UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Advanced Programming II
Lab 3. Using the List API

Unless otherwise indicated, you have to implement the practice functions by using the functions of theScala's `List` class (`foldRight`, `foldLeft`, `map`, etc.). The functions `foldRight` and `foldLeft` are defined as follows:

```
def foldRight[B](acc:B)(f:(A,B)=>B):B = l match
  case Nil => acc
  case a::r => f(a,r.foldRight(acc)(f))

def foldLeft[B](acc:B)(f:(B,A)=>B):B = l match
  case Nil => acc
  case a::r => r.foldLeft(f(acc,a))(f)
```

1. Using `foldRight`, define the functions
   ```
   def sum(l:List[Int]):Int
   def product(l:List[Int]):Int
   def length[A](l:List[A]):Int
   ```
   which, respectively, add/multiply the items in the list l, and calculate its length, respectively. Examples:
   ```
   sum(List(1,2,3)) == 6
   product(List(1,3,5)) == 15
   length(List("Hola", " ", "Mundo")) == 3
   ```

2. Using `foldLeft` or `foldRight` define the functions
   ```
   def reverse[A](l:List[A]):List[A]
   def append[A](l1:List[A],l2:List[A]):List[A]
   ```
   which calculate the length of the list l and invert it. Examples:
   ```
   reverse(List(1,2,3)) == List(3,2,1)
   append(List(1,2,3),List(1,2)) == List(1,2,3,1,2)
   ```

3. Using `foldLeft` or `foldRight` defines the function
   ```
   def exists[A](l:List[A],f:A=>Boolean):Boolean
   ```
   which checks whether l has any element that satisfies f. Examples:
   ```
   exists(List(1,2,3),_>2) == true
   exists(List("Hola","Mundo"),_.length>=5) == true
   exists(List("Hola","Mundo"),_.length<3) == false
   ```

4. Define the function
   ```
   def f(l:List[Int]):List[Int]
   ```
   that given the list l constructs a list with the absolute values of the negative elements of l. For example,
   ```
   f(List(1,-2,3,-4,-5,6)) == List(2,4,5)
   ```
   Implement the function following two approaches:
   a) By means of a tail-recursive function, making use of pattern matching.
   b) Using only higher-order functions (map, filter, etc.).

5. Using `foldRight` implement the function:
   ```
   def unzip[A](l:List[(A,B)]):(List[A],List[B])
   ```
   that given a list of tuples `List((a1, b1), ..., (an, bn))` returns two lists of the form `List(a1, ..., an)` and `List(b1, ..., bn)`. Example:
   ```
   unzip(List((1,'a'),(2,'b'),(3,'c'))) == (List(1,2,3), List('a','b','c'))
   ```

6. Using `foldRight` implement the function:
```
def compose[A](lf:List[A=>A],v:A):Boolean
```
that given a list of functions `List(f1,f2,..,fn)` and a value `v` calculates `f1(f2(... fn(v))...)`. For example,
```
compose(List[Int => Int](Math.pow(_,2).toInt, _+2), 5) == (5+2)^2 == 49
```

7. Using `foldRight` implement the function:
```
def remdups[A](lista:List[A]):List[A]
```
that removes adjacent duplicates from the `list`. For example:
```
remdups(List(1,1,3,3,3,2,1,2,2,1,2)) == List(1, 3, 2, 1, 2, 1, 2)
```

8. Using `foldRight` implement the function:
```
def fibonnaci(n:Int):Int
```
which given a number n calculates the nth Fibonacci number. For example:
```
fibonacci(5) == 5
fibonacci(10) == 55
```

9. Using `foldRight` implement the function:
```
def inits[A](l:List[A):List[List[A]]
```
which constructs a list with all the lists prefixes of `l`. For example:
```
inits(List(1,2,3)) == List(List(),List(1),List(1,2),List(1,2,3))
inits(List(3)) == List(List(),List(3))
inits(List()) == List(List())
```

10. Write a function
```
def halfEven(l1:List[Int],l2:List[Int]):List[Int]
```
which takes as input two lists of integers and adds their elements l1(i) and l2(i). If the sum l1(i) + l2(i) is even, it is divided by two. Otherwise, it is removed from the resulting list. For example:
```
halfEven(List(1,2,3,4),List(3,2,4)) == List(2,2)
```
Implement the function following two approaches:

a) By means of a tail-recursive function, making use of pattern matching.

b) Using only higher-order functions (map, filter, etc.).

11. Given a list of strings, each one beginning with "ERROR", "INFO", or "WARNING", we want to (1) count the number of messages of each type and (2) extract the error messages and save them to a list.

Given the following data:
```
val logs = List(
  "ERROR: Null pointer exception",
  "INFO: User logged in",
  "ERROR: Out of memory",
  "WARNING: Disk space low",
  "INFO: File uploaded",
  "ERROR: Database connection failed"
)
```
The output for (1) must be:
```
HashMap(WARNING -> 1, ERROR -> 3, INFO -> 2)
```
and for (2):
```
List(ERROR: Null pointer exception, ERROR: Out of memory, ERROR: Database connection failed)
```

12. Given a list of sales transactions represented as (productName, quantitySold, pricePerUnit), we want to (1) calculate the total revenue and (2) get the list of incomes per productName (quantitySold * pricePerUnit) greater than (or equal) to 100 sorted by the income.

Given the following data:
```
val sales = List(
```

```
    ("Laptop", 2, 1000.0),
    ("Mouse", 10, 15.0),
    ("Keyboard", 5, 50.0),
    ("Monitor", 3, 200.0),
    ("USB Drive", 20, 4.0)
  )
```

The expected output for (1) is

```
3100.0
```

And for (2) is:

```
List((Laptop,2000.0), (Monitor,600.0), (Keyboard,250.0), (Mouse,150.0))
```

13. Given a list of sentences, we want to extract the **unique words** (i.e., remove duplicates), convert them to lowercase, and remove non-meaningful words (such as "to", "the", "is", and "of").

Given the following data:

```
val sentences = Set(
  "Scala is a functional language",
  "The power of functional programming is great",
  "Functional programming is elegant"
)
val stopWords = Set("a", "the", "is", "of")
```

we should produce a set like:

```
HashSet(programming, language, scala, power, elegant, functional, great)
```

14. :Given a list of words, we want to count the **frequency of each word**.

Given the following data:

```
val words = List("scala", "is", "awesome", "scala", "functional", "scala",
"is", "great")
```

the expected output is:

```
HashMap(is -> 2, awesome -> 1, scala -> 3, functional -> 1, great -> 1)
```

15. Given two maps that represent the stock of products in two different warehouses, we want to combine them **by adding the amounts of** the products that appear in both.

Given the following data:

```
val warehouse1 = Map("laptop" -> 5, "mouse" -> 20, "keyboard" -> 10)
val warehouse2 = Map("laptop" -> 3, "mouse" -> 15, "monitor" -> 8)
```

the expected output is:

```
Map(laptop -> 8, mouse -> 35, keyboard -> 10, monitor -> 8)
```