клас Functor

class Functor f where

Клас Functor використовується для типів, які передбачають відображення (mapped over).

Екземпляри (втілення) функтора (Functor) повинні задовольняти наступним законам:

fmap id == id

fmap (g . h) == fmap g . fmap h

клас Functor

Minimal complete definition: fmap

fmap :: (a -> b) -> f a -> f b
метод 1 арг. 2 арг. рез.

1 арг.: функція a -> b
2 арг.: тип (a) в функторній обгортці (f) f a
результат: тип (b) в функторній обгортці (f) f b

Метод отримує функцію типу a -> b та значення типу a в функторній обгортці, повертає значення типу b в тій же функторній обгортці.

Функторна обгортка — конструктор типу (з одним параметром)

клас Functor

Список є екземпляром класу типів Functor

```
instance Functor [] where
   fmap = map

>fmap  (*2)  [1..3]
 [2,4,6]
>map  (*2)  [1..3]
 [2,4,6]
>fmap  (*2)  []
 [ ]
>map  (*2)  []
 [ ]
```

клас Functor

Maybe

```
instance Functor Maybe where
  fmap f  (Just x) =  Just (f x)
  fmap f  Nothing = Nothing

> fmap (*2) (Just 3)
Just 6
> fmap (*2) Nothing
Nothing
```

клас Functor

Maybe

```
instance Functor Maybe where
  fmap f  (Just x) =  Just (f x)
  fmap f  Nothing = Nothing

> fmap (++ "Використання J_U_S_T") (Just "Ось
приклад ")
Just "Ось приклад Використання J_U_S_T"

>map (++ "Використання J_U_S_T") Nothing
Nothing
```

клас Functor

Дерево

```
instance Functor Tree where
  fmap f  EmptyTree =  EmptyTree
  fmap f  (Node x left right)
=
  Node (f x) (fmap f left) (fmap f right)
```

клас Functor

Дерево

```
instance Functor Tree where
  fmap f  EmptyTree =  EmptyTree
  fmap f  (Node x left right)
=
  Node (f x) (fmap f left) (fmap f right)

> fmap (*2)   EmptyTree
EmptyTree
> fmap (*4)   (foldr treeInsert EmptyTree
[5,7,3])
Node 20   (Node 12  EmptyTree EmptyTree)
 (Node 28  EmptyTree EmptyTree)
```

клас Functor

Either

```
instance Functor (Either a) where
    fmap _ (Left x) = Left x
    fmap f (Right y) = Right (f y)
```

клас Functor

Either

```
instance Functor (Either a) where
    fmap _ (Left x) = Left x
    fmap f (Right y) = Right (f y)

> let s = Left "foo" :: Either String Int
> let n = Right 3 :: Either String Int

> fmap (*2) s
Left "foo"
> fmap (*2) n
Right 6
```

клас Applicative

```
class Functor f => Applicative f where

   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

<*> - оператор послід-го застосування (sequential application)

f – екземпляр аплікативного функтора.
pure отримує значення типу **a** та повертає у обгортці аплікативного функтора **f a** (вміщує у мінімальний / чистий / за-замовчанням контекст)

клас Applicative

інші методи

```
(*>) :: f a -> f b -> f b
(<*) :: f a -> f b -> f a
```

клас Applicative

```
Порівняємо з сигнатурою fmap
(<*>) :: f (a -> b) -> f a -> f b
 fmap ::   (a -> b) -> f a -> f b

fmap працює з фукцією і функтором
<*>  - працює з двома функторами
```

клас Applicative

```
(<*>) :: f (a -> b) -> f a ->  f b
         -----------   -----   ----
         1-й функтор   2-й ф-р рез-т
            містить    містить  містить
            функцію    зн.т. a зн.т. b
            типу (a -> b)
```

<*> витягує функцію з першого функтора та застосовує до другого

клас Applicative

```
  Maybe
instance  Applicative  Maybe where
  pure = Just
  Nothing <*> _ = Nothing
--(бо Nothing не містить функції)
  Just f <*> somesing = fmap f somesing
-- (витягує з Just f ф-ю та застосовує її)
```

клас Applicative

```
Prelude> Just (+2) <*> Just 10
Just 12
Prelude> Just (+2) <*> pure 10
Just 12
Prelude> pure (+2) <*> Just 10
Just 12
Prelude> Just (+2) <*> Nothing
Nothing
Prelude> Nothing <*> Just 10
Nothing
```

клас Applicative

```
Prelude> Just ("abcd"++) <*> Just "efgh"
Just "abcdefgh"

Prelude> Just ("abcd"++) <*> Nothing
Nothing
```

клас Applicative

Списки

```
instance Applicative [] where
    pure x    = [x]
    fs <*> xs = [f x | f <- fs, x <- xs]
-- ліворуч – список ф-ій, праворуч –
список значень
аплікативна обгортка - [ ]
```

клас Applicative

```
Prelude> [(*0),(+100),(^2)] <*> [1,2,3]
[0,0,0,101,102,103,1,4,9]

Prelude> [(*),(+),(^)] <*> [1,2,3] <*>
[4,5,6]
[4,5,6,8,10,12,12,15,18,5,6,7,6,7,8,7,8,9,
1,1,1,16,32,64,81,243,729]

Prelude> [(*),(+),(^)] <*> [1,2,3] <*>
[4,5]
[4,5,8,10,12,15,5,6,6,7,7,8,1,1,16,32,81,2
43]
```