Discussion Section: Folds

2021/05/07

Higher-Order Functions

- map :: (a -> b) -> [a] -> [b]
- filter :: (a -> Bool) -> [a] -> [a]

Quiz

What's the result of `map (\x -> x `mod` 10) [1,2,100,85]`?

A. `[1,2,100,85]`
B. `[0,0,10,8]`
C. `[1,2,0,5]`
D. `[5,0,2,1]`
E. None of the above

Quiz

What's the result of `map (\x -> x `mod` 10) [1,2,100,85]`?

A. `[1,2,100,85]`
B. `[0,0,10,8]`
C. **`[1,2,0,5]`**
D. `[5,0,2,1]`
E. None of the above

Quiz

What's the result of `filter (not . even) [1,2,3,4,5,6]`?

A. `[1,2,3,4,5,6]`
B. `[2,4,6]`
C. `[1,3,5]`
D. `[6,4,2]`
E. None of the above

Quiz

What's the result of `filter (not . even) [1,2,3,4,5,6]`?

A. `[1,2,3,4,5,6]`
B. `[2,4,6]`
C. **`[1,3,5]`**
D. `[6,4,2]`
E. None of the above

Higher-Order Functions

- map :: (a -> b) -> [a] -> [b]
- filter :: (a -> Bool) -> [a] -> [a]
- foldl :: (b -> a -> b) -> b -> [a] -> b

```
foldl f z0 xs0 = helper z0 xs0
  where
    helper z []     = z
    helper z (x:xs) = helper (f z x) xs
```

Higher-Order Functions

- map :: (a -> b) -> [a] -> [b]
- filter :: (a -> Bool) -> [a] -> [a]
- foldl :: (b -> a -> b) -> b -> [a] -> b
- foldr :: (a -> b -> b) -> b -> [a] -> b

```
foldr _ z []     =  z
foldr f z (x:xs) =  f x (foldr f z xs)
```

foldl vs foldr

```
foldl (+) 0 [1, 2, 3]  ==> ((0 + 1) + 2) + 3  -- Left
foldr (+) 0 [1, 2, 3]  ==> 1 + (2 + (3 + 0))  -- Right
```

Quiz: foldl vs foldr

What's the result of **`foldr (-) 0 [1,2,3,4]`**

A. `[1,2,3,4]`
B. `-10`
C. `0`
D. `-2`
E. None of the above

Quiz: foldl vs foldr

What's the result of **`foldr (-) 0 [1,2,3,4]`**

A. `[1,2,3,4]`
B. `-10`
C. `0`
D. `-2`
E. None of the above

```
=> (-) 1 ((-) 2 ((-) 3 ((-) 4 0)))

=> 1 - (2 - (3 - (4 - 0)))

=> -2
```

Quiz: foldl vs foldr

What's the result of **`foldl (-) 0 [1,2,3,4]`**

A. `[1,2,3,4]`
B. `-10`
C. `0`
D. `-2`
E. None of the above

Quiz: foldl vs foldr

What's the result of **`foldl (-) 0 [1,2,3,4]`**

A. `[1,2,3,4]`

B. `-10`

C. `0`

D. `-2`

E. None of the above

```
=> (-) ((-) ((-) ((-) 0 1)) 2) 3) 4

=> (0 - 1) - 2 - 3 - 4

=> -10
```

Practice

```
reverse :: [a] -> [a]

reverse xs = foldl f base xs

  where

    f a x =

    base =
```

Practice

```
last :: [a] -> a
last [] = error “last: empty list”
last (x:xs) = foldl f base xs
  where
    f a x =
    base =
```

Practice

```
append :: [a] -> [a] -> [a]

append xs ys = foldr f base l

  where

    f x a =

    base =

    l =
```

Practice

```
map :: (a -> b) -> [a] -> [b]

map f xs = foldr fold_fun base xs
```

Practice

```
filter :: (a -> Bool) -> [a] -> [a]

filter p xs = foldr f base xs
```