "Ford" "Mustang" 1967 -- no commas
```

initializing a record

```
Car "Ford" "Mustang" 1967
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

Reader Monad Example

Consider a **Reader** that increments its **environment value**,
returning the same type:

```
reader1 :: Num r => Reader r r
```

```
reader1 = Reader $ r -> r + 1
```

```
runReader reader1 100
```

```
> 101
```

Another **Reader** that converts its environment value into a **string**:

```
reader2 :: Show r => Reader r String
```

```
reader2 = Reader $ r -> "reader2: " ++ (show r)
```

```
runReader reader2 100
```

```
> "reader2: 100"
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

Reader Monad – a runReader implementation

`runReader` is just taking the function `(r -> a)`
out of the data type `Reader r a`

a possible implementation

```
runReader' :: Reader r a -> (r -> a)
```

```
runReader' (Reader f) = f
```

```
reader1 :: Reader r r
```

```
reader1 = Reader $ r -> r + 1
```

```
runReader' reader1 100
```

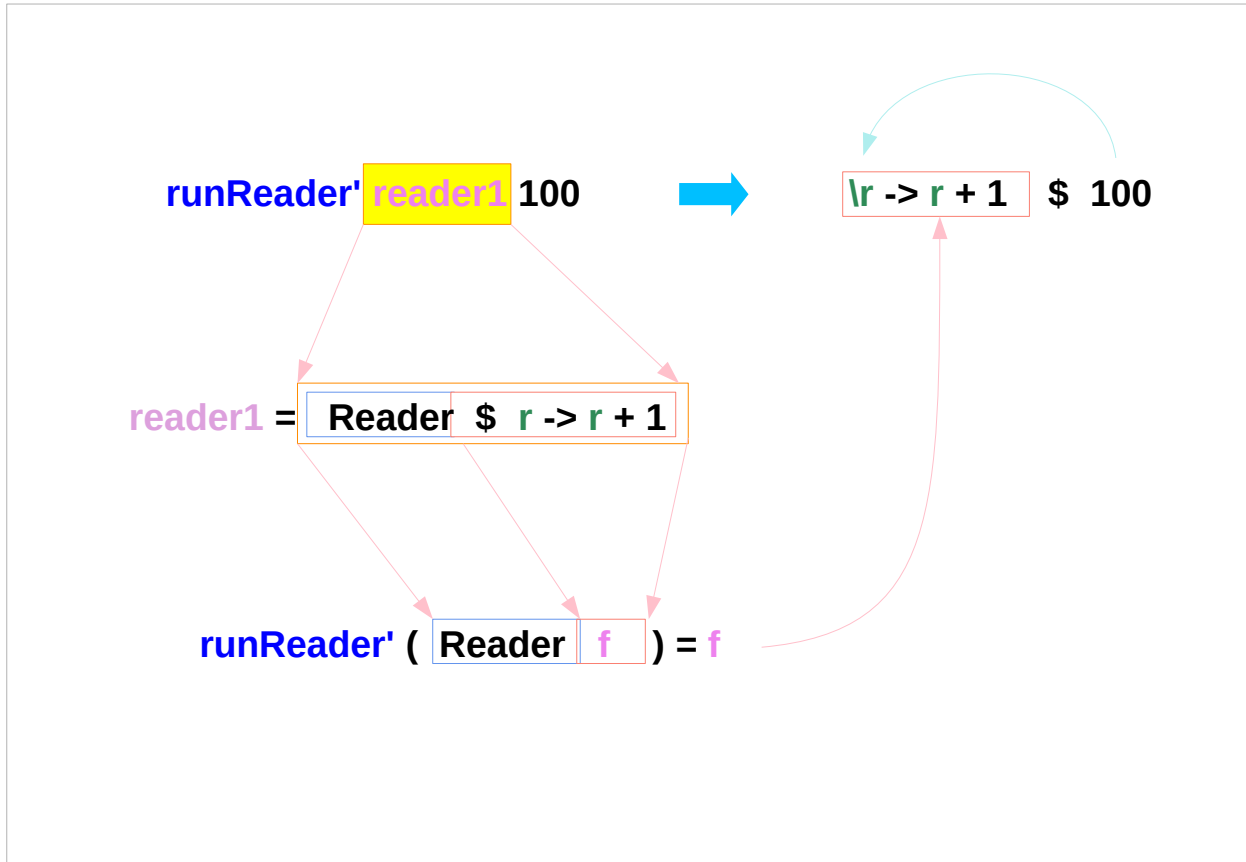
```
> 101
```

`f :: (r -> a)` -- function type

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing `ReaderT`

Reader Monad – pattern matching in runReader



`runReader' :: Reader r a -> (r -> a)`

`runReader' (Reader f) = f`

`reader1 :: Reader r r`

`reader1 = Reader $ r -> r + 1`

`runReader' reader1 100`

`> 101`

Reader ≠ Reader
Type Data

Reader Monad – implementation (I)

make **Reader** an instance of **Monad**:

```
newtype Reader r a = Reader { runReader :: r -> a }
```

```
instance Monad (Reader r) where
```

```
-- return :: a -> Reader r a
```

```
return a = Reader $ \_ -> a
```

```
-- (>>=) :: forall a b . Reader r a -> (a -> Reader r b) -> Reader r b
```

```
m >>= k = Reader $ \r -> runReader (k (runReader m r)) r
```

```
-- Reader $ \r -> runReader (k (runReader m r :: a) :: Reader r b) r
```

comparison with (->) r monad

$$(m \gg= k) r = k (m r) r$$

m : a function (a monadic value)

r : a value

m r : a value (the result)

k (m r) : a function (a monadic value)

k (m r) r : a value (the result)

m :: Reader r

runReader m r :: a

runReader (k (runReader m r)) r :: b

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

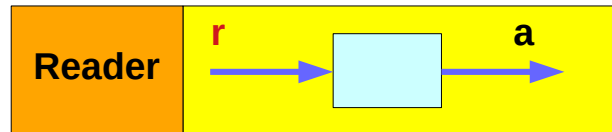
Implementing **ReaderT**

Reader Monad – implementation (I) – runReader m

```
newtype Reader r a = Reader { runReader :: r -> a }
```

```
m >>= k = Reader $ \r -> runReader (k (runReader m r)) r
```

$m :: \text{Reader } r$



$\text{runReader } m :: r \rightarrow a$

$\text{runReader } m \ r :: a$



comparison with (->) r monad

$(m \gg= k) \ r = k \ (m \ r) \ r$

m : a function (a monadic value)

r : a value

$m \ r$: a value (the result)

$k \ (m \ r)$: a function (a monadic value)

$k \ (m \ r) \ r$: a value (the result)

$m :: \text{Reader } r$

$\text{runReader } m \ r :: a$

$\text{runReader } (k \ (\text{runReader } m \ r)) \ r :: b$

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

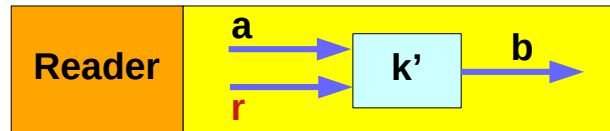
Reader Monad – implementation (I) – runReader k

```
newtype Reader r a = Reader { runReader :: r -> a }
```

```
m >>= k = Reader $ \r -> runReader (k (runReader m r)) r
```

```
m >>= k, k :: Reader r
```

```
k' :: a -> r -> b
```



```
runReader m :: r -> a
```

```
runReader m r :: a
```

```
runReader k :: a -> r -> b
```

```
k' :: a -> r -> b
```

```
runReader (k (runReader m r)) :: r -> b
```

```
runReader (k (runReader m r)) r :: b
```

comparison with (->) r monad

$$(m \gg= k) r = k (m r) r$$

m : a function (a monadic value)

r : a value

$m r$: a value (the result)

$k (m r)$: a function (a monadic value)

$k (m r) r$: a value (the result)

```
m :: Reader r
```

```
runReader m r :: a
```

```
runReader (k (runReader m r)) r :: b
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

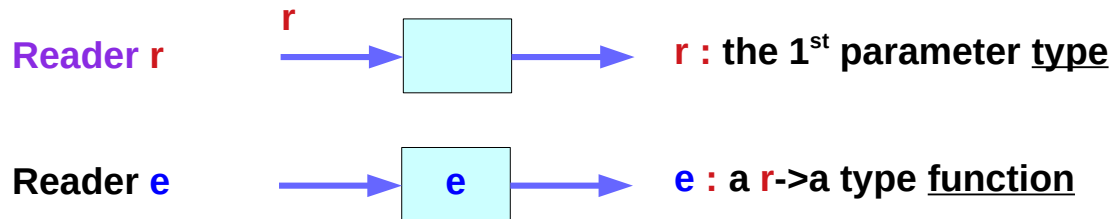
Reader Monad – implementation (II)

```
newtype Reader r a = Reader { runReader :: r -> a }
```

```
instance Monad (Reader r) where
```

```
  return a      = Reader $ \r -> a
```

```
  (Reader e) >>= f = Reader $ \r -> runReader (f (e r)) r
```



comparison with $(\rightarrow) r$ monad

$(m \gg= f) r = f (m r) r$

m : a function (a monadic value)

r : a value

$m r$: a value

$f (m r)$: a function (a monadic value)

$f (m r) r$: a value

Reader \neq Reader
Type Data

https://wiki.haskell.org/All_About_Monads

https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

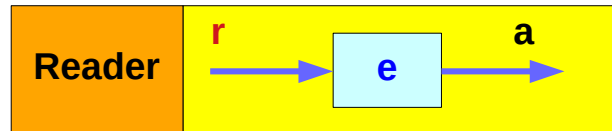
Implementing ReaderT

Reader Monad – implementation (II) – Reader e

```
newtype Reader r a = Reader { runReader :: r -> a }
```

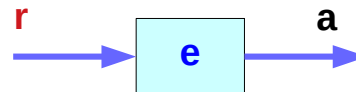
```
(Reader e) >>= f = Reader $ \r -> runReader (f (e r)) r
```

```
m = Reader e :: Reader r
```



```
runReader m :: r -> a
```

```
runReader m r :: a
```



```
e = runReader m :: r -> a
```

```
e r = runReader m r :: a
```

comparison with $(->)$ r monad

$(m \gg= f) r = f (m r) r$

m : a function (a monadic value)

r : a value

$m r$: a value

$f (m r)$: a function (a monadic value)

$f (m r) r$: a value

Reader \neq Reader
Type Data

https://wiki.haskell.org/All_About_Monads

https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

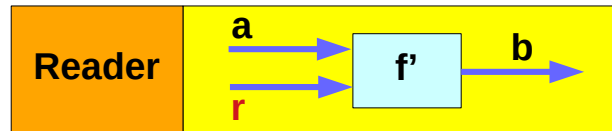
Implementing ReaderT

Reader Monad – implementation (II) – runReader k

```
newtype Reader r a = Reader { runReader :: r -> a }
```

```
(Reader e) >>= f = Reader $ \r -> runReader (f (e r)) r
```

```
m >>= f, f :: Reader r  
f' :: a -> r -> b
```



```
runReader m :: r -> a
```

```
runReader m r :: a
```

```
runReader f :: a -> r -> b
```

```
f' :: a -> r -> b
```

```
runReader (f (e r)) :: r -> b
```

```
runReader (f (e r)) r :: b
```

comparison with (->) r monad

$$(m \gg= f) r = f (m r) r$$

m : a function (a monadic value)

r : a value

$m r$: a value (the result)

$f (m r)$: a function (a monadic value)

$f (m r) r$: a value (the result)

```
m :: Reader r
```

```
runReader m r :: a
```

```
runReader (f (runReader m r)) r :: b
```

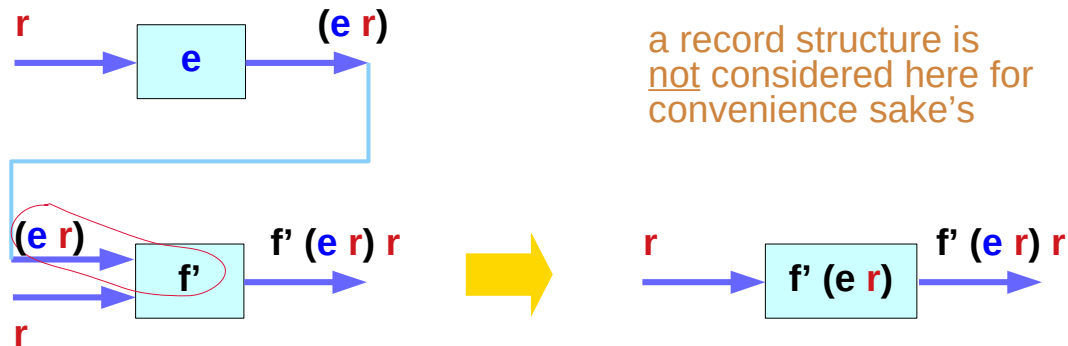
<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

Reader Monad – implementation (II) – >>=

```
newtype Reader r a = Reader { runReader :: r -> a }
```

```
(Reader e) >>= f = Reader $ \r -> runReader (f (e r)) r
```



comparison with (->) r monad

$(m \gg= f) r = f (m r) r$

m : a function (a monadic value)

r : a value

$m r$: a value

$f (m r)$: a function (a monadic value)

$f (m r) r$: a value

$(\gg=) :: (r \rightarrow a) \rightarrow (a \rightarrow r \rightarrow b) \rightarrow (r \rightarrow b)$

$m :: r \rightarrow a$

$f :: a \rightarrow r \rightarrow b$

$m \gg= f :: r \rightarrow b$

https://wiki.haskell.org/All_About_Monads

https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

Implementing ReaderT

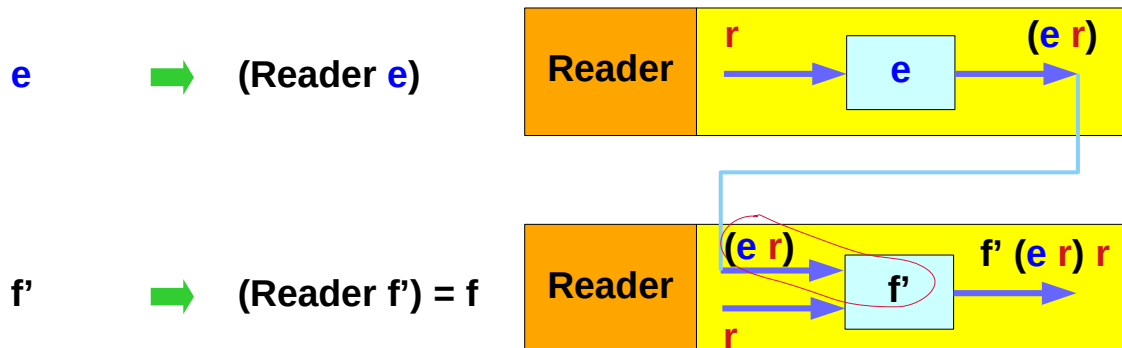
Reader Monad – implementation (II) – f' and f

instance Monad (Reader r) where

return a = Reader \$ \r -> a

(Reader e) >>= f = Reader \$ \r -> runReader (f (e r)) r

newtype Reader r a = Reader { runReader :: r -> a }



https://wiki.haskell.org/All_About_Monads

https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

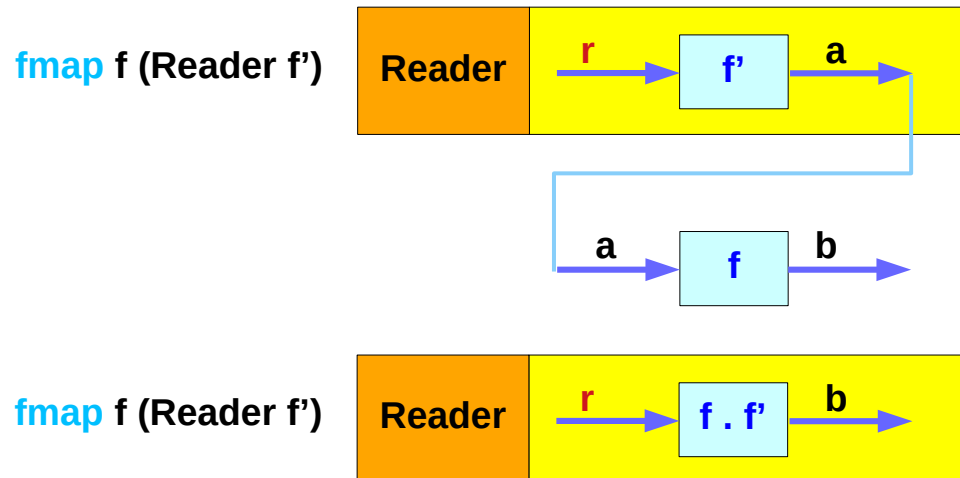
Implementing ReaderT

Reader Functor Implementation – fmap

instance Functor (Reader r) where

```
-- fmap :: (a -> b) -> Reader r a -> Reader r b
```

```
fmap f (Reader f') = Reader $ f . f'
```



https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

Implementing **ReaderT**

Reader Applicative Implementation – `<*>` type signature

```
instance Applicative (Reader r) where
```

```
-- pure :: a -> Reader r a
```

```
  pure a = Reader (const a)
```

```
-- (<*>) :: Reader r (a -> b) -> Reader r a -> Reader r b
```

```
  (Reader f) <*> (Reader g) = Reader $ \r -> f r (g r)
```

(Reader f)

Reader

$r \rightarrow (a \rightarrow b)$

(Reader g)

Reader

$r \rightarrow a$

(Reader f) `<*>` (Reader g)

Reader

$r \rightarrow b$

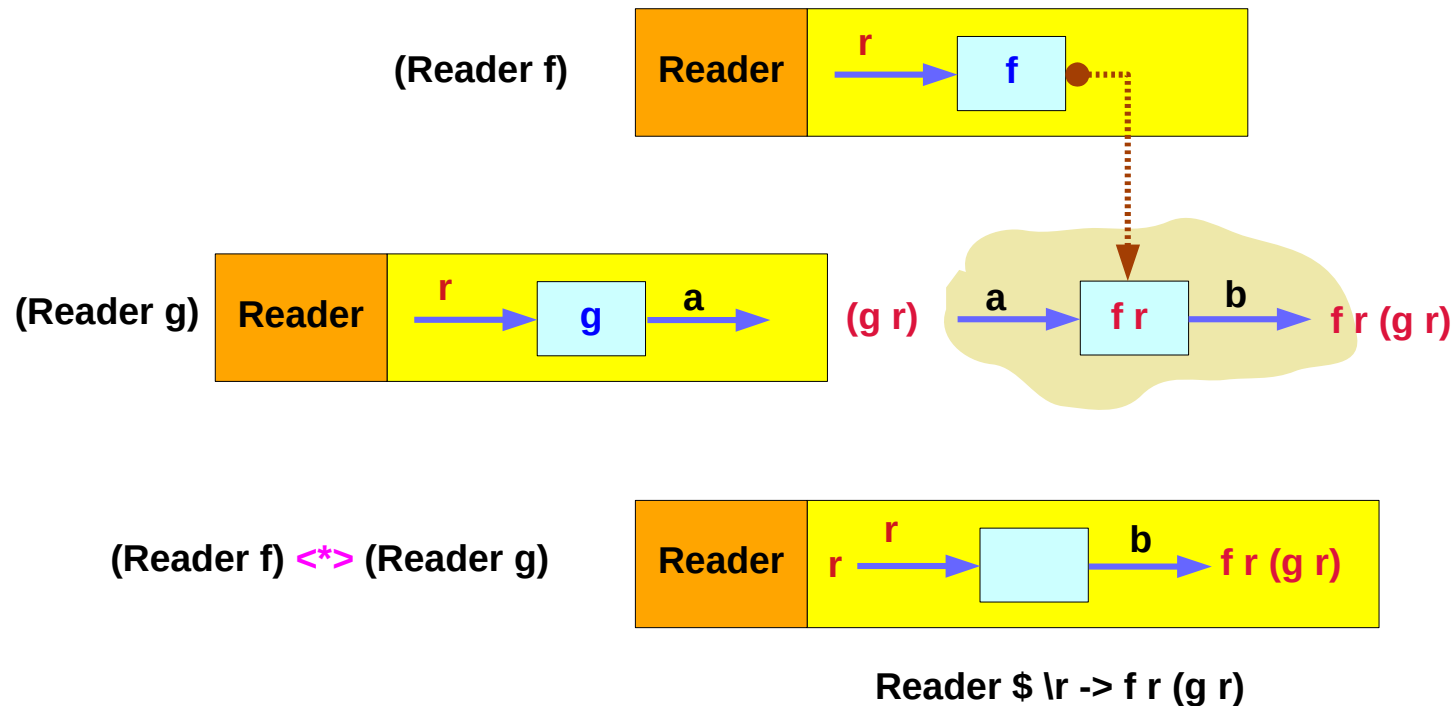
https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

Implementing **ReaderT**

Reader Applicative Implementation – $\langle * \rangle$

-- $\langle * \rangle :: \text{Reader } r \text{ (a -> b) -> Reader } r \text{ a -> Reader } r \text{ b}$

$(\text{Reader } f) \langle * \rangle (\text{Reader } g) = \text{Reader } \$ \backslash r \text{ -> } f \ r \ (g \ r)$



https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

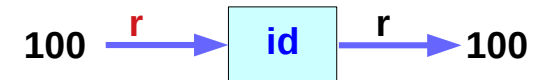
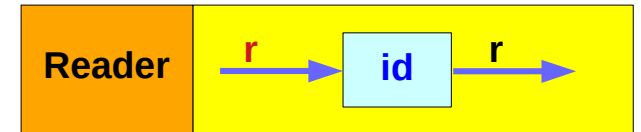
Implementing **ReaderT**

Reader Monad – examples eg1

```
eg1 :: Reader Int String
eg1 = Reader id >>= \e -> return $ "hey, " ++ show e
```

```
ghci> runReader eg1 100
"hey, 100"
```

Note that we use `id` to produce a **Reader** that just passes its **environment** argument along as the **output**. See **readerAsk** later for the equivalent situation which uses return for **MonadReader**.



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

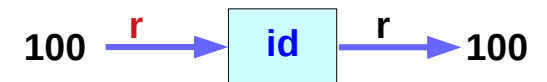
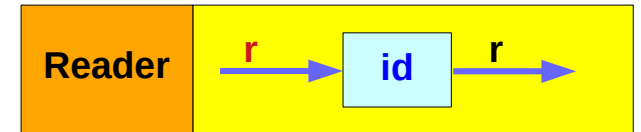
Implementing **ReaderT**

Reader Monad – examples eg1'

```
eg1' :: Reader Int String
eg1' = do e <- Reader id
         return $ "hey, " ++ show e
```

```
ghci> runReader eg1' 100
"hey, 100"
```

Note that we use `id` to produce a **Reader** that just passes its **environment** argument along as the **output**. See `readerAsk` later for the equivalent situation which uses return for **MonadReader**.



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

Reader Monad – examples eg1”

```
eg1" :: Reader Int String
```

```
eg1" = do e <- Reader (return :: Int -> [] Int)
```

```
return $ "hey, " ++ show e
```

```
ghci> runReader eg1" 100
```

```
"hey, [100]"
```

[100] : a monadic value

ambiguous type variable 'm0'
arising from a use of 'return'
Need to specify the type
signature

use a type annotation

```
return of Monad m  
return :: a -> m a  
return :: Int -> [] Int
```

[] is an instance of Monad
[100] : a monadic value

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

Reader Monad – src1

```
newtype Reader e a = Reader { runReader :: (e -> a) }

instance Monad (Reader r) where
  return a = Reader $ \_ -> a
  m >>= k = Reader $ \r -> runReader (k (runReader m r)) r

eg1 :: Reader Int String
eg1 = Reader id >>= \e -> return $ "hey, " ++ show e

eg1' :: Reader Int String
eg1' = do e <- Reader id
        return $ "hey, " ++ show e

eg1'' = do e <- Reader (return :: Int -> [] Int)
          return $ "hey, " ++ show e

instance Functor (Reader r) where
  fmap f (Reader f') = Reader $ f . f'

instance Applicative (Reader r) where
  pure a = Reader (const a)
  (Reader f) <*> (Reader f') = Reader $ \r -> f r (f' r)
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

Reader Monad – src2

```
newtype Reader e a = Reader { runReader :: (e -> a) }

instance Functor (Reader r) where
  fmap f (Reader f') = Reader $ f . f'

instance Applicative (Reader r) where
  pure a = Reader (const a)

  (Reader f) <*> (Reader f') = Reader $ \r -> f r (f' r)

instance Monad (Reader e) where
  return a      = Reader $ \e -> a
  (Reader r) >>= f = Reader $ \e -> runReader (f (r e)) e

eg1' :: Reader Int String
eg1' = do e <- Reader id
  return $ "hey, " ++ show e
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

ReaderT Monad Transformer

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

ReaderT Transformer

ReaderT is the **transformer** version of **Reader**.
can add the **first argument threading** capabilities of **Reader**
with another **Monad**.

ReaderT e IO.

One advantage of using **ReaderT** directly is
that there is a more expressive version of **local**

Unlike **local**, **withReaderT** can change
the type of the **environment** from **r** to **r'**.

1st argument - reader e
monad IO.

<https://hackernoon.com/the-reader-m Monad-part-1-1e4d947983a8>

Implementing **ReaderT**

ReaderT Transformer – typeclass definition

to use the **Reader** in conjunction with other monads,
create a transformer **ReaderT**:

for example, running IO actions and also
having access to the **reader's environment**.

```
newtype ReaderT r m a = ReaderT { runReaderT :: r -> m a }
```

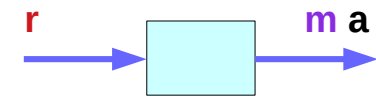
- **r** : the **reader environment**
- **m** : the **monad** (eg. **IO**)
- **a** : the **result** type of the **monad**

The type signature of the accessor function **runReaderT**:

```
runReaderT :: ReaderT r m a -> (r -> m a)
```

ReaderT r m a

r -> m a



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

ReaderT Transformer – runReaderT

```
newtype ReaderT r m a = ReaderT { runReaderT :: r -> m a }
```

```
runReaderT :: ReaderT r m a -> (r -> m a)
```

```
runReaderT (ReaderT f) = f
```

`runReaderT` takes a `ReaderT` type

and returns a **function**

that takes a **reader environment** of type `r`

and produces a **monadic value** `m a`

`ReaderT`

`(r -> m a)`

`runReaderT`

```
runReader' :: Reader r a -> (r -> a)
```

```
runReader' (Reader f) = f
```

`runReader`

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing `ReaderT`

ReaderT Transformer – ReaderT instance

```
instance (Monad m) => Monad (ReaderT r m) where
```

```
  return a = ReaderT $ \_ -> return a
```

```
  h >>= k = ReaderT $ \r -> do
```

```
    a <- runReaderT h r
```

```
    runReaderT (k a) r
```

```
  fail msg = ReaderT $ \_ -> fail msg
```

```
runReaderT :: ReaderT r m a -> (r -> m a)
```

```
runReaderT (ReaderT f) = f
```

```
instance Monad (Reader r) where
```

```
  return a = Reader $ \_ -> a
```

```
  m >>= k = Reader $ \r -> runReader (k (runReader m r)) r
```

ReaderT r m

ReaderT r m a

r -> m a

h :: ReaderT r m

Reader r

m :: Reader r

<https://downloads.haskell.org/~ghc/6.8.3/docs/html/libraries/mtl/src/Control-Monad-Reader.html>

Implementing ReaderT

ReaderT Transformer – ReaderT summary

```
newtype ReaderT r m a = ReaderT { runReaderT :: r -> m a }
```

```
runReaderT :: ReaderT r m a -> (r -> m a)
```

```
runReaderT (ReaderT f) = f
```

```
instance (Monad m) => Monad (ReaderT r m) where
```

```
  return a = ReaderT $ \_ -> return a
```

```
  h >>= k = ReaderT $ \r -> do
```

```
    a <- runReaderT h r
```

```
    runReaderT (k a) r
```

```
  fail msg = ReaderT $ \_ -> fail msg
```

ReaderT r m

f :: (r -> m a)

<https://downloads.haskell.org/~ghc/6.8.3/docs/html/libraries/mtl/src/Control-Monad-Reader.html>

Implementing ReaderT

ReaderT Transformer – >>= bind operator

```
h >>= k = ReaderT $ \r -> do
```

```
  a <- runReaderT h r
```

```
  runReaderT (k a) r
```

```
(>>=) :: ReaderT r m a -> (a -> ReaderT r m b) -> ReaderT r m b
```

```
h      :: ReaderT r m a
```

```
k      :: a -> ReaderT r m b
```

```
h >>= k  :: ReaderT r m b
```

ReaderT r m

<https://downloads.haskell.org/~ghc/6.8.3/docs/html/libraries/mtl/src/Control-Monad-Reader.html>

Implementing ReaderT

ReaderT Transformer – liftReaderT

```
liftReaderT :: m a -> ReaderT r m a
```

```
liftReaderT m = ReaderT (\_ -> m)
```

with a **ReaderT** as the **outer monad**,
it is desired to impart the **reader capability**
to a **monadic computation**

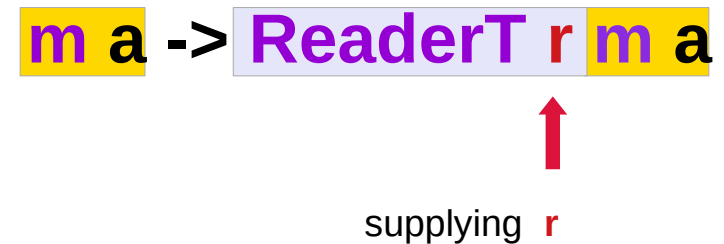
lift

when a **monadic action m** has nothing to do
with the **reader environment r**

In order to **lift** such a **monadic value m**,
liftReaderT does not take the **environment**:
but return a **ReaderT** monad value

Outer **ReaderT** monad
lift all monads to **ReaderT**

LiftReaderT does not use
environment r but return
ReaderT monad value



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

ReaderT Transformer – liftReaderT example

```
egLift :: ReaderT Int IO ()
```

```
egLift = do e <- ReaderT return ..... e :: Int <- ReaderT r m Int
          liftReaderT $ print "boo" ..... ReaderT r m ()
          liftReaderT $ print $ "value of e: " ++ show e ..... ReaderT r m ()
```

side effect

```
liftReaderT :: m a -> ReaderT r m a
```

```
liftReaderT m = ReaderT (\_ -> m) -- (1)
```

```
liftReaderT m = ReaderT (const m) -- (2)
```

```
print :: Show a => a -> IO ()
```

```
liftReaderT print "boo"
```

```
IO () -> ReaderT r IO ()
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

print

print takes a value of any type that's an **instance** of **Show** (meaning that we know how to represent it as a **string**), calls **show** with that value to stringify it and then outputs that **string** to the terminal.

It first runs **show** on a **value** and then **feeds** that to **putStrLn**, which **returns** an **I/O action** that will print out our value.

putStrLn . show.

main = do	\$ runhaskell print_test.hs
print True	True
print 2	2
print "haha"	"haha"
print 3.2	3.2
print [3,4,3]	[3,4,3]

<http://learnyouahaskell.com/input-and-output>

print vs. putStrLn

When we **type** out a value (like 3 or [1,2,3]) and press the **return** key, GHCi actually uses **print** on that value to display it on our terminal!

```
ghci> 3
3
ghci> print 3
3
ghci> map (++"!") ["hey","ho","woo"]
["hey!","ho!","woo!"]
ghci> print (map (++"!") ["hey","ho","woo"])
["hey!","ho!","woo!"]
```

to print out **strings**, use **putStrLn**
no **quotes**

to print out values of other types
use **print**

<http://learnyouahaskell.com/input-and-output>

show

- **Show a => a** : just run show and print it
- **Show a => IO a** : execute the action, run show and print
- **IO ()** : print nothing

```
ghci>15
15
ghci>'a' : 'b' : 'c' : []
"abc"
ghci>putStrLn "Hello, world!"
Hello, world!
ghci>putStrLn "Hello, world!" >> return 42
Hello, world!
42
ghci>
```

ghci has **three cases** for return values:

if you type an IO action, it get's executed and the result gets printed if it's not ().

<https://stackoverflow.com/questions/8332307/show-for-io-types>

ReaderT Transformer – examples

```
egLift :: ReaderT Int IO ()
egLift = do e <- ReaderT return
          liftReaderT $ print "boo"
          liftReaderT $ print $ "value of e: " ++ show e
```

```
eg1 :: Reader Int String
eg1 = Reader id >=> \e -> return $ "hey, " ++ show e
```

```
eg1' :: Reader Int String
eg1' = do e <- Reader id
        return $ "hey, " ++ show e
```

```
return :: a -> m a
return :: Int -> IO Int
```

```
ghci> runReaderT egLift 100
"boo"
"value of e: 100"
```

```
return :: a -> a
id :: a -> a
```

```
ghci> runReader eg1' 100
"hey, 100"
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

ReaderT Transformer – reading the environment

```
egLift :: ReaderT Int IO ()
egLift = do e <- ReaderT return
```

-- side effect

```
liftReaderT $ print "boo"
```

```
liftReaderT $ print $ "value of e: " ++ show e
```

```
ghci> runReaderT egLift 100
```

```
"boo"
```

```
"value of e: 100"
```

```
ReaderT r m a
ReaderT Int IO ()
```

```
ReaderT return
```

```
ReaderT $ r -> m r
```

```
ReaderT $ Int -> IO Int
```

```
runReaderT egLift 100
```

```
runReaderT ReaderT $ Int -> IO Int 100
```

```
(Int -> IO Int) 100
```

```
IO 100
```

```
e <- IO 100
```

side effect

```
e <- 100
```

only the result

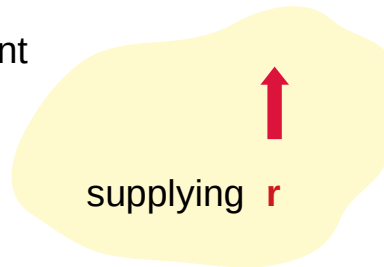
<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

ReaderT Transformer – return

```
instance (Monad m) => Monad (ReaderT r m) where
  return a = ReaderT $ \_ -> return a           -- (1)
  return   = liftReaderT . return                -- (2)
  return   = liftReaderT . pure                  -- (3)
```

The return function ignores the environment
Inject a value into the monadic type.



```
instance Monad (Reader r) where
  return a = Reader $ \_ -> a
  m >>= k = Reader $ \r -> runReader (k (runReader m r)) r
```

ReaderT r m

ReaderT r m a

r -> m a

Reader r

m :: Reader r

Class Constraints and Overloading

```
class Eq a where
```

```
(==)      :: a -> a -> Bool
```

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
```

```
instance (Eq a) => Eq (Tree a) where
```

```
Leaf a == Leaf b
```

```
= a == b
```

```
(Branch l1 r1) == (Branch l2 r2)
```

```
= (l1 == l2) && (r1 == r2)
```

```
  _ == _
```

```
= False
```

```
== in (Tree a)
```

```
== in a
```

<https://www.haskell.org/tutorial/classes.html>

Implementing ReaderT

ReaderT Transformer – return

```
instance (Monad m) => Monad (ReaderT r m) where
```

```
return a = ReaderT $ \_ -> return a
```

```
return = liftReaderT . return
```

```
return = liftReaderT . pure
```

```
return in  
ReaderT r m
```

```
return, pure in  
m
```

```
return ::  
a -> m a
```

```
return ::  
liftReaderT . return ::  
a -> ReaderT r m a
```

↑
supplying r

```
readerT $ \_ -> return a
```

↑
supplying r

<https://downloads.haskell.org/~ghc/6.8.3/docs/html/libraries/mtl/src/Control-Monad-Reader.html>

Implementing ReaderT

ReaderT Functor

```
mapReaderT :: (m a -> n b) -> ReaderT r m a -> ReaderT r n b
```

```
mapReaderT f m = ReaderT $ f . runReaderT m
```

```
instance (Functor m) => Functor (ReaderT r m) where
```

```
  fmap f = mapReaderT (fmap f)
```

```
instance Functor (Reader r) where
```

```
-- fmap :: (a -> b) -> Reader r a -> Reader r b
```

```
fmap f (Reader f') = Reader $ f . f'
```

<https://downloads.haskell.org/~ghc/6.8.3/docs/html/libraries/mtl/src/Control-Monad-Reader.html>

Implementing ReaderT

ReaderT Functor – mapReaderT

`mapReaderT :: (m a -> n b) -> ReaderT r m a -> ReaderT r n b`

`mapReaderT f m = ReaderT $ f . runReaderT m`

`mapReaderT f h = ReaderT $ f . runReaderT h`

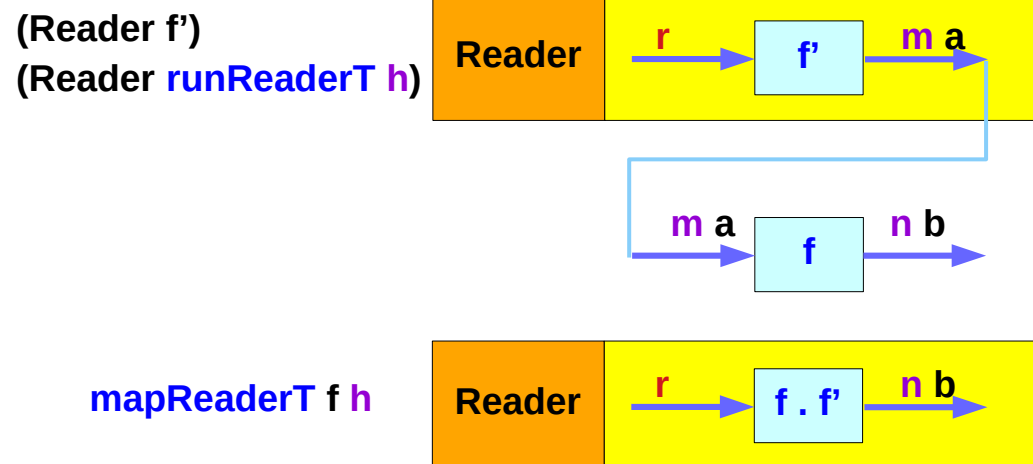
`newtype ReaderT r m a`
`= ReaderT { runReaderT :: r -> m a }`

`runReaderT ::`
`ReaderT r m a -> (r -> m a)`

`runReaderT (ReaderT f') = f'`

`m' :: ReaderT r m a`
`h :: ReaderT r m a`

`runReaderT m :: (r -> m a)`
`runReaderT h :: (r -> m a)`



ReaderT Applicative

```
instance (Applicative m) => Applicative (ReaderT r m) where
  pure   = liftReaderT . pure
  f <*> v = ReaderT $ \r -> runReaderT f r <*> runReaderT v r
```

```
instance Applicative (Reader r) where
  -- pure :: a -> Reader r a
  pure a = Reader (const a)
  -- (<*>) :: Reader r (a -> b) -> Reader r a -> Reader r b
  (Reader f) <*> (Reader f') = Reader $ \r -> f r (f' r)
```

<http://hackage.haskell.org/package/transformers-0.5.5.0/docs/src/Control.Monad.Trans.Reader.html#line-143>

Implementing ReaderT

ReaderT Applicative – pure

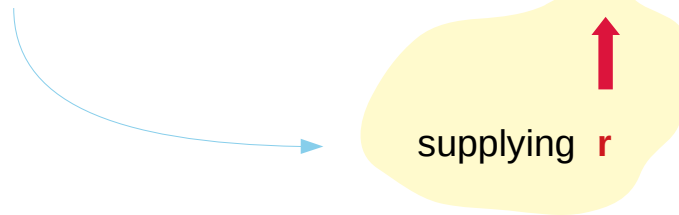
```
pure :: a -> ReaderT r m a
```

```
pure = liftReaderT . pure
```

```
pure :: a -> m a
```

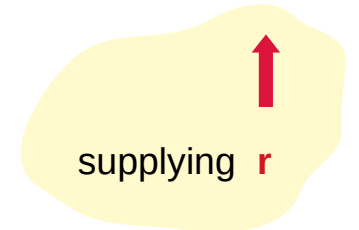
```
pure :: a -> ReaderT r m a
```

```
liftReaderT . return :: a -> ReaderT r m a
```



```
liftReaderT :: m a -> ReaderT r m a
```

```
liftReaderT m = ReaderT (\_ -> m)
```



<http://hackage.haskell.org/package/transformers-0.5.5.0/docs/src/Control.Monad.Trans.Reader.html#line-143>

Implementing ReaderT

ReaderT Applicative – <*>

```
(<*>) :: ReaderT r m (a -> b) -> ReaderT r m a -> ReaderT r m b  
f <*> v = ReaderT $ \r -> runReaderT f r <*> runReaderT v r
```

f = (ReaderT f')

Reader

r -> m (a -> b)

v = (ReaderT v')

Reader

r -> m a

(ReaderT f') <*> (ReaderT v')

Reader

r -> m b

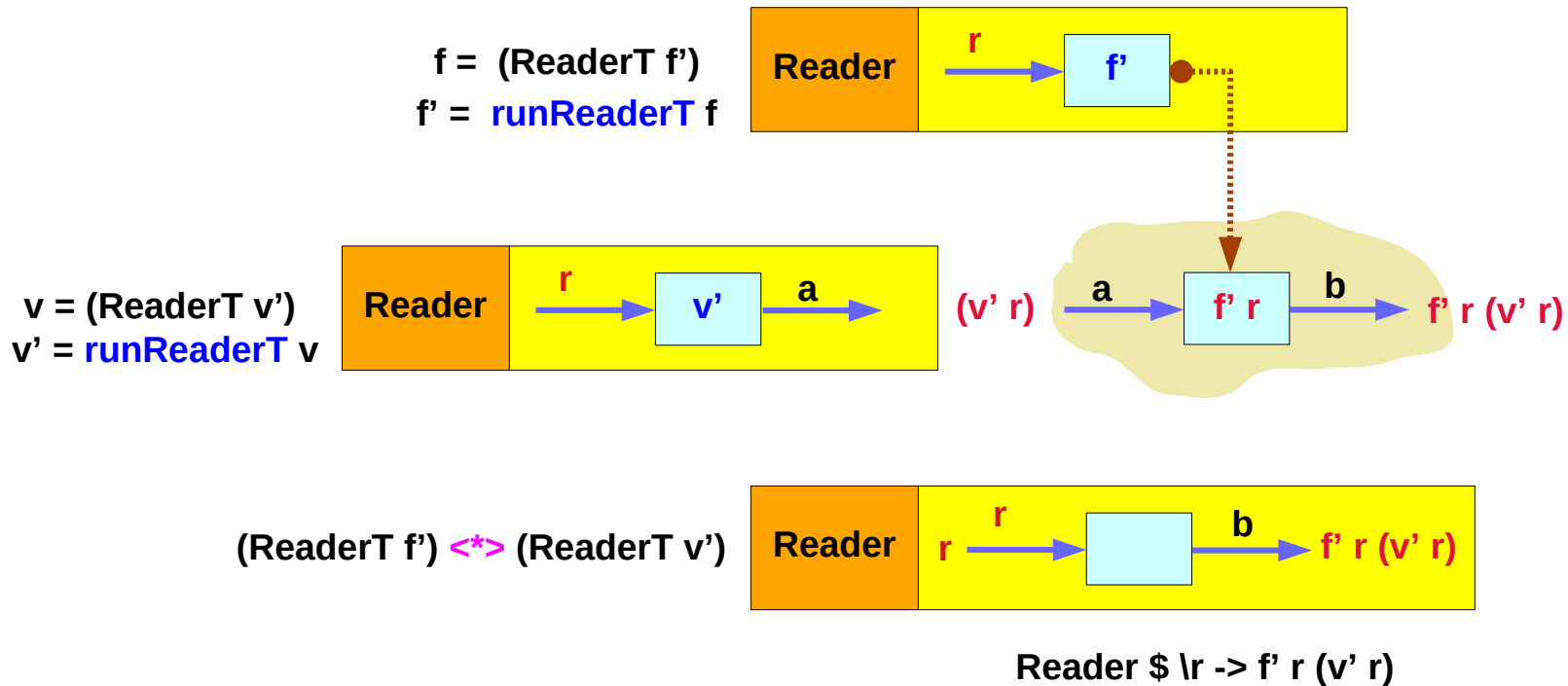
<http://hackage.haskell.org/package/transformers-0.5.5.0/docs/src/Control.Monad.Trans.Reader.html#line-143>

Implementing ReaderT

Reader Applicative Implementation – $\langle * \rangle$

$\langle * \rangle :: \text{ReaderT } r \ m \ (a \rightarrow b) \rightarrow \text{ReaderT } r \ m \ a \rightarrow \text{ReaderT } r \ m \ b$

$f \langle * \rangle v = \text{ReaderT } \$ \ |r \rightarrow \text{runReaderT } f \ r \langle * \rangle \text{runReaderT } v \ r$




https://www.reddit.com/r/haskellquestions/comments/6r46zp/applicative_instance_for_reader/

Implementing ReaderT

ReaderT Transformer – readerAsk

```
readerAsk :: (Monad m) => ReaderT r m r  
readerAsk = ReaderT return
```



ask function:

ReaderT r m r

return :: r -> m r



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

ReaderT Transformer – withReaderT

`withReaderT :: (r' -> r) -> ReaderT r m a -> ReaderT r' m a`

`withReaderT f rt = ReaderT $ (runReaderT rt) . f`

to modify the **environment**, use `withReaderT`
which takes a **function** as its first parameter
to modify the **environment**.

The type of the supplied reader `ReaderT r m a`

The type of the modified reader is `ReaderT r' m a`

so the **function** is of type `r' -> r` which modifies

`f :: r' -> r` *modifying function*

`rt :: ReaderT r m a` *supplied reader*

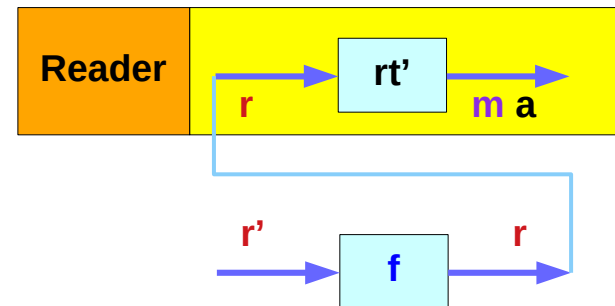
`(r' -> r)`

`(r into r')`

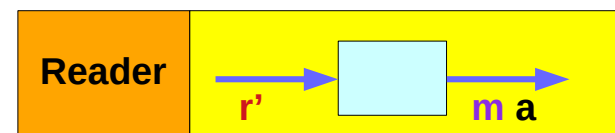
(original)

(modified)

`withReaderT`



`withReaderT`



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing `ReaderT`

ReaderT Transformer – readerReader, readerAsk

```
readerReader :: Monad m => (r -> a) -> ReaderT r m a
```

```
readerReader f = ReaderT $ \r -> return (f r)
```

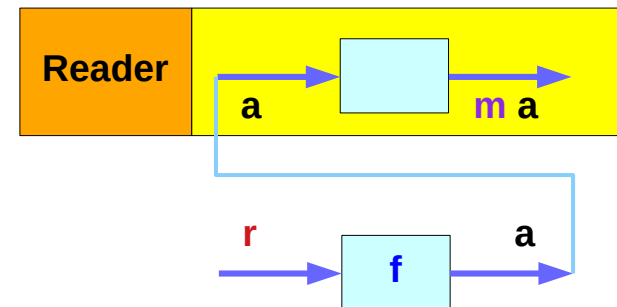
to apply a **function f** to the current environment. (f r)

This is almost the same as **readerAsk** except that we create a **reader** that returns **f r** instead of **f**.

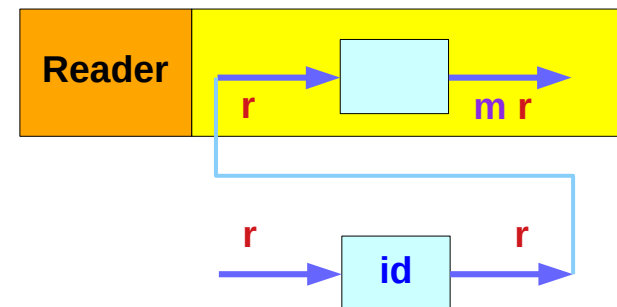
```
readerAsk' :: (Monad m) => ReaderT r m r
```

```
readerAsk' = readerReader id
```

readerReader



readerAsk'



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

ReaderT Transformer – src (1)

```
newtype ReaderT r m a = ReaderT { runReaderT :: r -> m a }

mapReaderT :: (m a -> n b) -> ReaderT r m a -> ReaderT r n b
mapReaderT f m = ReaderT $ f . runReaderT m

instance (Functor m) => Functor (ReaderT r m) where
  fmap f = mapReaderT (fmap f)

instance (Applicative m) => Applicative (ReaderT r m) where
  pure   = liftReaderT . pure
  f <*> v = ReaderT $ \ r -> runReaderT f r <*> runReaderT v r

instance (Monad m) => Monad (ReaderT r m) where
  return a = ReaderT $ \ _ -> return a
  m >=> k = ReaderT $ \ r -> do
    a <- runReaderT m r
    runReaderT (k a) r
  fail msg = ReaderT $ \ _ -> fail msg

liftReaderT :: m a -> ReaderT r m a
liftReaderT m = ReaderT (\_ -> m)
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

ReaderT Transformer – src (2)

```
egLift :: ReaderT Int IO ()
egLift = do e <- ReaderT return
          liftReaderT $ print "boo"
          liftReaderT $ print $ "value of e: " ++ show e

-- ghci> runReaderT egLift 100
-- "boo"
-- "value of e: 100"

readerAsk :: (Monad m) => ReaderT r m r
readerAsk = ReaderT return

withReaderT :: (r' -> r) -> ReaderT r m a -> ReaderT r' m a
withReaderT f rt = ReaderT $ (runReaderT rt) . f

readerReader :: Monad m => (r -> a) -> ReaderT r m a
readerReader f = ReaderT $ \r -> return (f r)

readerAsk' :: (Monad m) => ReaderT r m r
readerAsk' = readerReader id
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

MonadReader Class

ReaderT instance of MonadReader class


An instance declaration for our **ReaderT** type:

```
instance Monad m => MonadReader r (ReaderT r m) where
```

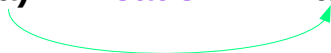
```
ask    = readerAsk    :: ReaderT r m r
```



```
local  = withReaderT  :: (r -> r) -> ReaderT r m a -> ReaderT r m a
```



```
reader = readerReader :: (r -> a) -> ReaderT r m a
```



<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

ReaderT Monad Example (1)

```
eg2 :: ReaderT Int IO String
eg2 = do

    e <- ask :: ReaderT Int IO Int

    liftReaderT $ print $ "in eg2 the environment is: " ++ (show e)

    return $ "returned value: " ++ show e
```

use the **IO** monad as the inner monad in a **ReaderT**, with an **Int environment (r)** and **String result (a)** type.

Get the reader environment.

Run an **IO** action; we have to use **liftReaderT** since we are currently

in the **ReaderT** monad, not the **IO** monad.

Final return value, a string.

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing **ReaderT**

ReaderT Monad Example (2)

```
eg2 :: ReaderT Int IO String
eg2 = do
  e <- ask :: ReaderT Int IO Int
  liftReaderT $ print $ "in eg2 the environment is: " ++ (show e)
  return $ "returned value: " ++ show e
```

```
ghci> result "in eg2 the environment is: 100"
```

```
ghci> result
```

```
"returned value: 100"
```

<https://carlo-hamalainen.net/2014/03/05/note-to-self-reader-monad-transformer/>
<https://github.com/carlohamalainen/playground/blob/master/haskell/transformers/MyOwnReaderT.lhs>

Implementing ReaderT

Auto-lifting in **mtl** MonadReader

Each **monad** in the **mtl** is defined in terms of a type class.

Reader is an instance of **MonadReader**,

ReaderT is also an instance of **MonadReader**

anything that wraps a **MonadReader** is
also set up to be a **MonadReader**

asks and **local** functions will work without any (manual) lifting.

Other **mtl monads** behave in a similar way.

https://wiki.haskell.org/Monad_Transformers_Explained

MonadReader Class Definition

```
class Monad m => MonadReader r m | m -> r where
```

See examples in `Control.Monad.Reader`.

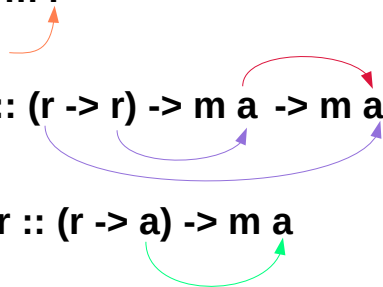
Note, the partially applied function type `(->) r` is a simple **reader** monad.

`(ask | reader), local`

`ask :: m r`

`local :: (r -> r) -> m a -> m a`

`reader :: (r -> a) -> m a`



<http://hackage.haskell.org/package/mtl-2.2.2/docs/Control-Monad-Reader.html>

MonadReader Class Methods

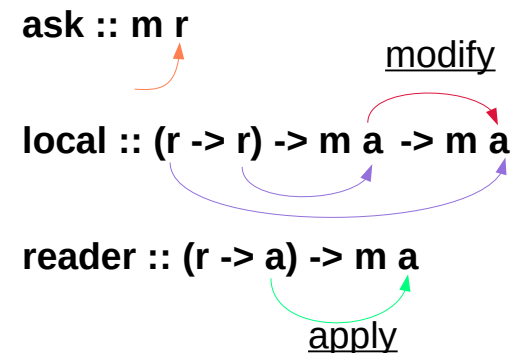
```
class Monad m => MonadReader r m | m -> r where
```

(ask | reader), local

```
ask :: m r      -- retrieves the monad environment.
```

```
local :: (r -> r) -- the select function to modify the environment.  
      -> m a      -- reader to run in the modified environment.  
      -> m a      -- executes a computation in a modified environment.
```

```
reader :: (r -> a) -- the selector function to apply to the environment.  
       -> m a      -- retrieves a function of the current environment.
```



<http://hackage.haskell.org/package/mtl-2.2.2/docs/Control-Monad-Reader.html>

References

- [1] <ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf>
- [2] <https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf>