Scala - Exercises 2 - Functions, HOFs and Currying

Warm Up

1. Write a function **value** that takes five strings and returns the sum of the length of all of the strings. Write this function using both long hand syntax (e.g. FunctionX) and short hand. Which do you prefer and why?
2. Write a function that takes an int as an argument and returns *another* function as a result
3. Explain (in words) and provide an example implementation (the implementation does not need to be meaningful but it must compile) what the following type signature says:

```
val function: (String, (Int, (String => Int)) => Int) => (Int => Int)
```

4. Convert the following function in to a curried version and provide an example of calling both the curried and uncurried version of the function

```
val addThreeNumbers(x: Int, y: Int, z: Int) = x + y + z
```

5. Write a function that makes your recursive function used to concatenate two string into a generic function (you should test your function on Lists, Vectors and Strings). The definition of the function is:

```
def concatenate[T](item:T, item2:T, concat: T=> T): T
```

Now we're warm...

Our aim here is to write a generic FileMatcher using higher order functions that allows us to return the file only if some specific matcher is matched. We'll build this up step by step in the exercise

```
object FileMatcher:
    private def filesHere = (new Java.io.File(".")).listFiles
```

1. Add to the code above methods as follows:
    1.
```
def filesEnding(query: String)
```

2. `def filesContaining(query: String)`
3. `def filesRegex(query: String)`

2. Notice the repetition in the above code, write a function called filesMatching, that uses a higher order function to remove the repetition
3. Re-write filesEnding, filesContaining and filesRegex in terms of your filesMatching function
4. Provide some example uses of your fileMatcher class, passing in the appropriate function
5. Think of a different function you might want to use your fileMatcher for and write a usage of it, where that function is passed as an anonymous function
6. Change (or rewrite) your function to used curried parameters, which do you prefer and why?

Challenge

1. Complete the implementation of the following merge sort function:

```
def msort[T](less:(T,T) => Boolean)(xs: List[T]): List[T] = {
  def merge(xs:List[T], ys: List[T]): List[T] = {
    ???
  }

  ???
}
```

2. Using the above write functions intSort and reverseIntSort which sorts integers in ascending and descending order respectively.
3. By using the intSort and reverseIntSort methods above, explain how the use of currying has helped us to make our code easier to read

Hints:

- Look up merge sort and how it works and try to implement it using recursion.
- **Don't** worry about your solution being tail recursive but it *should* be recursive
- You can split a list in scala by using the splitAt method