PCL1 Exercise(s) 2

Contents

F# Core Types

In the following exercises, we'll practice defining list functions, pattern matching, lambda and higher order functions.

2.1.1 – Pattern matching and recursion.

a. Define a function ***`vowelToUpper c`*** that converts the characters `a, e, i, o` and `u` to capitals (upper case). All other characters should be returned unchanged. You are *not* allowed to use the standard F# method `.ToUpper()` on strings! You can however use substring().

b. Define another function (choose your own function name) to convert all occurrences of `a, e, i, o` and `u` in a string to capitals. Write the function to use recursion and the ***`vowelToUpper`*** in 2.1.1a above.

2.1.2 – Functions on lists

a. Define a function *pmLength ls* that computes the length of a list. Use pattern matching.

   For example, *pmLength [‘x’; ‘y’; ‘z’]* should return *3*.

b. Define a function *pclSum ls* that sums all the numbers in a list.

   For example, *pclSum [2; 3; 5; 8]* should return *18*.

2.1.3 – List processing

Define a function, *pmTakeSome n ls* that returns list of first *n* elements from the list *ls*. Define the function using pattern matching:

Example *pmTakeSome 2 ["apple";"banana"; "carot"; "dewberry"];;* should return return *["apple"; "banana"]*

2.2.1 – List Functions - fold

fold is a List module function that when given a list of values, returns a single piece of data by applying a function to each element of the list. It takes three parameters, a function, and initial accumulator value and a list to fold over.

a. Define the function *pmFold* . Use pattern matching. You are *not* allowed to use the standard F# functions.

   For example, *pmFold (+) 0 [1; 2; 3]* should return *6*.

b. Define a function *pclSumWithFold* that changes the ***pclSum*** you defined previously in 2.1.2b) to use *pmFold* defined above.

2.2.2 – List Functions - foldback

foldBack is another List module function that when given a list of values, returns a single piece of data but folds from the right. It takes three parameters, a function, and initial accumulator value and a list to fold over.

a. Define the function *pmFoldBack* . Use recursion and pattern matching. You are *not* allowed to use the standard F# functions.

   For example, *pmFoldBack (+) 0 [1; 2; 3]* should return *6*.

b. Define a function *pclSumWithFoldBack* that changes the ***pclSum*** you defined previously in 2.1.2b) to use *pmFoldBack* defined above.

c. Compare the results. Could you find a test where there are different results?

2.2.3 – List Functions - map

a. Define a function *pmIncList* that adds one to each element in a list of integers. You are *not* allowed to use the standard F# functions. Use recursion and pattern matching. For example, *pmIncList [2; 3; 1; 4]* should return *[3; 4; 2; 5]*.
b. Define a function, *pclMap* that applies an arbitrary function ***f*** to the elements in a list.
c. Define a function *pclIncListWithMap* that changes the ***pmIncList*** you defined previously in 2.2.3a to use *pclMap* defined above.

2.2.4 – List Functions - filter

a. Define a function, *pclFilter* that removes all elements from a list that do not satisfy a given predicate.
b. Define a function *pclEven* that returns true for even numbers.
c. Test the functions:

   For example, *pclFilter pclEven [0; 1; 2; 3; 4; 5;6;7;8;9]*

   should return *[0; 2; 4;6;8]*

Higher-order functions, partial function application, closures

2.3.1 – HOF

Write an F# function ***countNumOfVowels*** to count the number of vowels in a given string. The type is:

countNumOfVowels : string -> int * int * int * int * int.
Consider using List.fold.

Test with: `countNumOfVowels "Higher-order functions can take and return functions of any order"`

2.3.2 – Fibonacci numbers

Consider the sequence of Fibonacci numbers defined as follows:

$$\mathrm{Fib}(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2) & \text{otherwise} \end{cases}$$

By the definition, Fibonacci numbers have the following sequence, where each number is the sum of the previous two: `0, 1, 1, 2, 3, 5, 8, 13, 21, ...`

Define an F# function `pclFib n` that, when given a number, returns the nth Fibonacci number.

2.3.3 – Partial function application

a. Define two F# functions `doubleNum x` that multiplies x by 2 and `sqrNum x` that multiplies x by itself.
b. Define another F# function `pclQuad x` that applies the `doubleNum` function defined above twice.
c. Define another F# function `pclFourth x` that applies the `sqrNum` function defined (in a.) above twice.