Exercise 9 - Pattern Matching

Objective

The objective of these exercises is to replace the if/else construct with pattern matching and get some experience of using pattern matching with recursion to complete implementing our list class. We also investigate extractors and deep matching to find specific details from objects

References

Chapter 9 of the slides – 198 to 212

Overview

1. Rewrite your "n-th list item" function from the functional chapter to use pattern matching
2. Write a function which removes all instances of a specific item from a list
   1. The input to the function should be a list of items (make it generic so anything can be used) and the specific items to remove (a word, a letter or a number for example)
   2. The function should return the new list
3. Create simple Person and Address objects, such that Person has a name and address, and Address consists of a city and country.
   1. We need both the class definition and an companion object for the classes
   2. Implement extractors in the companion objects for both classes
   3. Write a livesIn() function that uses pattern matching on a Person to tell you if they live in London, New York, or somewhere else.

Case classes can be used with pattern matching as they contain extractors to get the specific information the matcher is after. Extractors are automatically created for us when using case classes, but it is useful to know how to implement them ourselves to get specific information from an object.

1. Create a trait called Expression that has no members
2. Add these three case classes that represent parts of expressions.

// A class to represent an integer constant

case class Const(v: Int) extends Expression

// A class to represent a negation operation

case class Neg(e: Expression) extends Expression

// A class to represent an addition operation

case class Add(l: Expression, r: Expression) extends Expression

This allows us to write statements like this:

// 10 + (-(3 + 4))

Add(Const(10), Neg(Add(Const(3), Const(4))))

1. Now define a function called eval that can evaluate expressions. It will take an expression as an argument and return an Int as the final value

The body of the function is a match on the expression, and should have three cases:

* + one for Const, which simply evaluates to the value
  + one for Neg, which negates the result of calling eval recursively
  + one for Add, which returns the sum of calling eval on its left and right arguments

Test your code using the expression below:

val ex = Add(Const(10), Neg(Add(Const(3), Const(4))))

println(eval(ex))

Adding Functionality to the List Class

In this final exercise, you will add some extra functionality to the List that you created earlier. You will find that the implementations of these functions make frequent use of use pattern matching and recursion.

Implement the following functions, adding them to the List object

1. drop(lst, n), which returns a new list, ignoring the first n items of lst
2. append(lst1, lst2) which appends lst2 onto lst1, returning the result
3. reverse(lst) which returns a reversed copy of lst

If you have time, try these harder ones:

1. map(lst, f) which applies a function to each element of the list
2. filter(lst, pred) which removes items from lst that don’t match a predicate
3. take(lst, n), which returns a new list consisting the first n items of lst