Programación imperativa y Funcional

Imperative and Functional programming

Vargas Ramírez Leonel Maximo
lvargasr@@fcpn.edu.bo

Universidad Mayor de San Andrés, Facultad de Informática, Programación Funcional INF - 319] (1)

Resumen

El presente artículo se realiza una comparativa funcional entre lenguajes de programación que usan una programación imperativa con lenguajes de programación funcional. Realizando 5 algoritmos en 5 lenguajes de programación como ser Scala, Python, Java, C# y F#. Se quiere lograr estudiar, analizar y comparar la programación imperativa con la programación funcional, viendo así, sus ventajas y desventajas dentro de un contexto en específico.

Se verá que con la programación funcional, hay ciertas tareas que se ejecutan de forma más simple y sencilla, que la programación funcional. La sintaxis de un lenguaje funcional puede ser reducida drásticamente en ciertos casos, en comparación que otros lenguajes imperativos.

Los lenguajes de programación en su medida son más simples y están pensados en ser utilizados de la misma forma en la que actuamos, es por ello que la programación funcional, puede llevar un papel importante en realizar tareas como web servicies, ya que son fáciles y cortos de implementar.

Se obtuvo así un análisis y comparativa de 5 algoritmos que son Fibonacci4 (que es una variación del Fibonacci convencional, calculadora (de 2 números), calculadora de matrices (de dos matrices), factorial y el algoritmo para saber si un número es primo. Dichos algoritmos fueron comparados en sus versiones imperativas y funcionales. Observando así, la cantidad de líneas para su implementación, su facilidad, su accesibilidad y su tiempo de implementación.

Palabras Clave: Programación Funcional, Algoritmos, Comparación, Scala, F#.

Abstract

This article makes a functional comparison between programming languages that use imperative programming with functional programming languages. Performing 5 algorithms in 5 programming languages such as Scala, Python, Java, C # and F #. You want to be able to study, analyze and compare imperative programming with functional programming, thus seeing its advantages and disadvantages within a specific context.

It is seen that with functional programming, there are certain tasks that are executed in a simpler and easier way than functional programming. The syntax of a functional language can be drastically reduced in certain cases, compared to other imperative languages.

Programming languages in their measure are simpler and are designed to be used in the same way in which we act, that is why functional programming can play an important role in performing tasks such as web services, since they are easy and short to implement.

Thus, an analysis and comparison of 5 algorithms that are Fibonacci4 (which is a variation of the conventional Fibonacci, calculator (of 2 numbers), matrix calculator (of two matrices), factorial and the algorithm to know if a number is prime was obtained. Said algorithms were compared in their imperative and functional versions, thus observing the number of lines for their implementation, their ease, their accessibility and their implementation time.

Keywords: Functional Programming, Algorithms, Comparison, Scala, F #.

1. INTRODUCCIÓN

La programación imperativa ha surgido desde ya hace mucho tiempo, su surgimiento y popularización ha establecido un precedente en la informática a tal punto que al día de hoy la mayor parte de los lenguajes de programación centran su filosofía en este paradigma, tal es el caso de Java, C#, C, C++, etc. Sin embargo, enfocarnos y adentrarnos en otros paradigmas de programación también conlleva numerosas ventajas, tanto de rendimiento, como de lógica.

Es por tanto, que se quiere realizar una comparativa entre estos dos tipos de paradigmas de programación, tomando

como parámetros la cantidad de líneas de código codificadas, el rendimiento y su tiempo de implementación. Para poder ver así las ventajas y desventajas de un paradigma a otro. Y su aplicabilidad en diferentes áreas y contextos en específico.

Se va realizar la comparativa de cinco algoritmos implementados en los siguientes lenguajes de programación: Java, C#, F#, Python y Scala. Además de realizar una interfaz para la introducción de datos a nuestros programas. Entre los lenguajes de programación propuestos, Java y C# son puramente imperativos, Python y F# son híbridos de imperativos y funcionales y Scala es puramente funcional.

Presentaremos un marco referente a lo teórico sobre estos dos paradigmas, luego se realizar un marco aplicativo, donde se detalla mediante un cuadro la comparativa de los diferentes algoritmos en los cinco lenguajes de programación. Luego, la sección de análisis de ventajas y desventajas que conlleva aplicar programación funcional. Y por último se dará las conclusiones a los resultados obtenidos por el presente artículo.

La revisión (junto con las referencias bibliográficas incluidas) debería indicarle claramente al lector que existe una brecha en el tema estudiado y que esa brecha justifica la realización del trabajo presentado en el artículo.

2. MARCO TEORICO

2.1 Programación Imperativa

La programación imperativa (del latín imperare = ordenar) es el paradigma de programación más antiguo. De acuerdo con este paradigma, un programa consiste en una secuencia claramente definida de instrucciones para un ordenador.

El código fuente de los lenguajes imperativos encadena instrucciones una detrás de otra que determinan lo que debe hacer el ordenador en cada momento para alcanzar un resultado deseado. Los valores utilizados en las variables se modifican durante la ejecución del programa. Para gestionar las instrucciones, se integran estructuras de control como bucles o estructuras anidadas en el código.

Los lenguajes de programación imperativa son muy concretos y trabajan cerca del sistema. De esta forma, el código es, por un lado, fácilmente comprensible, pero, por el otro, requiere muchas líneas de texto fuente para describir lo que en los lenguajes de la programación declarativa se consigue con solo una parte de las instrucciones.

Los lenguajes de programación imperativa más conocidos son: Fortran, Java, Pascal, ALGOL, C, C#, C++, Basic, COBOL, Python, Ruby, Ensamblador.

2.2 Programación Funcional

Es un lenguaje de programación declarativo donde el programador especifica lo que quiere hacer, en lugar de lidiar con el estado de los objetos. Es decir, las funciones estarían en un primer lugar y nos centraremos en expresiones que pueden ser asignadas a cualquier variable.

En un código de forma declarativa se busca que no juegue con los objetos y sea más legible. Lo normal sería que un desarrollador tuviera que hacer un bucle, iterando y crear una lógica pero, con el lenguaje funcional, este te da funciones que hace que parezca más a leer y escribir que a programar.

El origen del modelo de programación funcional, pese a ser algo de relativa reciente aceptación, tiene su origen en el cálculo lambda. El cálculo lambda es un sistema desarrollado en la década de los 30 del siglo XX, donde buscaban investigar la naturaleza de las funciones y la computabilidad.

En la actualidad contamos con un buena cantidad de lenguajes funcionales, los cuales son los siguientes: LISP, ML, Haskell, OCaml, F#, Erlang, Clojure y Scala.

Además, existen muchos lenguajes de programación conocidos con los que podemos aplicar modelos de

programación funcional entre sus paradigmas. Estos son: Perl, Ruby, Visual Basic .NET, Dyln, Javascript, Python.

3. MARCO APLICATIVO

3.1 ALGORITMOS PROPUESTOS

Para el desarrollo del siguiente artículo se ha propuesto cinco algoritmos, los cuales se presenta a continuación:

- Fibonacci 4: Es una variación del algoritmo de Fibonacci convencional que considera para su construcción las primeras 4 posiciones de la serie de Fibonacci, establecidas a 1, y el quinto elemento de la serie es la suma de las 4 posiciones anteriores. Es decir:
  1, 1, 1, 1, 4, 7, 13, 25, 49, 94, … , etc.

- Calculadora: El algoritmo de una calculadora de 2 números como argumentos de los operadores (+, -, *, /) un algoritmo bastante simple, pero que lo complicaremos con su implementación a través de funciones de orden superior (característica de la programación funcional)

- Calculadora de Matrices: Este algoritmo implementará las operaciones convencionales y permitidas con matrices de dimensiones (nxn), como ser: la suma de matrices, la multiplicación de matrices y la resta de matrices. También se lo implementará con funciones de orden superior.

- Factorial: Algoritmo muy utilizado como base de ejemplo, en cursos de programación y que nos servirá para adentrarnos a la sintaxis y formalidad, del paradigma de programación funcional.

- Algoritmo para saber si un número es primo: El algoritmo para determinar si un determinado número es primo, también nos ayudará a realizar este cambio de paradigma de programación.

3.2 TABLA COMPARATIVA DE LINEAS DE CODIGO

A continuación se muestra la tabla de la comparativa de los 5 algoritmos implementados en los diferentes lenguajes de programación imperativos, híbridos o funcionales.

	Java	Python	C#

Algoritmo de Fibonacci 4	public int[] fibonacci4(int n) {
	int[] fibo = new int[n+1];
	fibo[0] = 1;
	fibo[1] = 1;
	fibo[2] = 1;
	fibo[3] = 1;
	for(int i = 4; i < n; i++) {
	fibo[i] = fibo[i- 1] + fibo[i - 2] + fibo[i - 3] + fibo[i - 4];
	}
	return fibo;
	}
		def fibonacci_4(n):
		fibo = []
		fibo.append(1)
		fibo.append(1)
		fibo.append(1)
		fibo.append(1)
		for i in range(4, n):
		fibo.append(fibo[i - 1] + fibo[i - 2] + fibo[i - 3] + fibo[i - 4])
		return fibo
			public int[] fibonacci4(int n)
			{
			int[] fibo = new int[n];
			fibo[0] = 1;
			fibo[1] = 1;
			fibo[2] = 1;
			fibo[3] = 1;
			for (int i = 4; i < n; i++)
			{
			fibo[i] = fibo[i - 1] + fibo[i - 2] + fibo[i - 3] + fibo[i - 4];
			}
			
			return fibo;
			}

Algoritmo de la Calculadora

```
public int calculadora(int num1, int num2, String op) {
                switch(op) {
                case "suma":
                    return suma(num1, num2);
                case "resta":
                    return resta(num1, num2);
                case "multiplicacion":
                    return multiplicacion(num1, num2);
                case "division":
                    return division(num1, num2);
                }
                return -1;
            }
```

```
def calculadora(x, y, fun):

    return eval(fun)(x,y)
```

```
public int calculadora(int num1, int num2, String op)
        {
            switch (op)
            {
                case "suma":
                    return suma(num1, num2);
                case "resta":
                    return resta(num1, num2);
                case "multiplicacion":
                    return multiplicacion(num1, num2);
                case "division":
                    return division(num1, num2);
            }
```

Algoritmo de la calculadora De Matrices

```
public int[][] multiplicaMatriz(int[][] a, int[][] b, int[][] r) {
        int aux = 0;
        for(int i = 0; i < a.length; i++) {
            for(int k = 0; k < b[0].length; k++) { // columnas (2)
                for(int j = 0; j < a[0].length; j++) { // filas (3)
                    aux = aux + (a[i][j] * b[j][k]);
                }
                r[i][k] = aux;
                aux = 0;
            }
        }
        return r;
}
```

```
def multiplicacion(ma1, ma2):
    aux = 0
    res = []
    for i, v in enumerate(ma1): #2
        for k, z in enumerate(ma2[0]):
            for j, x in enumerate(ma2):
                aux = aux + (ma1[i][j] * ma2[j][k])
            res.append(aux)
            aux = 0
    return res
```

```
public int[,] multiplicaMatriz(int[,] a, int[,] b, int[,] r, int n, int m, int m2)
        {
            int aux = 0;
            for (int i = 0; i < n; i++)
            { // general
                for (int k = 0; k < m2; k++)
                { // columnas (2)
                    for (int j = 0; j < m; j++)
                    { // filas (3)
                        aux = aux + (a[i,j] * b[j,k]);
                    }
                    r[i,k] = aux;
                    aux = 0;
                }
            }
            return r;       }
```

Algoritmo del Factorial

```
public static int factorial(int x) {
                int fac = 1;
                for(int i = 1; i <= x; i++)
                    fac = fac * i;
                return fac;
}
```

```
def factorial(num):
    fac = 1
    for i in range(1, num+1):
        fac = fac * i
    return fac
```

```
public int factorial(int n)
        {
            if (n == 1 || n == 0)
return 1;
            else return n * factorial(n - 1);
        }
```

Algoritmo para Saber si un número es primo.

```
public static boolean esPrimo(int p) {
                int cnt = 0;
                for(int i = 1; i <= p; i++) {
                    if(p % i == 0) {
                        cnt++;
                    }
                }
                if(cnt <= 2) return true;
                else return false;}
```

```
def esPrimo(num):
    cnt = 0
    for i in range(1, num + 1):
        if num % i == 0:
            cnt += 1
    if cnt <= 2:
        return "SI"
    else:
        return "NO"
```

```
public bool esPrimo(int p)
        {
            int cnt = 0;
            for (int i = 1; i <= p; i++)
            {
                if (p % i == 0) cnt++;
            }
            if (cnt <= 2) return true;
            else return false;
        }
```

	F#	Scala
Algoritmo de Fibonacci 4	let rec fibonacci4(n:int) = let fibo : int array = Array.zeroCreate n Array.set fibo 0 1 Array.set fibo 1 1 Array.set fibo 2 1 Array.set fibo 3 1	def fibonacci4(n:Int):Array[Int] = { var fibo:Array[Int] = new Array[Int](n) fibo(0) = 1 fibo(1) = 1 fibo(2) = 1 fibo(3) = 1

	for i in 4 .. fibo.Length - 1 do
	Array.set fibo i (Array.get fibo (i - 1) + Array.get fibo (i - 2) + Array.get fibo (i - 3) + Array.get fibo (i - 4))
	fibo
		for(i <- (4 until n)) {
		fibo(i) = fibo(i - 1) + fibo(i - 2) + fibo (i - 3) + fibo(i - 4)
		}
		fibo
		}

Algoritmo de la calculadora	let rec calculadora(x:int, y:int, op:string) = match op with | "suma" -> suma(x,y) | "resta" -> resta(x,y) | "multiplicacion" -> multiplicacion(x,y) | "division" -> division(x,y) | _ -> -1	def calculadora(x:Int, y:Int, funcion:(Int, Int) => Int):Int = funcion(x,y)

Algoritmo de la calculadora de matrices	let multiplicacionM (x:int[,], y:int[,]) =
	let res = Array2D.create 2 2 0
	let mutable aux = 0 // asi se declara una varible mutable
	for i = 0 to 1 do
	for k = 0 to 1 do
	for j = 0 to 2 do
	aux <- aux + (x.[i,j] * y.[j,k])
	res.[i,k] <- aux // asi se asigna, el = solo se usa en las declaraciones
	aux <- 0
	res
		a match {
		case "multiplicacion" =>
		var aux = 0
		for (i <- 0 to 1) {
		for(k <- 0 to 1) {
		for(j <- 0 to 2) {
		aux = aux + (matrix1(i)(j) * matrix2(j)(k))
		}
		res(i)(k) = aux
		aux = 0
		}
		}
		}

Algoritmo de Factorial	let rec factorial n = match n with | 0 -> 1 | _ -> n * factorial(n - 1)	def factorial(x: Int): BigInt = if (x == 0) 1 else x * factorial (x - 1)

Algoritmo para saber si un número es primo	let rec divs (p:int, cnt:int, i:int) =
	if p < i then cnt
	elif p % i = 0 then divs(p, (cnt + 1), (i + 1))
	else divs(p, cnt, (i + 1))
	
	let rec esPrimo(p:int) =
	if divs(p, 0, 1) <= 2 then "SI"
	else "NO"
		def divs (n:Int, i:Int, cnt:Int):Int = {
		if (n < i) cnt
		else if (n % i == 0) divs(n, (i+1), (cnt+1))
		else divs(n, (i+1), cnt)
		}
		
		def esPrimo(x:Int):String = {
		if (divs(x,1,0) <= 2) "Si"
		else "No"
		}

3.3 VENTAJAS Y DESVENTAJAS DE AMBOS PARADIGMAS DE PROGRAMACIÓN

Como se pudo ver el cambio de un paradigma a otro puede ser un poco agobiante para el programador que viene de un paradigma de programación imperativa. Resulta útil a veces preguntarse, si dicho esfuerzo vale la pena. Es por ello que a continuación presentamos las numerosas ventajas y desventajas que tiene tanto el paradigma de programación imperativa, como el paradigma de programación funcional.

PROGRAMACIÓN IMPERATIVA

VENTAJAS

- Estos lenguajes son comparativamente más fáciles de aprender, ya que el código se lee como unas instrucciones paso a paso. Por esta razón, normalmente los programadores empiezan su formación con el aprendizaje de un lenguaje imperativo.
- La buena legibilidad es un factor decisivo en el día a día laboral. Al fin y al cabo, el mantenimiento y la optimización de las aplicaciones no tienen por qué depender exclusivamente de una persona, sino que

los puede llevar a cabo cualquier empleado, sin necesidad de que este haya escrito el código desde cero.
- Se puede tener en cuenta características de casos especiales de la aplicación.

DESVENTAJAS

- El código se convierte rápidamente en demasiado amplio y difícil de abarcar.
- Mayor riesgo durante la edición
- El mantenimiento bloquea el desarrollo de la aplicación, ya que la programación funciona estrechamente con el sistema.
- La optimización y la ampliación son más difíciles.

PROGRAMACIÓN FUNCIONAL

Al igual que la programación imperativa, la programación funcional conlleva una serie de ventajas y desventajas. Las cuales se presenta a continuación:

VENTAJAS

- Los programas no cuentan con estados.
- Es una forma de programar muy adecuada para la paralelización.
- Es un código fácilmente testable y verificable, incluso en las funciones que no cuentan con estado.
- Se trata de un código más corto, sencillo, legible y preciso.
- Fácilmente combinable con la programación orientada a objetos e imperativa.

DESVENTAJAS

- Los datos no pueden modificarse, es decir, las variables.
- No es recomendable usarlo para conexiones de base de datos y/o servidores. Además, no cuenta con un acceso eficiente a grandes cantidades de datos.
- No es la mejor opción para recursiones de la misma pila.
- Podemos tener errores graves en la programación recurrente.

4. CONCLUSIONES

En definitiva, el paradigma de programación funcional puede ser muy útil en ciertos casos. Se ha visto que su implementación reduce de forma considerable la cantidad de líneas escritas en un programa. Está pensado para declarar órdenes y no decirle como debería hacer su trabajo. Se asemeja más a al pensamiento humano y te permite desarrollar una lógica distinta si vienes de programar de forma imperativa.

Con la programación imperativa se logra una mayor gestión de la memoria, ya que está no usa, variables si no que sólo transforma los datos a otro tipo de datos. Las funciones son las encargadas de realizar este trabajo, y se asemejan más a las funciones matemáticas. Además, nos permite utilizar funciones de orden superior, que en ciertos casos son mucho más eficientes que las convencionales funciones imperativas.

Pero, la programación funcional, solo debería ser utilizado en ciertos contextos, se ha visto que la programación funcional, no es la indicada para el manejo y conexión de las bases de datos. Ya que sufren de cualquier tipo de problemas. Los algoritmos vistos son un paso para introducirlo a la programación funcional y tenga conocimiento de su existencia. Hay ciertas tareas que son mucho más eficientes con la programación funcional que con la imperativa, los

webservices son un ejemplo de ello. Es así que se concluye que la programación funcional, se debería practicar siempre y cuando se sepa en donde aplicarlo, conociendo sus ventajas y desventajas.

REFERENCIAS

Bahri, A. (2009). Managing the other side of the water cycle: making wastewater an asset. TEC Background Paper No. 13. Global Water Partnership, Stockholm.

Centro de Investigación de la Caña.- [CENICAÑA]. (2008). *Informe anual 2007*. Cali. 108p.

Centro de Investigación de la Caña.- [CENICAÑA]. (2006). *Carta trimestral. Producción de caña de azúcar en el valle del río Cauca.* Año 28. Nº1. 2006. p 16-37.

Food and Agriculture Organization of the United Nations- [FAO]. (2006). *Evapotranspiración del cultivo*. Guías para la determinación de los requerimientos de agua de los cultivos. Estudio FAO riego y drenaje.34p.