Things to try at home:

Type says it all!

The "map" pattern

Lets rename the functions to

Yikes, Most Code is the Same!

Example: squares

Let's write a function

Example: ALL CAPS!

For

Type of

So what's the type of

QUIZ: What is the type of

The "filter" pattern

Higher-Order Functions

Can we

Lets rename the functions to

Yikes! Most Code is the Same!

Example: four-letter words

Let's write a function

Example: evens

Last week:

Plan for this week
The “fold-left” pattern

Can you spot the pattern?

Note:

Is what is the most general type of

Accumulate the values from the

The “fold-right” pattern

Let’s refactor

General Pattern

Let’s write a function

Recall: summing a list

Benefits of

Don’t Repeat Yourself

Use a helper function with an extra accumulator argument

Combine result with the head using some binary operation

Recurse on tail

Avoid bugs due to repetition

Syntax Error

Type error

Type Error

3

Use a helper function with an extra accumulator argument

Combine result with the head using some binary operation

Recurse on tail

Avoid bugs due to repetition

Syntax Error

Type error

Type Error
The “fold-left” pattern

```plaintext
foldl f b [x1, x2, x3, x4] =>
helper b [x1, x2, x3, x4] =>
helper (f b x1) [x2, x3, x4] =>
helper (f (f b x1) x2) [x3, x4] =>
helper (f (f (f b x1) x2) x3) [] =>
(f (f (f b x1) x2) x3) x4
```

Accumulate the values from the left

For example:

```plaintext
foldl (+) 0 [1, 2, 3, 4] =>
((0 + 1) + 2) + 3 =>
(((0 + 1) + 2) + 3) + 4
```

Left vs. Right

```plaintext
foldl f b [x1, x2, x3] => f (f (f b x1) x2) x3 -- left
foldr f b [x1, x2, x3] => f x1 (f x2 (f x3 b)) -- right
```

For example:

```plaintext
foldl (+) 0 [1, 2, 3] => (1 + 2) + 3 => 6 -- left
foldr (+) 0 [1, 2, 3] => 1 + (2 + (3 + 0)) => 6 -- right
```

Different types!

```plaintext
foldl :: (b -> a -> b) -> b -> [a] -> b
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Higher Order Functions

Iteration patterns over collections:
- **Filter** values in a collection given a predicate
- **Map** (iterate) a given transformation over a collection
- **Fold** (reduce) a collection into a value, given a binary operation to combine results

HOFs can be put into libraries to enable modularity
- Data structure libraries implement `map`, `filter`, `fold` for its collections
- Data structure clients use HOFs with specific operations
- no need to know the implementation of the collection

Crucial foundation of
- “big data” revolution e.g. MapReduce, Spark, TensorFlow
- “web programming” revolution e.g. jQuery, Angular, React

Generated by Hakyll, template by Armin Ronacher, suggest improvements here.