COMP105 Lecture 14

Higher Order Functions

Higher order functions

A **higher order function** is a function that

- Takes another function as an argument, or
- Returns a function

```
apply_twice :: (a -> a) -> a -> a
apply_twice f input = f (f input)
```

```
ghci> apply_twice tail [1,2,3,4]
[3,4]
```

Apply_twice examples

```
apply_twice :: (a -> a) -> a -> a
apply_twice f input = f (f input)
```

```
ghci> apply_twice ((+) 2) 2
6
```

```
ghci> apply_twice (drop 2) [1,2,3,4,5]
[5]
```

```
ghci> apply_twice reverse [1,2,3,4]
[1,2,3,4]
```

The apply_twice type

```
apply_twice :: (a -> a) -> a -> a
apply_twice f input = f (f input)
```

The type specifies that

- `f :: (a -> a)`
- `input :: a`
- The function returns type `a`

So the following will give a type **error**

```
ghci> apply_twice head [[1,2], [3,4]]
```

Function composition

Function **composition** applies one function to the output of another

- Composing `f` with `g input` gives `f (g input)`

```
compose :: (b -> c) -> (a -> b) -> a -> c
compose f g input = f (g input)
```

```
ghci> compose (+1) (*2) 4
9
```

```
ghci> compose head head [[1,2], [3,4]]
1
```

The . operator

In Haskell compose is implemented by the . operator

```
ghci> compose head head [[1,2], [3,4]]
1

ghci> (head . head) [[1,2], [3,4]]
1


ghci> :t (.)
(.) :: (b -> c) -> (a -> b) -> a -> c
```

The . operator

The . operator is particularly useful when composing a **long list** of functions

```
f list = length (double (drop_evens (tail list)))
```

```
f' list = (length . double . drop_evens . tail) list
```

The use of . removes the need for nested brackets

- but it is stylistic
- you never need to use .

The $ operator

```
evaluate :: (a -> b) -> a -> b
evaluate f input = f input
```

This function just **evaluates** its input

```
ghci> evaluate length [1,2,3]
3
```

The $ operator

The $ operator is exactly the same as **evaluate**

```
ghci> ($) length [1,2,3]
3

ghci> length $ [1,2,3]
3

ghci> :t ($)
($) :: (a -> b) -> a -> b
```

The $ operator

The $ operator has the lowest **precedence** of all operators

- It is mainly used to avoid brackets

```
ghci> length ([1,2,3] ++ [4,5,6])
6

ghci> length $ [1,2,3] ++ [4,5,6]
6



ghci> (length . tail) [1,2,3,4]
3

ghci> length . tail $ [1,2,3,4]
3
```

Exercises

1. Write a function `applyThrice f x` that applies f to x three times

2. Use `.` and `$` to remove the brackets from the following function

```
f x = succ (sum (tail (tail x)))
```