Lab 3. Lists in Scala

Objectives:

- get familiar with **pattern matching** lists, as well as common list operations from Scala and how they work
- get familiar with common **higher-order functions** over lists (partition, map, foldRight, foldLeft, filter)

3.1. Common list operations

3.1.1. Write a function which returns true if a list of integers has at least `k` elements. Use **patterns**. Write a second function which returns `true` if the list has at least `k` elements that satisfy a predicate.

```
def atLeastk(k: Int, l: List[Int]): Boolean = {
  if (k == 0) ???
  else ???
}

def atLeastkPred(pred: Int => Boolean)(k: Int, l: List[Int]): Boolean = ???
```

3.1.2. Write a function which returns the first `n` elements from a given list. The function should **not** be implemented as tail-recursive.

```
def take(n: Int, l: List[Int]): List[Int] = ???
//take(3,List(1,2,3,4,5)) = List(1,2,3)
```

3.1.3. Write a function which *drops* the first `n` elements from a given list.

```
def drop(n: Int, l: List[Int]): List[Int] = ???
//drop(3,List(1,2,3,4,5)) = List(4,5)
```

3.1.4. Write a function which takes a predicate `p: Int ⇒ Boolean`, a list `l` and returns a sublist of `l` containing those elements for which `p` is true. The function should be **curried**.

```
def takeP(p: Int => Boolean)(l: List[Int]): List[Int] = ???
//takeP(_%2 == 0)(List(1,2,3,4,5,6)) = List(2,4,6)
```

3.1.5. Write a function which uses a predicate to partition (split) a list.

```
def part(p: Int => Boolean)(l: List[Int]): (List[Int], List[Int]) = ???
// part(_%2 == 0)(List(1,2,3,4,5,6)) = (List(2,4,6),List(1,3,5))
```

3.1.6. Write a function that reverses the elements of a list. Use a `fold` function (determine if it should be right or left).

```
def rev(l: List[Int]): List[Int] = ???
// rev(List(1,2,3,4,5,6)) = List(6,5,4,3,2,1)
```

3.2. String processing

In what follows, we shall encode a String as a list of characters, using the type defined below:

```
type Str = List[Char]
```

Add this type alias to your code before solving the following exercises.

The following is an input test. You can add more examples to it:

```
val l: List[Str] = List("matei@gmail.com", "mihai@gmail.com", "tEst@mail.com", "email
@email.com", "short@ax.ro").map(x => x.toList)
```

Use `map`, `foldr` / `foldl`, instead of recursive functions.

3.2.1. Remove uppercases from emails. (Do **not** use recursion). Use the Internet to find the appropriate character function.

```
def remUpper(list: List[Str]): List[Str] = ???
```

3.2.2. Write a function which removes emails longer than a given size. Try to think of two ways to implement this using already defined functions (do not define your own auxiliary functions).

```
def longer(k: Int, list: List[Str]): List[Str] = ???
```

3.2.3. Count the number of emails longer than k characters. Use `foldRight`.

```
def howMany(k: Int)(list: List[Str]): Int = ???
```

(!) 3.2.4. Implement a function that tokenizes a String split by a given delimiter using `foldRight`. Try to figure out what the accumulator should do.

```
def mySplit(l: Str, sep: Char): List[Str] = ???
```

3.2.5. Implement a function that return the domains without the dot (ex. `gmail`).

```
def domains(list: List[Str]): List[Str] = ???
```

3.3. Gradebooks

More general implementations of `taken`, `dropn` and `part` are already implemented in Scala and can be used as member functions of lists. Examples are shown below:

```
val l = List(1,2,3,4,5,6,7,8,9)
l.take(3)
l.drop(3)
l.partition(_%2 == 0)
```

In what follows, we shall encode a gradebook as a list of pairs `(<name>,<grade>)`, where `<name>` is a String and `<grade>` is an Int. Example:

```
val gradebook: List[(Str, Int)] = List((List('G'),3), (List('F'), 10), (List('M'),6),
(List('P'),4))
```

To make the type signatures more legible, we can introduce type aliases in Scala:

```
type Gradebook = List[(Str,Int)] //the type Gradebook now refers to a list of pairs o
f String and Int
```

Add this type alias to your code before solving the following exercises.

3.3.1. Write a function which adds one point to all students that satisfy a given predicate (ex: grade >= 5), and leaves all other grades unchanged.

```
def increment(g: Gradebook, p: (Str, Int) => Boolean): Gradebook =
  g.map(???)
```

3.3.2. Find the average grade from a gradebook. You must use `foldRight`.

```
def average(g: Gradebook): Double = ???
```

3.3.3. Write a function which takes a gradebook and returns the list of names which have passed. Use filter and map from Scala.

```
def pass(g: Gradebook): List[Str] = ???
```

3.3.4. Implement merge-sort (in ascending order) over gradebooks:

```
def mergeSort(l: Gradebook): Gradebook = {
   def merge(u: Gradebook, v: Gradebook): Gradebook = ???
   ???
}
```

3.3.5 Write a function which takes a gradebook and reports all passing students in **descending** order of their grade.

```
def honorsList(g: Gradebook): List[Str] = ???
```