CS 421 --- Higher Order Functions Activity

Manager	Keeps team on track	
Recorder	Records decisions / QC	
Reporter	Reports to class	
Reflector	Assesses team performance	

Learning Objectives

Mapping, folding, and zipping allow us to abstract away common list computations. Knowing how to use them will make you more productive as a programmer.

1. Reduce code size by using `map`, `foldr`, and `zipWith`.
2. Use type signatures to implement `curry`, `uncurry`, and `flip`.

Mapping (5 minutes)

Consider the following code and sample run, stolen from *Haskell Programming From First Principles*, Chapter 9.

```
Prelude> map (+1) [1, 2, 3, 4]
[2,3,4,5]
Prelude> map (1-) [1, 2, 3, 4]
[0,-1,-2,-3]
Prelude> map (take 2) [[1, 4, 9], [2, 3, 5], [11, 12, 13]]
[[1,4],[2,3],[11,12]]
Prelude> zipWith (+) [1, 2, 3] [10, 11, 12]
[11,13,15]
Prelude> zipWith max [10, 5, 34, 9] [6, 8, 12, 7]
[10,8,34,9]
```

Problem 1) What does the `(+1)` code mean?

Problem 2) In the second example, why are we using `(1-)` and not `(-1)`?

Problem 3) How would you describe the difference between `map` and `zipWith`?

Reducing

Haskell has some functions `foldr` and `foldl` that behave like the `reduce` function found in other languages such as Python and JavaScript. Perhaps you have used it!

Consider the following sample run.

```
Prelude> :set +m
Prelude> { showPair x "" = x
Prelude| ; showPair x y = "(" ++ x ++ "," ++ y ++ ")"
Prelude| }
Prelude> showPair "10" ""
"10"
Prelude> showPair "10" "20"
"(10,20)"
```

Problem 4) What do you think the `:set +m`, curly braces, and semicolons are about? What happens if we don't use them?

Problem 5) What does `showPair` do?

Now consider this run:

```
Prelude> sumList1 xx = foldr (+) 0 xx
Prelude> sumList1 [3,4,5,6]
18
Prelude> sumList2 xx = foldl (+) 0 xx
Prelude> sumList2 [3,4,5,6]
18
Prelude> pairList1 xx = foldr showPair "" xx
Prelude> pairList2 xx = foldl showPair "" xx
Prelude> pairList1 ["3","4","5","6"]
"(3,(4,(5,6)))"
Prelude> pairList2 ["3","4","5","6"]
"((((,3),4),5),6)"
```

Problem 6) Is there an observable difference between `sumList1` and `sumList2`?

Problem 7) Is there an observable difference between `pairList1` and `pairList2`?

Problem 8) Write a function `prodList` using the same technique you see here.

Problem 9) Do you think either `foldr` or `foldl` is tail recursive? Why or why not?

List Comprehensions

List comprehensions are similar to higher order functions, and can allow you to write very compact code.

```
Prelude> [x+1 | x <- [1..10]]
[2,3,4,5,6,7,8,9,10,11]
Prelude> [x+1 | x <- [1..10], x>5]
[7,8,9,10,11]
Prelude> stuff = [8,6,7,5,3,0,9]
Prelude> [ x+1 | x <- stuff ]
[9,7,8,6,4,1,10]
Prelude> [ x+1 | x <- stuff, x > 5]
[9,7,8,10]
Prelude> [ x+1 | x <- stuff, x > 5, even x]
[9,7]
Prelude> [ x + y | x <- stuff, y <- [10,20]]
[18,28,16,26,17,27,15,25,13,23,10,20,19,29]
```

Problem 10) What is the purpose of the `x <- stuff` expression?

Problem 11) What is the purpose of `x > 5`, and `even x`?

Problem 12) How do you describe the order in which `x` and `y` are created in the last example?

Problem 13) What does the following code do?

```
guess [] = []
guess (x:xs) = guess [y | y <- xs, y < x]
               ++ [x] ++
               guess [y | y <- xs, y >= x]
```

Currying

Consider these two functions:

```
Prelude> uplus (a,b) = a + b
Prelude> cplus a b = a + b
```

Problem 14) What is the difference between `cplus` and `uplus`? What would it look like to use them?

The function `cplus`, which is written in idiomatic Haskell, is said to be *curried*. This makes it taste better.

Problem 15) Write a function `curry :: ((a,b) -> c) -> a -> b -> c` that takes a non-curried function and returns an equivalent curried version.

```
Prelude> plus (a,b) = a + b
Prelude> :t plus
Num a => (a,a) -> a
Prelude> cplus = curry plus
Prelude> cplus 10 20
30
```

Problem 16) Write a function `flip :: (a -> b -> c) -> (b -> a -> c)` that takes a function that takes two arguments and returns an equivalent function where the arguments have been reversed.

```
Prelude> sub a b = a - b
Prelude> flip sub 10 2
-8
```

Problem 17) Consider the types of `flip` and `curry`. Can you write another function that has either of those types? Why or why not?

Higher Order Functions Activity--- Reflector's Report

Manager	Keeps team on track	
Recorder	Records decisions	
Reporter	Reports to Class	
Reflector	Assesses team performance	

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?

Higher Order Functions Activity --- Team's Assessment (SII)

Manager or Reflector: Consider the objectives of this activity and your team's experience with it, and then answer the following questions after consulting with your team.

1. What was a **strength** of this activity? List one aspect that helped it achieve its purpose.

2. What is one things we could do to **improve** this activity to make it more effective?

3. What **insights** did you have about the activity, either the content or at the meta level?