Ejercicios programación funcional

Algoritmos 3

Brasburg-Cano-Raik
FIUBA

1 de septiembre de 2023

Los siguientes ejercicios deben ser pensados dentro del paradigma funcional. Pueden ser resueltos utilizando otros paradigmas pero el objetivo es el de lograr resolverlos utilizándolo, específicamente utilizando el lenguaje *Scala*.

Para resolverlos se deben utilizar los conceptos y las herramientas vistas en clase, aunque puede ser necesario para resolverlos investigar otras herramientas por propia cuenta.

Ejercicios básicos:

1. Escribir una función *entreNumeros(inicio: Int, fin: Int): List[Int]* que retorne una lista que incluya todos los números enteros entre *inicio* y *fin* incluyéndolos.

2. Escribir una función *repetidos(lista1: List[Int], lista2: List[Int]): List[Int]* que retorne una nueva lista que contenga los elementos que están presentes en ambas listas.

3. Escribir una función *eliminarRepetidos(lista: List[Int]): List[Int]* que retorne una nueva lista que contenga los mismos elementos que la original pero sin elementos repetidos.

4. Escribir una función *aplicar(lista: List[Int], f: (Int, Int) =¿Int): List[Int]* que retorne una nueva lista que contenga los resultados de aplicar f a cada elemento de *lista*

5. Escribir una función *sumar(lista: List[Int]): Int* que retorne la suma de todos los elementos de la lista.

Ejercicios sobre funciones impuras: Indicar y justificar cuales de las siguientes funciones son impuras:

1.
```scala
def f(x: Int, y: Int): Int = {
  x + y
}

```

2.
```scala
def fecha(): String = {
  LocalDate.now.format(DateTimeFormatter.ofPattern("
    yyyyMMdd"))
}

```

3.
```scala
def contar(l: List[Int], e: Int): Int = {
  var cont = 0;
  l.foreach(elemento => {
    if(elemento == e) {
      cont = cont + 1
    }
  })
  cont
}

```

4.
```scala
import scala.collection.mutable.Map

def actualizarAUno(mapa: Map[Int, Int]) = {
  for (k, v) <- mapa do
    mapa(k) = 1
}

```

5.
```scala
import scala.collection.mutable.Map

def randomEntre(a: Int, b: Int): Int = {
  val rand = new scala.util.Random
  rand.between(a, b)
}

```

6.
```scala
def merge[A](list1: List[A], list2: List[A]): List[A] = {
  list1 ::: list2
}

```

7.
```scala
import java.io.PrintWriter

def guardarEnArchivo(texto: String, ruta: String): Unit =
    {
  val escritor = new PrintWriter(ruta)
  try {
    escritor.write(texto)
  } finally {
    writer.close()
  }
}

```

Ejercicios complejos: (no estan ordenados por dificultad necesariamente)

1. Escribir una función *contar(palabras: List[String]): Map[String, Int]* que retorne un mapa donde las claves sean las palabras de la lista pasada por parámetro y los valores la cantidad de apariciones que tiene dicha palabra en la lista

2. Escribir una función *topK(numeros: List[Int], k: Int, f: (Int, Int) =>Int): List[Int]* que retorne una lista con los k elementos de *números* mas grandes según la función *f*. La respuesta debe estar ordenada segun el criterio de la función *f*.

3. Definir la función *curry*, que dada una función de dos argumentos, devuelve su equivalente currificada.

4. Definir la función *uncurry*, que dada una función currificada de dos argumentos, devuelve su versión no currificada equivalente. Es la inversa de la anterior.

5. Generar la lista de los primeros mil números primos.

6. Dado un texto, crear una función que tome el texto como entrada y devuelva un mapa que muestre la frecuencia de cada palabra en el texto. Los espacios no deben ser considerados

7. Escribir una función que reciba un entero n y una lista l y retorne una lista con la diferencia de cada uno de la lista a n. Solo debe tener en cuenta los números para los cuales la diferencia es mayor a 10, ignorar el resto.

8. Escribir una función que junte los números de una lista sin usar métodos de ordenamiento. Ej: (1, 2, 3,4, 1, 3) =¿(2, 1, 1, 3, 3, 4).

9. Escribir una función que retorne el n-esimo número de la sucesión de Fibonacci a partir de un número n.

10. Escribir una función que reciba un string y retorne un booleano mostrando capicua o no.

11. Escribir una función que retorne el máximo de una lista (no es valido usar la función reverse):

    Utilizando match

    Sin utilizarlo