**Entregable #2.1 [ejercicios] [semana 19 Feb 2018]**

* **Objetivos específicos** que persigue la realización del trabajo.
  + - Practicar con conceptos básicos de Scala sobre eclipse
* **Competencias**: CC14

**Apellidos y nombre de los componentes del grupo**

|  |
| --- |
| Gutierrez Ospina Anderson |
| Altamirano Cargua Fabricio |

1. **Conceptos básicos de Scala**

* Abre el entorno "eclipse". Crea un proyecto con tu nombre a través de los siguientes pasos:

File -> New -> Scala Project

Project name: nombreEstudiante

* Situado sobre el proyecto recién creado, crea un nuevo objeto:

File -> New -> Scala Object

Name: pruebaScala

En la práctica 1 comentábamos que Scala importa implícitamente los paquetes “java.lang”, “scala” y “Predef”, y que gracias a ello, algunos programas no requieren importaciones explícitas. En esta práctica vamos a trabajar con algunos paquetes que no se importan automáticamente. Por comodidad, puedes preparar un esqueleto de programa con una serie de importaciones fundamentales y luego rellenar el esqueleto con las instrucciones concretas que requiera el problema pedido en cada momento. Estas son algunas de las sentencias de importación que pueden requerirse en esta práctica, junto con el esqueleto de programa para pruebas:

import scala.io.StdIn.{readLine, readInt} // Para lectura de datos

import scala.math.\_ // Para funciones matemáticas

import scala.collection.mutable.ArrayBuffer // Para arrays variables

import java.io.PrintWriter // Para imprimir en fichero

import scala.io.Source // Para lectura desde fichero

object pruebaScala {

def main(args: Array[String]) {

// Cuerpo de la prueba concreta

}

}

Como Scala trabaja sobre la máquina virtual de Java, observarás que algunas bibliotecas se importan directamente de Java.

Dado el esqueleto anterior, en adelante solo indicaré las sentencias que debes probar dentro. Una vez introducidas las sentencias indicadas, deberás salvar el programa y ejecutarlo:

Run -> Run as -> Scala application

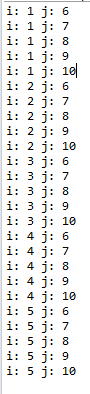
Introduce la siguiente sentencia y la ejecutas:

(0 to 10).foreach(println)

2.1.1 ¿Qué hace dicha sentencia? ¿Con qué sentencia más clásica harías lo mismo?

* + - * Imprime los números enteros del 0 al 10 con un salto de línea.
      * Con un una sentencia for e incluyendo println.

Introduce la siguiente sentencia y ejecuta el programa:



for (i <- 1 to 5; j <- 6 to 10) {

println("i: " + i + " j: " + j)

}

2.1.2 ¿Cómo avanzan los índices del bucle for?

* Itera **j** que va desde el numero entero 6 al 10 por cada iteración de i

El lenguaje Scala no tiene sentencias break o continue para salir de los bucles. En el programa siguiente se puede ver una solución alternativa para salir del bucle:

var i = 0

def imprimePrimos() {

val listaPrimos = List(1,2,3,5,7,11)

for (i <- listaPrimos) {

if (i == 11) {

return

}

if (i != 1) {

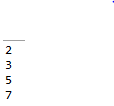
println(i)

}

}

}

imprimePrimos

2.1.3 Explica qué hace el programa y cómo se sale del bucle.

* Imprime los números PRIMOS entre el 2 y el 11 y sale del bucle simplemente al cumplirse la sentencia **return** de esta manera sale del bucle.

Introduce el siguiente trozo de código:

var opcion = 0

do {

print ("Teclea opción ")

opcion = readLine.toInt

} while (opcion!= 6)

printf ("Tecleaste la opción mágica %d\n", 6)

2.1.4 Modifica este programa para que salga cuando la opción tecleada sea "PCyA". Escribe a continuación el código modificado.

**var** opcion: String = "PCyA"

**do** {

print("Teclea opción ")

opcion = readLine.toString

} **while** (opcion != "PCyA")

printf("Tecleaste la opción mágica %s \n", "PCyA")

Introduce el siguiente programa:

def obtenerSuma(args: Int \*) : Int = {

var suma : Int = 0

for (num <- args) {

suma += num

}

suma

}

println ("Suma obtenida " + obtenerSuma(1,2,3,4,5,6))

2.1.5 Explica qué salida genera el programa al ejecutarse.

* El programa genera la suma de todos los argumentos de tipo entero pasados en la función, como una “lista” y suma todos sus elementos de uno en uno, es decir 1+2= 3, 3+3=6, 6+4=10, 10 +5 = 15, 15+6 =21, suma = 21. Siendo este el resultado de la salida.

A continuación se muestra el código para obtener la suma anterior utilizando el operador fold (/:)

**def** obtenerSuma(args: Int \*) : Int = {

(0/:args) (\_+\_)

}

2.1.6 Explica a continuación el funcionamiento de este código, indicando cuál es el valor inicial, la lista de valores y el operador binario.

Simplifica el bucle for sumando los argumentos que están siendo pasados y suma a partir del valor inicial 0 pasado como primer parámetro en el método **fold**:**(0/:args)**,

Es decir, suma el 0 + 1 = 1, 1+2= 3, 3+3=6, 6+4=10, 10 +5 = 15, 15+6 =21. Siendo este el resultado de la salida.

**Método fold**

* El método **fold** utiliza una función de operador binario asociativo como parámetro, y la usará para colapsar elementos de la colección. El método de **fold** también le permite especificar un **valor** **inicial** según la documentación de Scala.
* Lista de valores obtenerSuma(1,2,3,4,5,6)

**Apoyo Bibliográfico**

Métodos pasos fold [enlace](http://allaboutscala.com/tutorials/chapter-8-beginner-tutorial-using-scala-collection-functions/scala-fold-example/)

Paso de argumentos en funciones [enlace](https://alvinalexander.com/scala/how-to-define-methods-variable-arguments-varargs-fields)