Practical Sheet nº1

Content

- Values, Expressions and Types
- Functions: Types and Definition
- Introduction to Recursive Functions

Exercise 1

a) Create a Scala file (Fun.scala) with the following code

```
//singleton object

object Fun {

  def func1(x : Double, y :Int) = x + (70*y)

  def ex(a: Double) = 50 * a

}
```

b) In the Scala REPL load the file, using:

```
>:load C:\...\Desktop\Fun.scala
```

c) Use the interpreter to identify what is the type of the functions declared

```
>:type Fun.func1(_, _)
>:t  Fun.ex(_)
```

d) Evaluate the following expression

```
Fun.func1(Fun.ex(10.2), 1)
```

Exercise 2

a) Define a method that receives two pairs of integers and returns a pair of integers. The first element of the result being the sum of the first input pair, and the second element of the result pair the product of the second input pair.
b) Write a method that, given three Int numbers, returns a pair containing the largest element in the first element, and the second largest number in the second element.
c) Write a method that receives a triple of Int numbers and returns a triple where the same numbers are ordered in descending order.
d) The sides of any triangle respect the following constraint: the sum of the lengths of any two sides, is greater than the length of the third side. Write a method that receive the length of three segments and return a Boolean value indicating whether the constraint is satisfied.
e) Write a method `abrev` that receives a string containing a person's name and returns a string with the first and last name.
For example, `abrev` "José Carlos Martins Sousa" = "José Sousa"

The pre-defined function `split` might be useful:

```
scala> "hello world".split(" ")
res0: Array[java.lang.String] = Array(hello, world)
```

Exercise 3

Now consider the following mathematical definition of the factorial function for non-negative integers:

```
0! = 1
n! = n*(n-1)*(n-2)*...*1
```

Being `n! = n*(n-1)!` a possible Scala definition of the factorial method is:

```
def fact(n: Int):Int = if (n==0) 1 else n * fact(n-1)
```

Note that this method is recursive, i.e. it appears in the expression that defines it. The method is also said to invoke itself. The method calculation ends because the stop case is always reached (n = 0).

a) Define a method that calculates the result (Int) of the following exponentiation x^y without resorting to predefined functions.
b) Define a method that receives a Int List and constructs a pair with the first and the last element of the list.
c) Define a method that, given a Int list, gives the pair with that list and its length.
d) Define a method that, given a Double list, calculates its average. The pre-defined function `tail` of `List` might be useful.