

# Functor (1A)

---

Copyright (c) 2016 - 2017 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](mailto:youngwlim@hotmail.com).

This document was produced by using OpenOffice.

# Based on

---

<http://learnyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>

**Haskell in 5 steps**

[https://wiki.haskell.org/Haskell\\_in\\_5\\_steps](https://wiki.haskell.org/Haskell_in_5_steps)

# Maybe as a functor

---

**Functor** type class:

- transforming one type to another
- transforming operations of one type to those of another

**Maybe** has a useful instance of a **functor** type class

**Functor** provides **fmap** method

*maps functions* of the *base type* (such as *Integer*)  
to *functions* of the *lifted type* (such as *Maybe Integer*).

<https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell>

# Maybe as a functor

A *function* `f` transformed with `fmap`  
can work on a Maybe value

**case** maybeVal of

```
Nothing -> Nothing    -- there is nothing, so just return Nothing  
Just val -> Just (f val) -- there is a value, so apply the function to it
```

```
father :: Person -> Maybe Person  
mother :: Person -> Maybe Person
```

```
f :: Int -> Int  
fmap f :: Maybe Integer -> Maybe Integer
```

a **Maybe Integer** value: `m_x`

```
fmap f m_x
```

<https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell>

# Maybe as a functor

---

In fact, you could apply a whole chain of **lifted Integer -> Integer** functions to **Maybe Integer** values and only have to worry about explicitly checking for **Nothing** once when you're finished.

<https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell>

# Typeclasses

**Typeclasses** are like **interfaces**

defines some **behavior**  
comparing for equality  
comparing for ordering  
enumeration

**Instances** of that typeclass  
types possessing such behavior

Such behavior is defined by  
**function definition**  
**type declaration** to be implemented

a type is an instance of a typeclass implies  
the functions defined by the typeclass with that type can be used

No relation with classes in Java or Python

<http://leamyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>

# A Typeclass Example

## the Eq typeclass

defines the functions `==` and `/=`

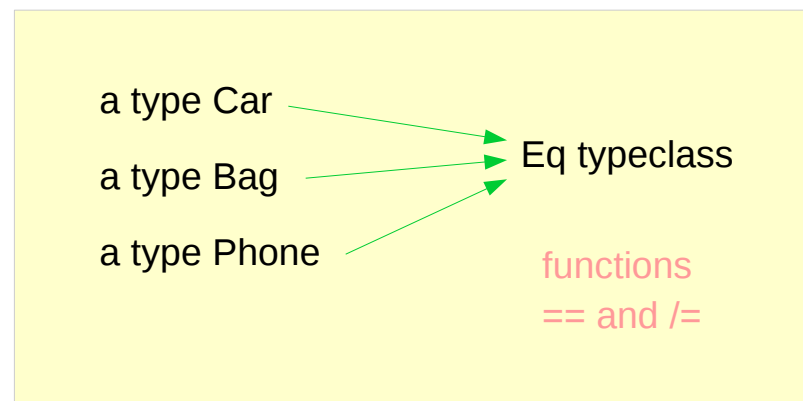
## a type Car

comparing two cars `c1` and `c2` with the equality function `==`

The Car type is an instance of Eq typeclass

Instances : various types

Typeclass : a group or a class of these similar types



<http://leamyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>



# Eq Typeclass Example

**class Eq** a **where**

<code>(==) :: a -&gt; a -&gt; Bool</code>	- a type declaration
<code>(/=) :: a -&gt; a -&gt; Bool</code>	- a type declaration
<code>x == y = not (x /= y)</code>	- a function definition
<code>x /= y = not (x == y)</code>	- a function definition

**data** TrafficLight = Red | Yellow | Green

**instance Eq** TrafficLight **where**

```
Red == Red = True
Green == Green = True
Yellow == Yellow = True
_ == _ = False
```

```
ghci> Red == Red
True
ghci> Red == Yellow
False
ghci> Red `elem` [Red, Yellow, Green]
True
```

<http://leamyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>

# Show Typeclass Example

```
class Show a where
  show :: a -> String      - a type declaration
  * * *
```

```
data TrafficLight = Red | Yellow | Green
```

```
instance Show TrafficLight where
  show Red = "Red light"
  show Yellow = "Yellow light"
  show Green = "Green light"
```

```
ghci> [Red, Yellow, Green]
[Red light, Yellow light, Green light]
```

<http://leamyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>

## References

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