

The Lens library



The background of the slide is a close-up photograph of a stone wall. The stones are irregular in shape and size, with a mix of light beige, tan, and reddish-brown hues. The texture is rough and natural.

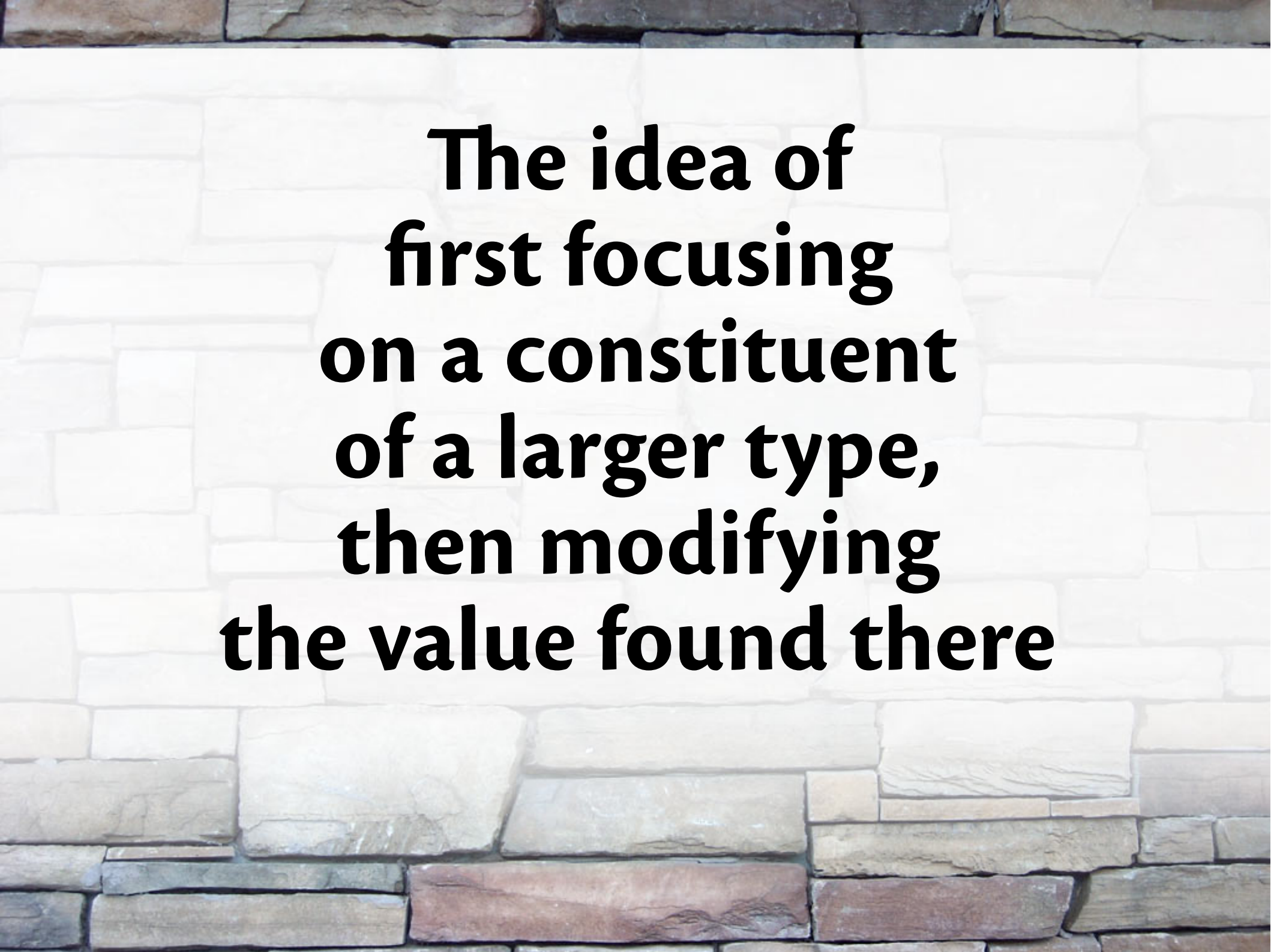
**Lenses
are a powerful way
of using functions
on complex data types**

The background of the image is a close-up of a stone wall. The stones are of various sizes and shapes, with colors ranging from light beige to dark brown. A large, white rectangular area is superimposed over the center of the wall, serving as a background for the text.

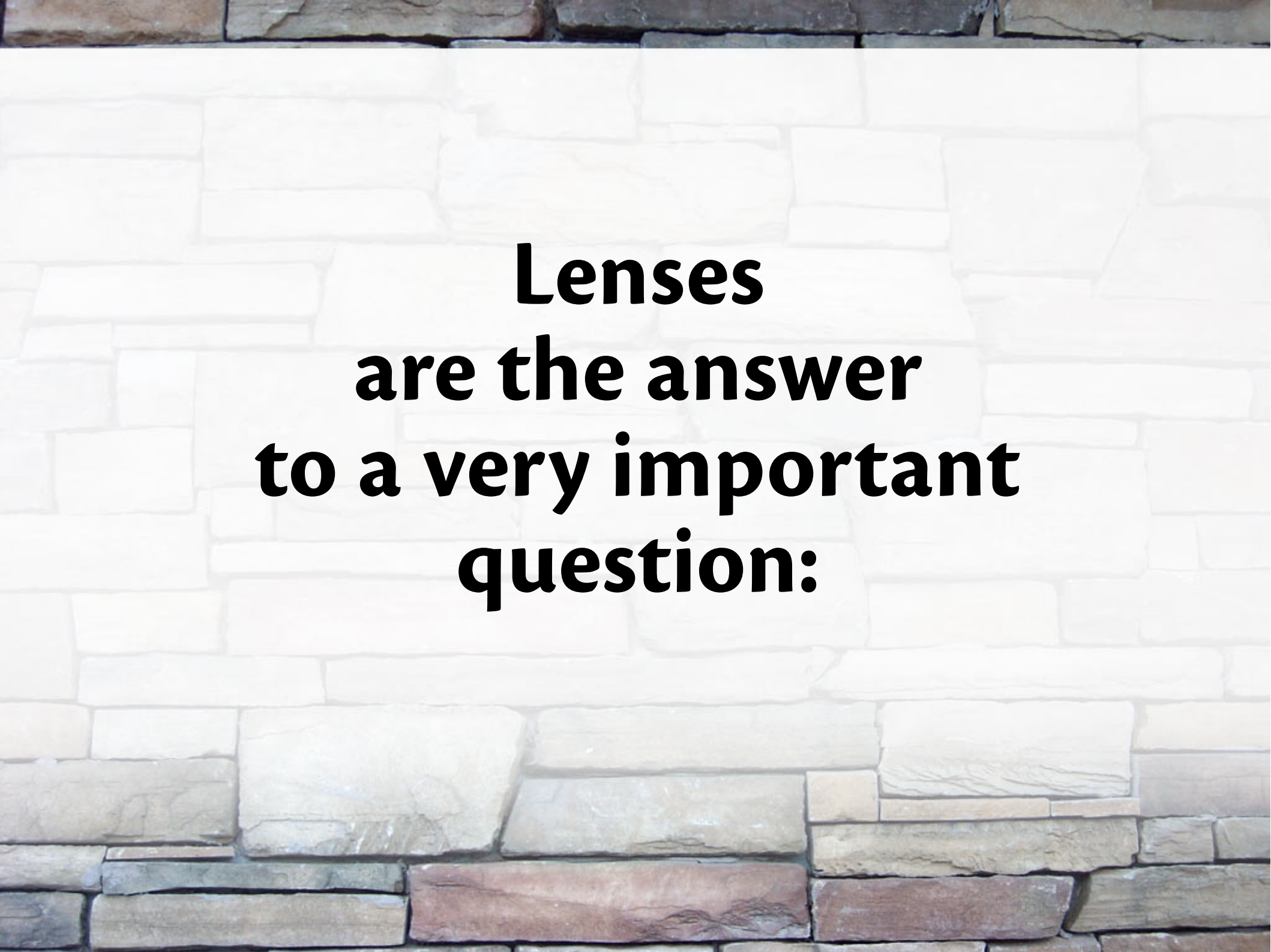
**A lens provides
a relationship between
a complex, nested type
and one of its constituents**

A background image of a stone wall. The top portion of the wall is composed of light-colored, rectangular stones. The bottom portion features a mix of darker, more irregularly shaped stones in shades of brown and grey.

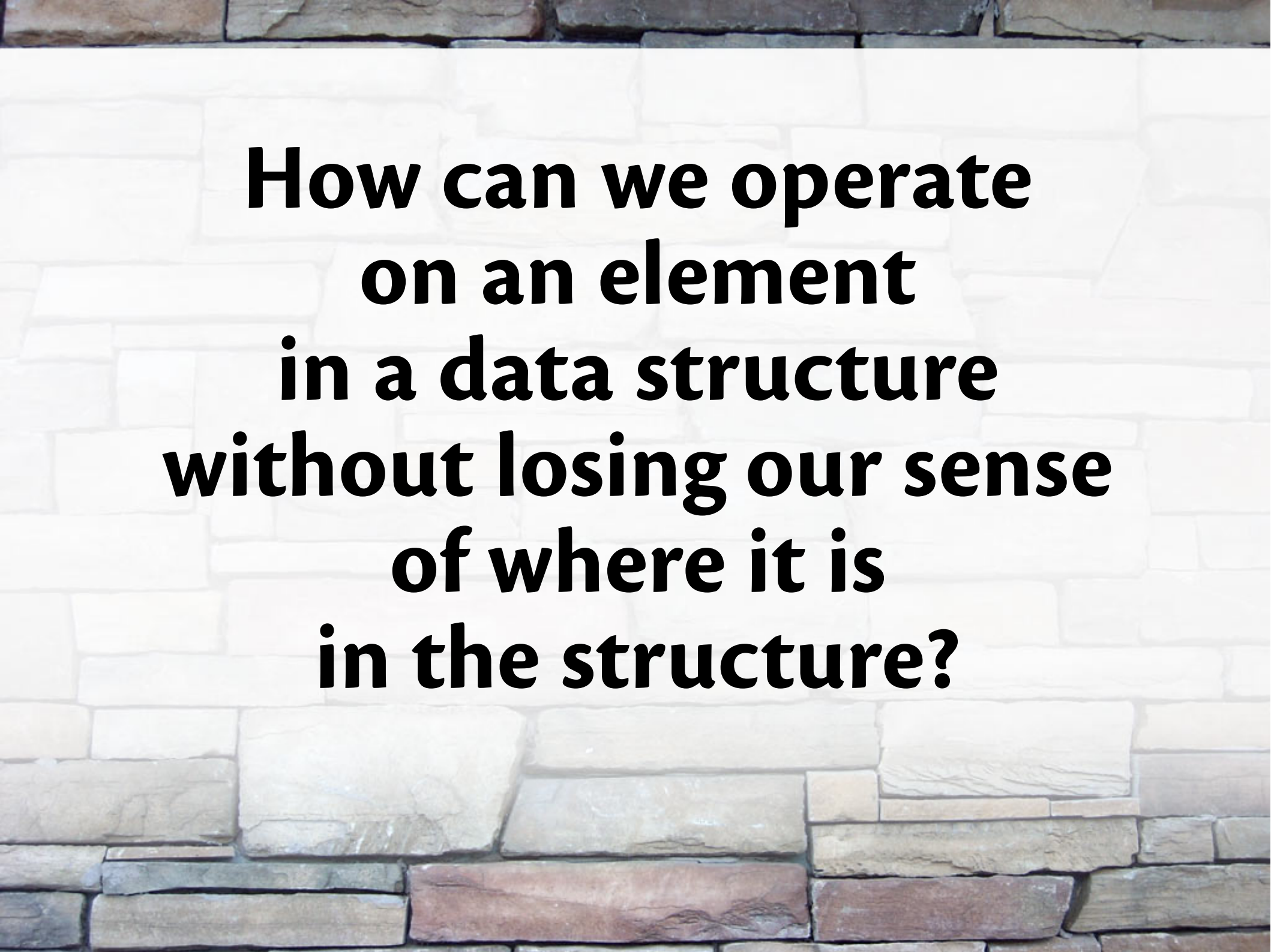
**Lenses provide
a useful new abstraction:**

The background of the image is a close-up of a stone wall. The stones are of various sizes and colors, including shades of tan, beige, and reddish-brown. The wall is composed of several horizontal courses of stones. In the center of the image, there is a large, white rectangular area that serves as a backdrop for the text.

**The idea of
first focusing
on a constituent
of a larger type,
then modifying
the value found there**



**Lenses
are the answer
to a very important
question:**



**How can we operate
on an element
in a data structure
without losing our sense
of where it is
in the structure?**

**In the first example,
lenses help by providing
a uniform interface
when retrieving
the second element
from an n -sized tuple,
where n is arbitrary**

Without lenses:

fst	(a, _)	= a
snd	(_, b)	= b
fst0f3	(a, _, _)	= a
snd0f3	(_, b, _)	= b
third0f3	(_, _, c)	= c
fst0f4	(a, _, _, _)	= a
snd0f4	(_, b, _, _)	= b
third0f4	(_, _, c, _)	= c
fourth0f4	(_, _, _, d)	= d

With lenses:

```
λ> over (_2) (+1) (7,77)  
(7,78)
```

```
λ> over (_2) (+1) (7,77,777)  
(7,78,777)
```

```
λ> over (_2) (+1) (7,77,777,7777)  
(7,78,777,7777)
```

```
λ> over (_2) (+1) (7,77,777,7777,77777)  
(7,78,777,7777,77777)
```

**Before we move on
to more complex types,
let's import the stuff we need:**

```
import           Control.Lens  
import qualified Data.Map       as M  
import qualified Data.Set       as S
```


The background of the slide is a close-up photograph of a stone wall. The wall is composed of irregularly shaped stones in various shades of beige, tan, and light brown. The stones are laid in a pattern that is roughly horizontal but with many vertical joints. The lighting is even, highlighting the textures of the stone surfaces.

Question:

**What do
the following expressions
have in common?**

```
[ (1, (11, 111, 1111))  
, (2, (22, 222, 2222))  
]
```

```
[ (1, (11, 111, 1111, 11111))  
, (2, (22, 222, 2222, 22222))  
]
```

```
M.fromList [(1, (11, 111)), (2, (22, 222))]
```

```
(1, 2, 3, ((9, 8, 7, "bingo"), 'a', 'b'), 4, 5, 6)
```

```
('a', M.fromList [('a', 1), ('b', 2), ('c', 3)])
```

```
initialGameState = [ [2,0,0,2]  
                      , [0,0,0,0]  
                      , [0,0,0,0]  
                      , [2,0,0,2]  
                      ]
```



```
applicationState =
  ( 'a'
  , 'b'
  , ( M.fromList $ zip [1,2,3] [11,22,33]
    , M.fromList $ zip [4,5,6] [44,55,66]
    , M.fromList $ zip [7,8,9] [ S.fromList ["77","777","7777"]
                                , S.fromList ["88","888","8888"]
                                , S.fromList ["99","999","9999"]
                                ]
    )
  )
```

A background image of a stone wall. The top portion of the wall is composed of light-colored, rectangular stones. The bottom portion features a mix of darker, reddish-brown and grey stones, some of which are rectangular while others are more irregular and layered.

Answer:

**They are all nested
(to a lesser or greater degree)**

```
λ> over ( FILL_IN ) (+5) $ M.fromList [ (1,(11,111))  
                                          , (2,(22,222))  
                                          ]  
fromList [(1,(16,116)),(2,(22,222))]
```



```
λ> over (at 1 . non (88,888) . both) (+5) $ M.fromList
      [ (1,(11,111))
        , (2,(22,222))
        ]
fromList [(1,(16,116)),(2,(22,222))]
```

```
λ> over ( FILL_IN ) (+5) $ M.fromList [ (1,(11,111))  
                                           , (2,(22,222))  
                                           ]  
fromList [(1,(11,111)),(2,(22,222)),(5,(93,893))]
```

```
λ> over (at 5 . non (88,888) . both) (+5) $ M.fromList
      [ (1,(11,111))
        , (2,(22,222))
        ]
fromList [(1,(11,111)),(2,(22,222)),(5,(93,893))]
```



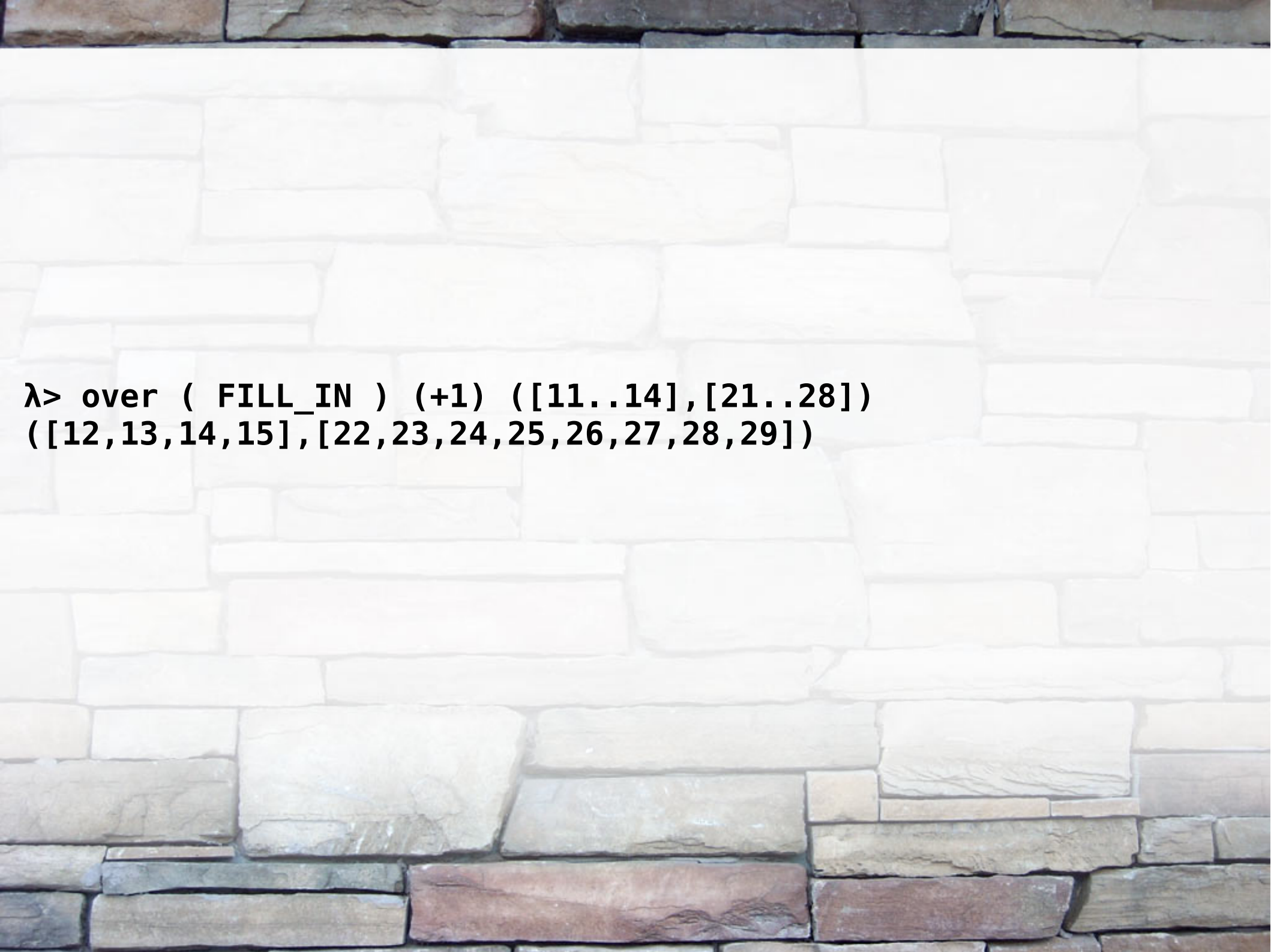
```
λ> set ( FILL_IN ) (Just (55,555)) $ M.fromList [ (1,(11,111))  
                                                    , (2,(22,222))  
                                                    ]  
fromList [(1,(11,111)), (2,(22,222)), (5,(55,555))]
```

```
λ> set (at 5) (Just (55,555)) $ M.fromList [ (1,(11,111))  
                                              , (2,(22,222))  
                                              ]  
fromList [(1,(11,111)),(2,(22,222)),(5,(55,555))]
```

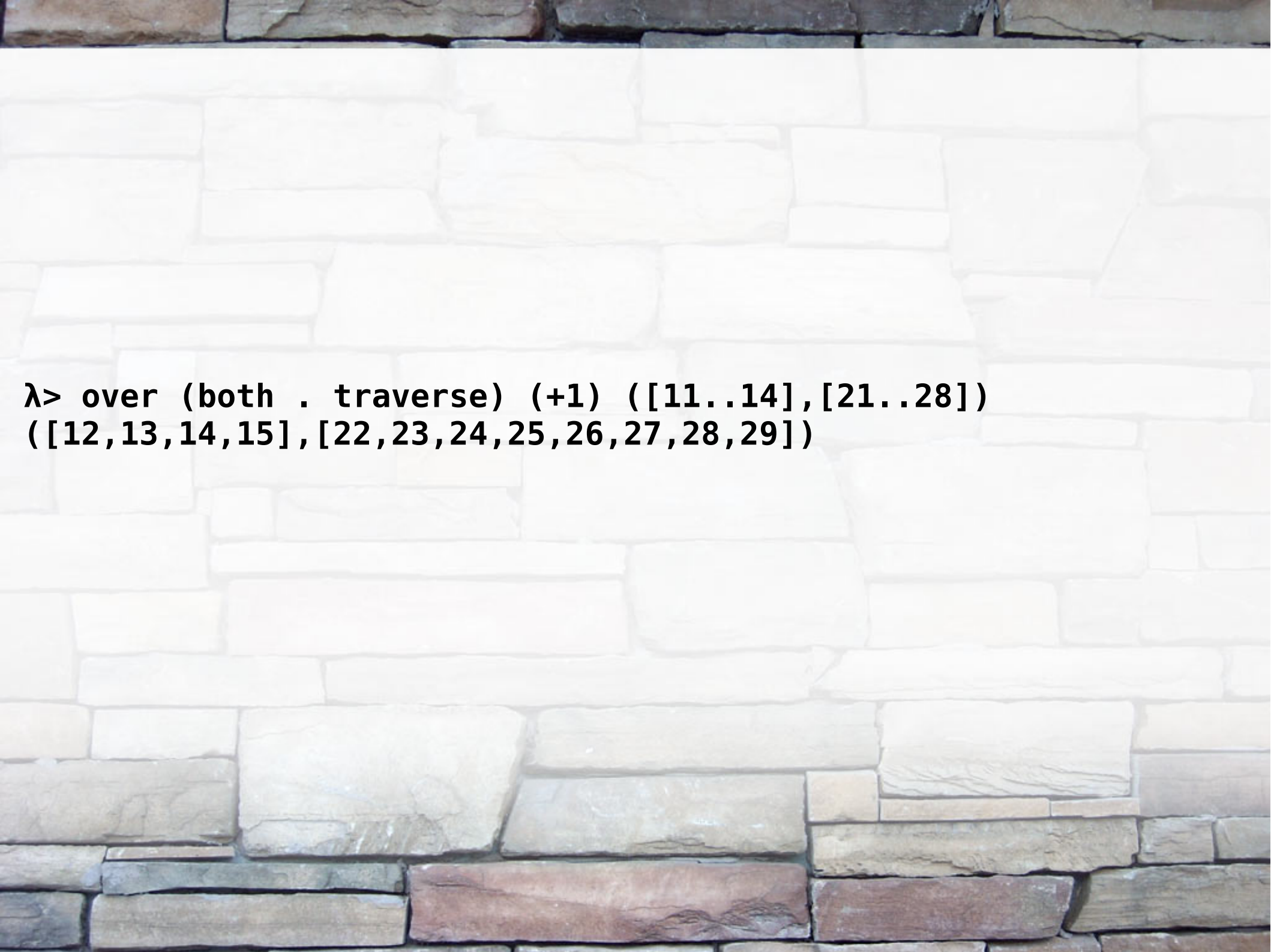

[illegible]

```
λ> over ( FILL_IN ) (+1) [ (1,(11,111,1111))  
                           , (2,(22,222,2222))  
                           ]  
[(1,(12,112,1112)),(2,(23,223,2223))]
```

```
λ> over (traverse . _2 . each) (+1) [ (1,(11,111,1111))  
                                         , (2,(22,222,2222))  
                                         ]  
[(1,(12,112,1112)), (2,(23,223,2223))]
```





```
λ> over ( FILL_IN ) (+1) ([11..14],[21..28])  
([12,13,14,15],[22,23,24,25,26,27,28,29])
```



**$\lambda >$ over (both . traverse) (+1) ([11..14],[21..28])
([12,13,14,15],[22,23,24,25,26,27,28,29])**

```
λ> over ( FILL_IN ) (+1) initialState  
[[3,1,1,3],[1,1,1,1],[1,1,1,1],[3,1,1,3]]
```

```
λ> over (traverse . traverse) (+1) initialState  
[[3,1,1,3],[1,1,1,1],[1,1,1,1],[3,1,1,3]]
```

```
λ> over ( FILL_IN ) (+1) initialState  
[[2,0,0,2],[0,0,0,0],[0,0,0,0],[3,1,1,3]]
```

```
λ> over (ix 3 . traverse) (+1) initialGameState  
[[2,0,0,2],[0,0,0,0],[0,0,0,0],[3,1,1,3]]
```



```
λ> over ( FILL_IN ) (+1) initialState  
[[2,0,0,2],[0,0,0,0],[0,0,0,0],[2,1,0,2]]
```

```
λ> over (ix 3 . ix 1) (+1) initialState  
[[2,0,0,2],[0,0,0,0],[0,0,0,0],[2,1,0,2]]
```

```
λ> over ( FILL_IN ) (+3) applicationState
( 'a'
, 'b'
, ( fromList [ (1,11), (2,22), (3,36) ]
, fromList [ (4,44), (5,55), (6,66) ]
, fromList [ (7, fromList [ "77", "777", "7777" ])
, (8, fromList [ "88", "888", "8888" ])
, (9, fromList [ "99", "999", "9999" ])
]
)
)
```



```
λ> over (_3 . _1 . at 3 . non 8) (+3) applicationState
( 'a'
, 'b'
, ( fromList [ (1,11), (2,22), (3,36) ]
, fromList [ (4,44), (5,55), (6,66) ]
, fromList [ (7, fromList [ "77", "777", "7777" ])
, (8, fromList [ "88", "888", "8888" ])
, (9, fromList [ "99", "999", "9999" ])
]
)
)
```

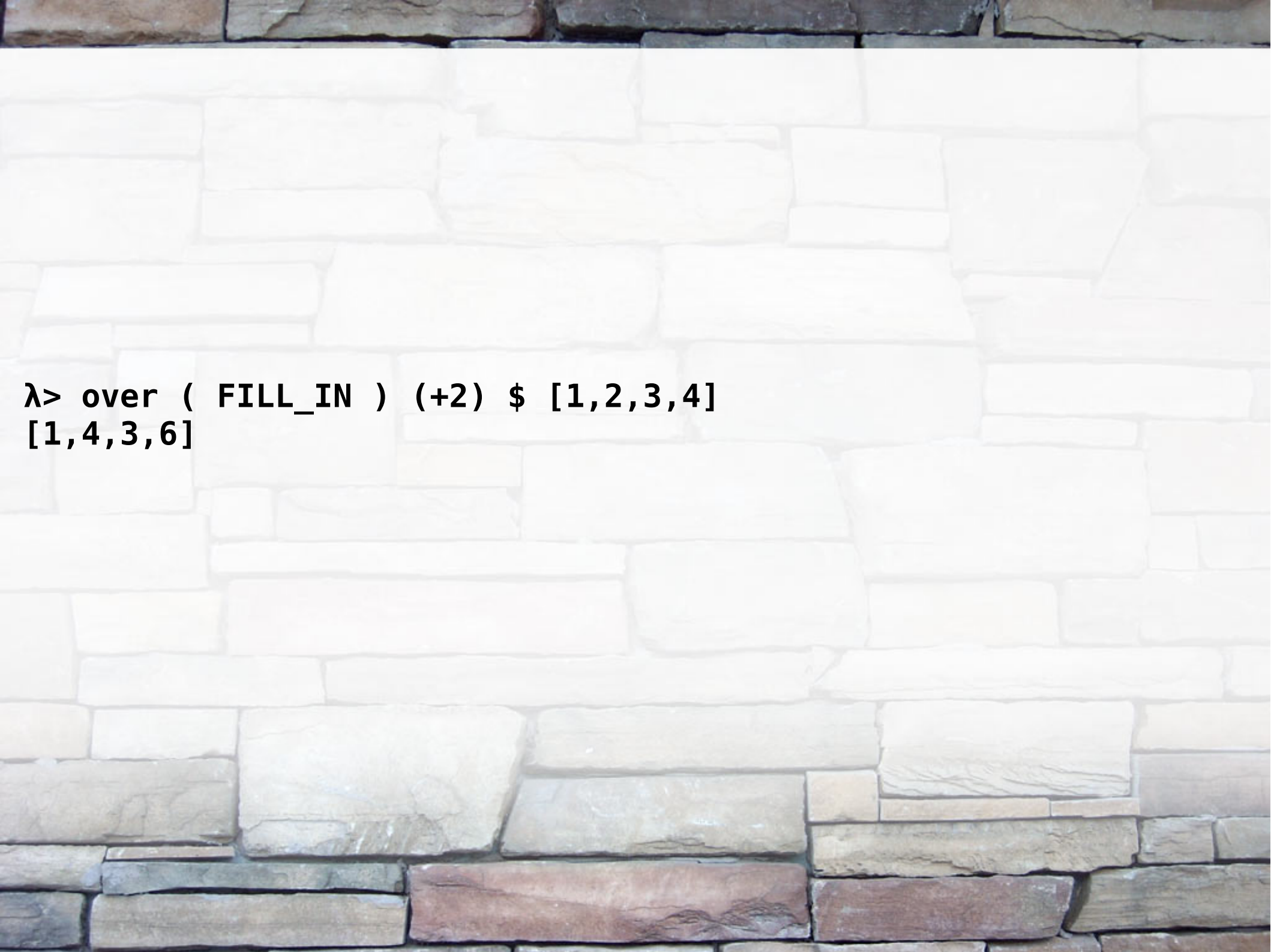
```
λ> over ( FILL_IN ) (+3) applicationState
( 'a'
, 'b'
, ( fromList [ (1,11), (2,22), (3,33), (4,11) ]
, fromList [ (4,44), (5,55), (6,66) ]
, fromList [ (7, fromList [ "77", "777", "7777" ])
, (8, fromList [ "88", "888", "8888" ])
, (9, fromList [ "99", "999", "9999" ])
]
)
)
```

```
λ> over (_3 . _1 . at 4 . non 8) (+3) applicationState
( 'a'
, 'b'
, ( fromList [ (1,11), (2,22), (3,33), (4,11) ]
, fromList [ (4,44), (5,55), (6,66) ]
, fromList [ (7, fromList [ "77", "777", "7777" ])
, (8, fromList [ "88", "888", "8888" ])
, (9, fromList [ "99", "999", "9999" ])
]
)
)
```



```
λ> set ( FILL_IN )  
      False  
      applicationState  
( 'a'  
  , 'b'  
  , ( fromList [ (1,11), (2,22), (3,33) ]  
      , fromList [ (4,44), (5,55), (6,66) ]  
      , fromList [ (7, fromList [ "77", "777", "7777" ] )  
                  , (8, fromList [ "88", "888", "8888" ] )  
                  , (9, fromList [ "99", "999" ] )  
      ]  
  )  
)
```

```
λ> set (_3 . _3 . at 9 . non S.empty . contains "9999")
      False
      applicationState
( 'a'
, 'b'
, ( fromList [ (1,11), (2,22), (3,33) ]
  , fromList [ (4,44), (5,55), (6,66) ]
  , fromList [ (7, fromList [ "77", "777", "7777" ])
    , (8, fromList [ "88", "888", "8888" ])
    , (9, fromList [ "99", "999" ])
  ]
)
)
```



```
λ> over ( FILL_IN ) (+2) $ [1,2,3,4]  
[1,4,3,6]
```




```
λ> over (traverse . filtered even) (+2) $ [1,2,3,4]  
[1,4,3,6]
```



```
λ> over ( FILL_IN )  
      toUpper  
      "hello lens world"  
"HeLlO LeNS WoRLD"
```

```
λ> over ( traverse . filtered (`notElem` "aeiou") )  
      toUpper  
      "hello lens world"  
"HeLlO LeNS WoRLD"
```




```
λ> over ( FILL_IN ) toUpper "abcdefghijkeeab"  
"abcdefghijkeeāB"
```



```
λ> over (_last) toUpper "abcdefghijkeeab"  
"abcdefghijkeeab"
```



What about records?


```
{-# LANGUAGE TemplateHaskell #-}
```

```
import Control.Lens
```

```
data XYPosition = MkXYPosition { _xPos :: Int  
                                , _yPos :: Int  
                                } deriving (Show)
```

```
data Ghost = MkGhost { _isDangerous    :: Bool  
                      , _ghostLocation :: XYPosition  
                      } deriving (Show)
```

```
makeLenses ''XYPosition
```

```
makeLenses ''Ghost
```

```
λ> let p = MkXYPosition 5 6
λ>
λ> p
MkXYPosition {_xPos = 5, _yPos = 6}
λ>
λ> over (xPos) (+10) p
MkXYPosition {_xPos = 15, _yPos = 6}
```




Follow me on GitHub

github.com/dserban