Proyecto final INF-319 Programación funcional

INF-319 Functional Programming final project

Daniel Horacio Fernandez Garcia

dfernandezg@fcpn.edu.bo

Universidad Mayor de San Andrés, Facultad de Ciencias Puras y Naturales, Carrera de Informática

Resumen

En el siguiente artículo se compara el código de cinco programas sencillos, cada uno escrito en cinco lenguajes diferentes, desde lenguajes no funcionales (Java y C#), lenguajes intermedios (Python) y lenguajes funcionales (Scala y F#). Primero se detalla el pseudocódigo y a continuación una comparación del código en cada lenguaje. En el código escrito en cada lenguaje se puede ver la diferencia en cada tipo de lenguaje, como en los lenguajes no funcionales la construcción de código es mucho más complicada que en los lenguajes funcionales diseñados con funciones de orden superior en mente. También se puede ver las particularidades de cada lenguaje, los métodos o distintas formas de trabajar entre lenguajes y la diferencia entre los más tradicionales y los lenguajes funcionales.

Palabras clave: programación funcional, benchmarking.

Abstract

The following article compares the code of five simple programs, each written in five different languages, ranging from non-functional languages (Java and C #), intermediate languages (Python), and functional languages (Scala and F #). The pseudocode is detailed first and then a comparison of the code in each language. In the code written in each language is possible to see the difference in each type of language, as in non-functional languages the construction of code is much more complicated than in functional languages designed with higher-order functions in mind. It's also possible to see the particularities of each language, the included methods and different ways to work between each language and the difference between the more traditional and functional languages.

Keywords; functional programming, benchmarking.

1. INTRODUCCIÓN

1.1 Programación funcional

La programación funcional o functional programming se centra en las funciones. En un programa funcional, todos los elementos pueden entenderse como funciones y el código puede ejecutarse mediante llamadas de función secuenciales. Por el contrario, no se asignan valores de forma independiente. Una función se imagina mejor como una variante especial de un subprograma. Esta es reutilizable y, a diferencia de un procedimiento, devuelve directamente un resultado.

Por supuesto, en muchos lenguajes de programación superiores hay funciones que se definen y después se aplican. Por esto, esta no es la característica especial de la programación funcional. Lo que hace que la estrategia funcional sea tan importante para la informática y a la vez tan versátil es el hecho de que las funciones dentro de este paradigma de programación pueden adoptar diferentes "formas": estas pueden enlazarseentre sí como los datos y utilizarse como parámetro y como resultado de la función. Este tratamiento especial de las funciones

permite a los programadores implementar y procesar tareas computacionales muy complejas (especialmente las de naturaleza simbólica).

La programación funcional ofrece un alto grado de abstracción, ya que está basada en el concepto matemático y el principio de función. Cuando se aplica de forma correcta, este tipo de programación crea un código muy preciso. A partir de tantas unidades pequeñas, reutilizables y altamente especializadas como sea posible, se crea un programa para la solución de una tarea sustancialmente mayor.

Desde un punto de vista de mantenimiento, lógico y estructural, la programación funcional sobresale cuando no hay historias con las que lidiar. Funciona particularmente bien cuando no se requieren límites o esos límites ya están predefinidos. Prospera en situaciones en las que el estado no es un factor y hay muy poca o ninguna participación con datos cambiantes.

La programación funcional proporciona ventajas como eficiencia, evaluación diferida, funciones anidadas, código libre de errores, programación paralela. En lenguaje simple, la programación funcional consiste en escribir la función que tiene declaraciones para ejecutar una tarea particular para la aplicación. Cada pequeña función hace su parte y solo su parte. La función se puede invocar y reutilizar fácilmente en cualquier momento. También ayuda a administrar el código y no es necesario escribir la misma cosa o declaraciones una y otra vez. Permite un código muy modular y limpio que funciona en conjunto en armonía.

1.2 Programación funcional v programación orientada a objetos

La programación orientada a objetos es un paradigma de programación en el que programa utilizando objetos para representar cosas sobre las que está programando (a veces cosas del mundo real). Estos objetos pueden ser estructuras de datos. Los objetos contienen datos sobre ellos en atributos. Los atributos de los objetos se manipulan mediante métodos o funciones que se le dan al objeto.

Quizás la única real diferencia entre la programación funcional y la programación orientada a objetos es el énfasis que cada una pone en su propósito: la programación funcional pone el foco en lo correcto y preciso de un programa, y de aquí su gran énfasis en evitar estado compartido, mutable, además de computaciones con efectos secundarios. En la programación orientada a objetos, el propósito es otro: es hacer que el código sea más fácil de razonar y comprender, por medio de abstraer o modelar el problema en constructos llamados clases. El foco está en la legibilidad.

El principal problema con la programación orientada a objetos es la capacidad de encapsular datos de personas externas. La encapsulación es la capacidad de ocultar variables dentro de la clase del acceso externo, lo que lo hace excelente por razones de seguridad, junto con el uso accidental, no deseado o con fugas. La mayoría de los programadores que utilizan el diseño orientado a objetos dicen que es un estilo de programación que le permite modelar escenarios del mundo real mucho más simple. Esto permite una buena transición de los requisitos al código que funciona como lo desea el cliente o el usuario.

Uno de los primeros conceptos que se aprenden en programación funcional es la idea de las funciones puras. Una función pura es una función cuyo output está determinado por su input, sin efectos secundarios observables.

Los lenguajes funcionales representan y necesitan una forma distinta de abordar o pensar un problema, aún más si uno está acostumbrado a un paradigma más tradicional como la orientación a objetos. En algunas cosas pueden dar opciones parecidas a un lenguaje orientado a objetos, pero aún así el enfoque tiene que ser distinto.

2. MATERIALES Y MÉTODOS/METODOLOGÍA

Se tienen cinco problemas básicos para programar, cada problema debe ser escrito en cinco lenguajes distintos, variando desde lenguajes no funcionales hasta lenguajes funcionales. Se utilizó Visual Studio Code como editor de código. Se utilizó django para crear la interfaz web del código escrito en Python, Apache Tomcat como servidor y archivos .jsp para la interfaz web en Java, ASP.NET para el código de C# y para F# y Scala se

utilizaron aplicaciones de consola. Se intentó escribir código parecido en los distintos lenguajes, exceptuando funciones, métodos puntuales o diferencias de sintaxis de cada lenguaje.

2.1 Serie Fibonacci

2.1.1 Pseudocódigo

```
funcion fibo(n):
        a = 0
        b = 1
        for i <- 0 to n:
                res += ", "
                aux = a
a = b
                b = aux+a
                print(a)
        return res
```

2.1.2 Algoritmo

Tabla comparativa del código escrito entre los distintos lenguajes:

Tabla 1. Código en lenguajes no funcionales e intermedio de la serie fibonacci

	Python	java	c#

Función	def func_fibo(n):
	a = 0
	b = 1
	res = "0"
	for i in range(n):
	res += ", "
	aux = a
	a = b
	b = aux+a
	res += str(a)
	return res
		int a, b, lim, aux;
		lim = Integer.parseInt(request.getParameter("n_fibo"));
		a = 0;
		b = 1;
		out.print("0, ");
		for (int i = 2; i <= lim; i++)
		{
		aux = a;
		a = b;
		b = aux + a;
		out.print(a + ", ");
		
		if (i % 10 == 0 && i != 0) out.print("
		");
		}
			int a, b, lim, aux;
			lim = int.Parse(TextBox1.Text);
			a = 0;
			b = 1;
			String res = "0, ";
			for (int i = 2; i <= lim; i++)
			{
			aux = a;
			a = b;
			b = aux + a;
			res += (a.ToString() + ", ");
			if (i % 10 == 0 && i != 0)
			res += "\n";
			}
			TextBox2.Text = res;

Tabla 2. Código en lenguajes funcionales de la serie fibonacci

	F#	scala

Función	let n = System.Int32.Parse(Console.ReadLine())
	let mutable a = 0
	let mutable b = 1
	let mutable aux = 0
	printf "0, "
	for i in 2..n do
	aux <- a
	a <- b
	b <- aux + a
	printf "%i, " a
		val n = scala.io.StdIn.readInt()
		var a = 0
		var b = 1
		var aux = 0
		print("0, ")
		for(i <- 2 to n){
		aux = a
		a = b
		b = aux + a
		print(a+", ")
		}

2.2 Calculadora con funciones de orden superior

2.2.1 Pseudocódigo

```
funcion suma(x,y): return x + y
funcion resta(x,y): return x – y
funcion mult(x,y): return x * y
funcion div(x,y): return x / y

funcion calculadora(a,b,funcion):
    return funcion(a,b)
```

2.2.2 Algoritmo

Tabla 3. Código lenguajes no funcionales de calculadora

	python	java	c#

Operaciones de calculadora	def suma(x,y): return x + y
	def resta(x,y): return x - y
	def mult(x,y): return x * y
	def div(x,y): return x / y
		public int suma(int a, int b) {
		return a+b;
		}
		public int resta(int a, int b) {
		return a-b;
		}
		public int multi(int a, int b) {
		return a*b;
		}
		public int divi(int a, int b) {
		return a/b;
		}
			public int suma(int number1, int number2){
			return number1 + number2;
			}
			public int resta(int number1, int number2)
			{
			return number1 - number2;
			}
			public int mult(int number1, int number2)
			{
			return number1 * number2;
			}
			public int divi(int number1, int number2)
			{
			return number1 / number2;

Funcion de orden superior de calculadora	def func_calcu(a,b,func):
	res = func(a,b)
	return str(res)
		public interface Callable {
		public int suma(int a, int b);
		public int resta(int a, int b);
		public int multi(int a, int b);
		public int divi(int a, int b);
		public int[][] sumamat(int[][] a, int[][] b);
		public int[][] restamat(int[][] a, int[][] b);
		public int[][] multimat(int[][] a, int[][] b);
		}
		
		public static int invoke(Callable callable, int a, int b, String op){
		if (op.equals("suma")) return callable.suma(a,b);
		else if (op.equals("resta")) return callable.resta(a,b);
			public int operación(Func<int,int,int> op, int num1,int num2) {
			return op(num1, num2);
			}

		else if (op.equals("multi")) return callable.multi(a,b);
		return callable.divi(a,b);
		}	

Control función orden superior	def inter(a,b,op):
	if (op == 'suma'):
	return func_calcu(a,b,suma)
	elif (op == 'resta'):
	return func_calcu(a,b,resta)
	elif (op == 'mult'):
	return func_calcu(a,b,mult)
	elif (op == 'div'):
	return func_calcu(a,b,div)
	else:
	return "Operacion no reconocida"
		out.print(op+"
		");
		Callable cmd = new Calcu();
		int res = invoke(cmd, a, b, op);
		out.print(res);
			***Cada operación tiene un botón que llama a la operación
			necesaria

Tabla 4. Código lenguajes funcionales de calculadora

	f#	scala
Operaciones de calculadora	let suma a b = a + b let resta a b = a - b let multi a b = a * b let divi a b = a / b	def suma(a:Int,b:Int):Int=a+b def resta(a:Int,b:Int):Int=a-b def mult(a:Int,b:Int):Int=a*b def div(a:Int,b:Int):Int=a/b def mod(a:Int,b:Int):Int=a%b
Funcion orden superior	let calcu a b op = op a b	def xay(x:Int, y:Int, funcion:(Int, Int) => Int):Int = funcion(x,y)

Control función orden superior	if op = "suma" then
	let res = calcu a b suma
	printfn "Resultado:%i " res
	else if op = "resta" then
	let res = calcu a b resta
	printfn "Resultado:%i " res
	else if op = "multi" then
	let res = calcu a b multi
	printfn "Resultado: %i " res
	else if op = "divi" then
	let res = calcu a b divi
	printfn "Resultado:%i " res
	else do printf"\nOperacion no reconocida"
		def calcu(x:Int, y:Int, operacion:String):Int={
		operacion match {
		case "suma"=>xay(x,y,suma)
		case "resta"=>xay(x,y,resta)
		case "mult"=>xay(x,y,mult)
		case "div"=>xay(x,y,div)
		}
		}

2.3 Calculadora de matrices con funciones de orden superior

2.3.1 Pseudocódigo

```
funcion sumat(m1, m2):
  res = [2,2]
  for i <- 0 to 2:
    for j <- 0 to 2:
```

```
            res[i][j] = m1[i][j] + m2[i][j]
        return res

    funcion resmatri(m1, m2):
        res = [2,2]
        for i <- 0 to 2:
            for j <- 0 to (2:
                res[i][j] = m1[i][j] - m2[i][j]
        return res

    funcion mulmat(m1, m2):
        res = [2,2]
        for i <- 0 to 2:
            for j <- 0 to 2:
                for k <- 0 to 2:
                    res[i][j] += m1[i][k] * m2[k][j]
        return res

    funcion func_mat(a,b,func):
        return func(a,b)
```

2.3.2 Algoritmo

Tabla 5. Código lenguajes no funcionales de calculadora de matrices

	python	java	c#

Operaciones calculadora de matrices	def sumat(m1, m2):
	res = [[0,0],[0,0]]
	cad = ""
	for i in range(2):
	for j in range(2):
	res[i][j] = m1[i][j] + m2[i][j]
	cad += str(res[i][j]) + " "
	cad+="
	"
	return cad
	
	def resmatri(m1, m2):
	res = [[0,0],[0,0]]
	cad = ""
	for i in range(2):
	for j in range(2):
	res[i][j] = m1[i][j] - m2[i][j]
	cad += str(res[i][j]) + " "
	cad+="
	"
	return cad
	
	def mulmat(m1, m2):
	res = [[0,0],[0,0]]
		public int[][] sumamat(int[][] mat1, int[][] mat2)
		{
		int[][] res = new int[2][2];
		for (int i = 0; i < 2; i++)
		{
		for (int j = 0; j < 2; j++)
		{
		res[i][j] = mat1[i][j] + mat2[i][j];
		}
		}
		return res;
		}
		public int[][] restamat(int[][] mat1, int[][] mat2)
		{
		int[][] res = new int[2][2];
		for (int i = 0; i < 2; i++)
		{
		for (int j = 0; j < 2; j++)
		{
		res[i][j] = mat1[i][j] - mat2[i][j];
		}
		}
			public int[,] sumamat(int[,] mat1, int[,] mat2)
			{
			int[,] res = new int[2, 2];
			for (int i = 0; i < 2; i++)
			{
			for (int j = 0; j < 2; j++)
			{
			res[i, j] = mat1[i, j] + mat2[i, j];
			}
			}
			return res;
			}
			
			public int[,] resmat(int[,] mat1, int[,] mat2)
			{
			int[,] res = new int[2, 2];
			for (int i = 0; i < 2; i++)
			{
			for (int j = 0; j < 2; j++)
			{
			res[i, j] = mat1[i, j] - mat2[i, j];
			}

	cad = ""
	for i in range(2):
	for j in range(2):
	for k in range(2):
	res[i][j] += m1[i][k] * m2[k][j]
	cad += str(res[i][j]) + " "
	cad+="
	"
	return cad
		return res;
		}
		public int[][] multimat(int[][] mat1, int[][] mat2)
		{
		int[][] res = new int[2][2];
		int n = 2;
		for (int i = 0; i < n; i++)
		{
		for (int j = 0; j < n; j++)
		{
		res[i][j] = 0;
		for (int k = 0; k < n; k++)
		{
		res[i][j] += mat1[i][k] * mat2[k][j];
		}
		}
		}
		return res;
		}
			}
			return res;
			}
			public int[,] mulmat(int[,] mat1, int[,] mat2)
			{
			int[,] res = new int[2, 2];
			int n = 2;
			for (int i = 0; i < n; i++)
			{
			for (int j = 0; j < n; j++)
			{
			res[i,j] = 0;
			for (int k = 0; k < n; k++)
			{
			res[i,j] += mat1[i, k] * mat2[k, j];
			}
			}
			}
			return res;
			}

Funcion orden superior y control	def interm(a,b,op):
	if (op == 'suma'):
	return func_mat(a,b,sumat)
	elif (op == 'resta'):
	return func_mat(a,b,resmatri)
	elif (op == 'multi'):
	return func_mat(a,b,mulmat)
	else:
	return "Operacion no reconocida"
	
	def func_mat(a,b,func):
	return func(a,b)
		public static int[][] mat(Callable callable, int[][] a, int[][] b, String op){
		if (op.equals("suma")) return callable.sumamat(a,b);
		else if (op.equals("resta")) return callable.restamat(a,b);
		return callable.multimat(a,b);
		}
		
		Callable cmd = new Calcu();
		int[][] res = mat(cmd, m1, m2, op);
			public int[,] matrices(Func<int[,], int[,], int[,]> op, int[,] mat1, int[,] mat2)
			{
			return op(mat1, mat2);
			}

Tabla 6. Código lenguajes funcionales de calculadora de matrices

	f#	scala

Operaciones calculadora de matrices	let sumat (a:int[,]) (b:int[,]) (c:int[,]) =
	for i = 0 to 1 do
	for j = 0 to 1 do c.[i,j] <- a.[i,j] + b.[i,j]
		def suma(a:Array[Array[Int]],b:Array[Array[Int]]) {
		var res = ofDim[Int](2,2)
		println("resultado:")
		for (i <- 0 to 1) {

	let resmat (a:int[,]) (b:int[,]) (c:int[,]) =
	for i = 0 to 1 do
	for j = 0 to 1 do c.[i,j] <- a.[i,j] - b.[i,j]
	
	let mulmat (a:int[,]) (b:int[,]) (c:int[,]) =
	for i = 0 to 1 do
	for j = 0 to 1 do
	c.[i,j] <- 0
	for k = 0 to 1 do c.[i,j] <- c.[i,j] + a.[i,k] * b.[k,j]
		for (j <- 0 to 1) {
		res(i)(j) = a(i)(j) + b(i)(j)
		print(" " + res(i)(j))
		}
		println()
		}
		}
		
		def resta(a:Array[Array[Int]],b:Array[Array[Int]]) {
		var res = ofDim[Int](2,2)
		println("resultado:")
		for (i <- 0 to 1) {
		for (j <- 0 to 1) {
		res(i)(j) = a(i)(j) - b(i)(j)
		print(" " + res(i)(j))
		}
		println()
		}
		}
		
		def multi(a:Array[Array[Int]],b:Array[Array[Int]]) {
		var res = ofDim[Int](2,2)
		println("resultado:")
		for (i <- 0 to 1) {
		for (j <- 0 to 1) {
		res(i)(j) = 0
		for(k <- 0 to 1) {
		res(i)(j) += a(i)(k) * b(k)(j)
		}
		print(" " + res(i)(j))
		}
		println()
		}
		}

Funcion orden superior y control	let matrices a b c op = op a b c
	
	if op = "suma" then
	matrices mat1 mat2 c sumat
	else if op = "resta" then
	matrices mat1 mat2 c resmat
	else if op = "multi" then
	matrices mat1 mat2 c mulmat
		def xay(x:Array[Array[Int]], y:Array[Array[Int]], funcion:(Array[Array[Int]], Array[Array[Int]]) => Unit):Unit = funcion(x,y)
		
		def calcu(x:Array[Array[Int]], y:Array[Array[Int]], operacion:String){
		operacion match {
		case "suma"=>xay(x,y,suma)
		case "resta"=>xay(x,y,resta)
		case "multi"=>xay(x,y,multi)
		}

	else do printf"\nOperacion no reconocida"	}

2.4 Criba de eratostenes

2.4.1 Pseudocódigo

```
funcion criba(n):
   arr=[n+1]
   for i <- 0 to sqrt(n):
      if(arr[i] == 0):
         j=2
         while(i*j < n):
            arr[i*j] = 1
            j+=1

   for i <- 2 to n:
      if(arr[i]==0):
         print(i)
```

2.4.2 Algoritmo

Tabla 7. Código lenguajes no funcionales de la criba de eratostenes

	python	java	c#

Criba de eratostenes	def func_criba(n):
	import math
	cad=""
	arr=[]
	arr = [0 for i in range(n+1)]
	for i in range(2,math.trunc(math.sqrt(n+1))):
	if(arr[i] == 0):
	j=2
	while(i*j < n):
	arr[i*j] = 1
	j+=1
	for i in range(2,n):
	if(arr[i]==0):
	cad+= str(i) + ", "
	return cad
		int n = Integer.parseInt(request.getParameter("n_criba"));
		Boolean[] marcados = new Boolean[n + 1];
		for (int i = 2; i < marcados.length; i++){
		marcados[i] = false;
		}
		for (int i = 2; i <= Math.sqrt(n); i++)
		{
		if (!marcados[i]){
		for (int j = i * 2; j <= n; j += i)
		marcados[j] = true;
		}
		}
		for (int i = 2; i < marcados.length; i++){
		if (marcados[i] == false)
		{
		out.print(i+", ");
		if (i % 10 == 0) out.print("
		");
		}
		}
			int n = int.Parse(TextBox18.Text);
			bool[] marcados = new bool[n + 1];
			String res = "";
			int co = 0;
			for (int i = 2; i <= Math.Sqrt(n); i++)
			{
			if (!marcados[i])
			for (int j = i * 2; j <= n; j += i)
			marcados[j] = true;
			}
			for (int i = 2; i < marcados.Length; i++)
			if (marcados[i] == false)
			{
			res += i.ToString() + ", ";
			co++;
			if (co % 10 == 0) res += "\n";
			}
			TextBox19.Text = res;

Tabla 8. Código lenguajes funcionales de la criba de eratostenes

	f#	scala

Criba de eratostenes	let max = System.Int32.Parse(Console.ReadLine())
	let array = new BitArray(max, true);
	let lastp = Math.Sqrt(float max) |> int
	for p in 2..lastp+1 do
	if array.Get(p) then
	for pm in p*2..p..max-1 do
	array.Set(pm, false);
	let primos = [for i in 2..max-1 do if array.Get(i) then yield i]
	
	printfn "\n primos"
	for i in primos do printf "%i " i
		val n = scala.io.StdIn.readInt()
		var marcados = new Array[Boolean](n+1)
		for(i <- 2 to Math.round(scala.math.sqrt(n).floatValue)){
		if(!marcados(i)){
		var j = i*2
		while(j <= n){
		marcados(j) = true
		
		j+=i
		}
		}
		}
		
		for(i <- 2 until n){
		if(!marcados(i)){
		print(i + " ")
		}
		}

2.5 Separar lista en pares e impares

2.5.1 Pseudocódigo

```
funcion separar(s):
    arr = string.toArrayInt(s)
    pares = []
    impares = []
    for x in arr:
        if(x % 2 == 0):
            pares.append(x)
        else:
            impares. append(x)

for i in pares:
    print(pares[i])

for i in pares:
    print(impares[i])
```

2.5.2 Algoritmo

Tabla 9. Código lenguajes no funcionales para separar una lista

	python	java	c#
Separar lista en pares e impares	def func_separar(s): arr = [int(x) for x in s.split()] pares = ""	String[] cad = request.getParameter("lista").trim().split(" "); int[] lista = new int[cad.length]; String pares ="";	String[] cad = (TextBox20.Text).Split(' '); int[] nums = Array.ConvertAll(cad, int.Parse); String pares ="";

	impares = ""
	for x in arr:
	if(x % 2 == 0):
	pares += str(x) +", "
	else:
	impares += str(x) +", "
	return "Pares:
	" + pares + "
	
	Impares:
	" + impares
		String impares ="";
		int cop=0;
		int coi=0;
		
		for(int i=0; i<lista.length; i++){
		lista[i]=Integer.parseInt(cad[i]);
		
		if (lista[i] % 2 == 0) pares += lista[i] + ", ";
		else impares += lista[i] + ", ";
		}
		
		out.println(" Pares ");
		out.println(pares);
		out.println(" Impares ");
		out.println(impares);
			String impares ="";
			int cop=0;
			int coi=0;
			for (int i = 0; i < nums.Length; i++)
			{
			if (nums[i] % 2 == 0)
			
			{
			pares += nums[i] + ", ";
			cop++;
			if (cop % 10 == 0) pares += "\n";
			}
			else {
			impares += nums[i] + ", ";
			coi++;
			if (coi % 10 == 0) impares += "\n";
			}
			}
			TextBox21.Text = pares;
			TextBox22.Text = impares;

Tabla 10. Código lenguajes funcionales para separar una lista

	f#	scala

Separar lista en pares e impares	let n = System.Int32.Parse(Console.ReadLine())
	let a () =
	printf "numero:"
	System.Int32.Parse(Console.ReadLine())
	
	let lista = [for i in 1 .. n -> a ()]
	let pares = [for i in lista do if i%2=0 then yield i]
	let impares = [for i in lista do if i%2<>0 then yield i]
	
	printfn "\n pares"
	for i in pares do printf "%i " i
	
	printfn "\n impares"
	for i in impares do printf "%i " i
		val n = scala.io.StdIn.readInt()
		var lista = new Array[Int](n)
		for(i <- 0 until n){
		print("["+i+"]:")
		lista(i) = scala.io.StdIn.readInt()
		}
		
		var pares = for (i <- lista if (i % 2 == 0)) yield i
		var impares = for (i <- lista if (i % 2 != 0)) yield i
		
		println("Pares:")
		println(pares.mkString(" "))
		println("Impares:")
		println(impares.mkString(" "))

3. RESULTADOS Y DISCUSIÓN

En el código escrito en cada problema se puede apreciar, sobre todo en las funciones de orden superior, se nota la diferencia entre lenguajes funcionales y no funcionales. En los lenguajes funcionales es posible declarar las funciones de orden superior con muy pocas líneas y de forma muy sencilla. En Python y C# también es relativamente sencillo declarar y utilizar funciones de orden superior.

En cambio, en Java la declaración de funciones de orden superior es más complicada, necesitando incluir una interface, una clase que herede esa interface y un tercer método para invocar a esa clase, se nota la diferencia y que Java no está construido ni pensado para programación funcional.

Por otro lado, se intentó escribir un código parecido en cada lenguaje, exceptuando algunas funciones incluidas de cada lenguaje, como por ejemplo, la función yield en F# y Scala que para el problema separar listas, permite resolver el problema con mucho menos código que en el resto de lenguajes.

4. CONCLUSIONES

La programación funcional es muy diferente a la programación orientada a objetos, y por lo tanto necesita un enfoque e incluso una manera de pensar distinta a la hora de resolver un problema. En este trabajo se intentó tener un código parecido entre todos los lenguajes y por lo tanto no se aprecia totalmente la diferencia y las ventajas o desventajas de la programación funcional, excepto en situaciones puntuales o puntos fuertes claros de la programación funcional, como la declaración y utilización de funciones de orden superior.

REFERENCIAS

Programación funcional VS Programación orientada a objetos (OOP) ¿Cuál es mejor. . .? (2020, 3 diciembre). ICHI.PRO. Recuperado 13 de diciembre de 2021, de https://ichi.pro/es/programacion-funcional-vs-programacion-orientada-a-objetos-oop-cual-es-mejor-143036073092390

Programación funcional: ideal para algoritmos. (2021, 3 diciembre). IONOS Digitalguide. Recuperado 13 de diciembre de 2021, de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/programacion-funcional/

Carter, M. N. (2021, 12 diciembre). POO VS PF: Una falsa dicotomía - Matías Navarro Carter. Medium. Recuperado 13 de diciembre de 2021, de https://medium.com/@mnavarrocarter/poo-vs-pf-una-falsa-dicotom%C3%ADa-c3a0f52e68a7