Reading 9

# Exercise 1: Summarize

Exploring advanced pattern matching and function definitions, uncovering the power of higher-order and predefined functions, while demonstrating how anonymous functions can be simplified for creating complex and elegant solutions.

# Exercise 2: Read & Code

ML uses space separated syntax and treats functions as left associative which removes the need for commas. This in turn means foldr does not require commas as it takes in three inputs, a binary function, an initial value, and a list in that order.

Since op ^ is a keyword shortcut for an operator for concatenation, it can be broken up into a full anonymous function: fn (x, y) => x ^ y; This takes in a tuple of two string values and adds then together, effectively doing the same as op ^. Therefore, our new function looks more like this:

Foldr (fn (x,y) => x^y “” [“a”, “b”, “c”]);

A walkthrough of foldr using list [a, b, c] is as follows.

(\*Initial Call\*)

foldr (fn (x, y) => x ^ y) "" ["a", "b", "c"];

(\*1. Apply initial value “” to starting from right side of list “c”\*)

(fn (x, y) => x ^ y) ("c", "") (\* Result: "c" \*);

(\*2. Apply the next element to the previous value made “b”\*)

(fn (x, y) => x ^ y) ("b", "c") (\* Result: "bc" \*);

(\*3. Apply the next element to the previous value made “a”\*)

(fn (x, y) => x ^ y) ("a", "bc") (\* Result: "abc" \*);

(\*The final result is "abc"\*)

# Exercise 3: Inquire

The ML foldr function, JavaScript reduce, and Ruby reduce functions all try to iterate through a list to reduce it to a single value. They each take a binary function, initial value, and a list to iterate through. The main differences are their syntax and that foldr works from right to left, while reduce from JavaScript and Ruby go from left to right.