Benchmarking entre C#, Java, Python, F#, Scala

Benchmarking between C#, Java, Python, F#, Scala

Cori Galvez Rodrigo Daniel
Coridaniel123@gmail.com

Universidad Mayor De San Andrés, Facultad de Ciencias Puras y Naturales, Programación Funcional

Resumen
El tipo de programación Funcional nos da otro punto de vista de resolver problemas de programación ya que con él se puede implementar funciones de tipo matemática con lo que el código se reduce.

Palabras Clave: Programación Funcional; Código; Tipos de programación; Python; F#

Abstract
The Functional programming type gives us another point of view of solving programming problems since with it mathematical functions can be implemented with which the code is reduced.

Keywords: Functional Programming; Code; Types of programming; Python; F#

1. INTRODUCCIÓN

Al momento de programar hay distintos tipos de programación que son muy importantes, según el problema que se necesite resolver. Entre los que podemos encontrar: La programación estructurada, Programación Modular, Programación Funcional, etc.

A lo largo del segundo semestre del 2021 se vio distintos lenguajes de programación, como ser Python, C#, Haskell, Scala, Java, con la intención de ver cada tipo de programación que maneja cada lenguaje. Con los lenguajes que están basados en el tipo de programación funcional se vio que era muy diferente a los demás lenguajes ya que tenia otra perspectiva de programación.

En este artículo se trata de mostrar la diferencia de tipos de programación centrándonos en el tipo programación funcional, realizando 5 algoritmos en los distintos lenguajes de programación antes mencionados.

1.1 Comparación entre Seudocódigos

Este apartado muestra la diferencia de tamaño y los distintos tipos de estructura de cada código.

Algoritmo	C#	Java	Python

Fibonacci de 4 términos iniciales	private static string fibo()
	{
	string pos = "";
	int c = 0;
	int a = 0;
	int b = 0;
	
	for (int i = 0; i < 4; i++)
	{
	if (i == 0 || i == 1)
	{
	pos = pos+ 0 + ", ";
		private String fibonacci() {
		// TODO Auto-generated method stub
		String serie="";
		int c = 0;
		int a = 0;
		int b = 0;
		
		for (int i = 0; i < 4; i++)
		{
		if (i == 0 || i == 1)
		{
			def fib(request):
			serie=""
			c=0
			a=0
			b=0
			for i in range (4):
			if (i==0 or i ==1):
			serie = serie + "0" + ", "
			a = 1
			else:
			c = a + b
			a = b

	a = 1;
	}
	else {
	c = a + b;
	a = b;
	b = c;
	pos= pos+ Convert.ToString(c) + ", ";
	}
	}
	return pos;
	}
		serie = serie + 0 + ", ";
		a = 1;
		}
		else {
		c = a + b;
		a = b;
		b = c;
		serie = serie + c + ", ";
		}
		}
		return serie;
		}
			b = c
			serie = serie + str(c) + ", "
			
			return HttpResponse("Fibonacci 4 terminos: "+ serie)

Calculadora con funciones de orden superior	static string calculadora(double a, double b, Func<double,double,double> calc)
	{
	return calc(a,b)+"";
	
	}
	public static Func<double,double,double> suma = (a, b) => a +b;
	public static Func<double, double, double> resta = (a, b) => a - b;
	public static Func<double, double, double> multiplicacion = (a, b) => a * b;
	public static Func<double, double, double> division = (a, b) => a / b;
		private Double calculadora(double number1, double number2, BiFunction<Double, Double, Double> op) {
		// TODO Auto-generated method stub
		return op.apply(number1, number2);
		}
		public BiFunction<Double,Double, Double> suma = (a, b) -> a +b;
		public BiFunction<Double,Double, Double> resta = (a, b) -> a -b;
		public BiFunction<Double,Double, Double> multiplicacion = (a, b) -> a *b;
		public BiFunction<Double,Double, Double> division = (a, b) -> a /b;
			def calcu(request):
			a = request.POST['na']
			b = request.POST['nb']
			op = request.POST['op']
			
			if (op == "suma"):
			return HttpResponse("La suma es: "+calculadora(int(a),int(b),suma))
			elif (op == "resta"):
			return HttpResponse("La resta es: "+calculadora(int(a),int(b),resta))
			elif (op == "multiplicacion"):
			return HttpResponse("La multipicacion es: "+calculadora(int(a),int(b),multiplicacion))
			else:
			return HttpResponse("la division es: "+calculadora(int(a),int(b),division))
			
			def calculadora(a, b, fun):
			return fun(a, b);
			suma = lambda a, b:a+b
			resta = lambda a,b:a-b
			multiplicacion = lambda a,b:a*b
			division = lambda a,b:a/b

Calculadora de matrices con funciones de orden superior	static double[,] calculadoraM(double[,] a, double[,] b, Func<double[,], double[,], double[,]> calc)
	{
	return calc(a, b);
	}
	
	public static Func<double[,], double[,], double[,]> suma = (a, b) =>
	{
	for (int i = 0; i < a.GetLength(0); i++)
	{
	for (int j = 0; j < a.GetLength(1); j++)
	{
	a[i,j]=a[i,j]+b[i,j];
	}
	}
	return a;
	};
		private Double[][] calculadora(Double[][] mat1,Double[][] mat2,BiFunction<Double[][],Double[][],Double[][]> op) {
		// TODO Auto-generated method stub
		return op.apply(mat1,mat2);
		}
		
		public BiFunction<Double[][],Double[][],Double[][]> suma = (a,b) ->
		{
		for (int i = 0; i < a.length; i++)
		{
		for (int j = 0; j < a[i].length; j++)
		{
		a[i][j]=a[i][j]+b[i][j];
		}
		}
			def calculadoraM(a, b, fun):
			return fun(a, b);
			def sumaM(a,b):
			for i in range(len(a)):
			for j in range(len(a[i])):
			a[i][j]= a[i][j]+b[i][j]
			return a
			def restaM(a,b):
			for i in range(len(a)):
			for j in range(len(a[i])):
			a[i][j]= a[i][j]-b[i][j]
			return a
			
			def multiplicacionM (a,b):
			matriz = []
			matriz = []
			for i in range(len(a)):
			p = [0]*len(b[0])
			matriz.append(p)
			for k in range(len(a)):
			for i in range(len(b[k])):
			suma=0

	public static Func<double[,], double[,], double[,]> resta = (a, b) =>
	{
	for (int i = 0; i < a.GetLength(0); i++)
	{
	for (int j = 0; j < a.GetLength(1); j++)
	{
	a[i, j] = a[i, j] - b[i, j];
	}
	}
	return a;
	};
	
	public static Func<double[,], double[,], double[,]> multiplicacion = (a, b) =>
	{
	
	double[,] mul = new double[a.GetLength(0), b.GetLength(1)];
	
	for (int k = 0; k < a.GetLength(0); k++)
	{
	for (int i = 0; i < b.GetLength(1); i++)
	{
	double su = 0;
	for (int j = 0; j < a.GetLength(1); j++)
	{
	su = su + (a[k, j] * b[j, i]);
	}
	mul[k, i] = su;
	}
	}
	return mul;
	};
		return a;
		};
		public BiFunction<Double[][],Double[][],Double[][]> resta = (a,b) ->
		{
		for (int i = 0; i < a.length; i++)
		{
		for (int j = 0; j < a[i].length; j++)
		{
		a[i][j]=a[i][j]-b[i][j];
		}
		}
		return a;
		};
		public BiFunction<Double[][],Double[][],Double[][]> multiplicacion = (a,b) ->
		{
		Double[][] mul = new Double[a.length][b[0].length];
		
		for (int k = 0; k < a.length; k++)
		{
		for (int i = 0; i < b[k].length; i++)
		{
		double su = 0;
		for (int j = 0; j < a[i].length; j++)
		{
		su = su + (a[k][j] * b[j][i]);
		}
		mul[k][i] = su;
		}
		}
		return mul;
		};
			for j in range(len(a[i])):
			suma=suma+(a[k][j]*b[j][i])
			matriz[k][i] = suma
			return matriz

Factorial	static int fact(int d)
	{
	int fac = d;
	int res = 1;
	for (int i = 1; i <= fac; i++)
	{
	res = res * i;
	}
	return res;
	}
		static int fact(int d)
		{
		int fac = d;
		int res = 1;
		for (int i = 1; i <= fac; i++)
		{
		res = res * i;
		}
		return res;
		}
			def fac(request):
			n = int(request.POST['nfac'])
			fact = 1
			for i in range(1,n+1):
			fact = fact * i
			
			return HttpResponse("el factorial de "+str(n) +" es:"+str(fact))

Es Primo	static bool esprimo(int p)
	{
	int pri = p;
	int con = 0;
	for (int i = 1; i <= pri; i++)
	{
	if ((pri % i) == 0)
	{
	con++;
		private String prim(int num) {
		int pri = num;
		int con = 0;
		for (int i = 1; i <= pri; i++)
		{
		if ((pri % i) == 0) {
		con++;
		}
		}
			def primo(request):
			primo = int(request.POST['np'])
			con=0
			for i in range(1,primo+1):
			if ((primo % i)==0):
			con= con+ 1
			print (i)
			if (con == 2 or primo ==1):
			re = "es primo"

	}
	}
	if (con == 2 || pri == 1)
	{
	return true;
	}
	return false;
	}
		if (con == 2 || pri == 1)
		{
		return "Es Primo";
		}
		else
		{
		return "No Es Primo";
		}
		
		}
			else:
			re = "no es primo"
			return HttpResponse("El numero "+str(primo)+" "+re)

Como vemos los lenguajes que tienen mayor codigo son Java y C# estos dos lenguajes son muy parecidos en su sintaxis en cambio Pythos esta sentrado en un lenguaje intermedio entre un lenguaje funcional y no funcional.

Ahora compartiremos los códigos en Scala y F#

Algoritmo	Scala	F#

Fibonacci de 4 términos iniciales	def fib():String =
	{
	var ret=""
	var c=0
	var a=0
	var b=0
	var i=0
	while (i< 4)
	{
	if (i ==0 || i ==1)
	{
	ret = ret + 0 +", "
	a=1
	}else{
	c = a + b;
	a = b;
	b = c;
	ret = ret + c +", "
	}
	i=i+1
	}
	return ret
	
	}
		let rec fibsRec a b =
		if a + b < 2 then
		let current = a + b
		let rest = fibsRec b current
		current :: rest
		else
		[]

Calculadora con funciones de orden superior	def calcu (a: Double, b: Double, f1:(Double,Double)=>Double): Double =
	{
	f1(a,b);
	}
	
	def resta (a: Double, b: Double): Double =
	{
	a - b;
		let suma a b = a + b
		let resta a b = a - b
		let multiplicacion a b = a * b
		let division a b :double = a / b
		
		let calculadora= fun op a b -> op a b

	}
	def suma (a: Double, b: Double): Double =
	{
	a + b;
	}
	def multi (a: Double, b: Double): Double =
	{
	a * b;
	}
	def divi (a: Double, b: Double): Double =
	{
	a / b;
	}	

Calculadora de matrices con funciones de orden superior	def calcu (a: Array[Array[Int]], b: Array[Array[Int]], f1:(Array[Array[Int]],Array[Array[Int]])=>Array[Array[Int]]): Array[Array[Int]] =
	{
	f1(a,b);
	};
	
	def suma (a: Array[Array[Int]], b: Array[Array[Int]]):Array[Array[Int]]=
	{
	
	for (p <- 0 to a.length-1)
	{
	for (k <-0 to a(p).length-1)
	{
	a(p)(k) = a(p)(k)+b(p)(k);
	}
	}
	return a;
	}
	def resta (a: Array[Array[Int]], b: Array[Array[Int]]):Array[Array[Int]]=
	{
	
	for (p <- 0 to a.length-1)
	{
	for (k <-0 to a(p).length-1)
	{
	a(p)(k) = a(p)(k)-b(p)(k);
	}
	}
	return a;
	}
	
	def multi (a: Array[Array[Int]], b:
		let rec vecsum a b =
		match a, b with
		| [], [] -> []
		| _, [] -> []
		| [],_ -> []
		| a::taila, b::tailb -> a + b :: (vecsum taila tailb)
		
		let rec matsum a b =
		match a, b with
		| [], [] -> []
		| _, [] -> []
		| [],_ -> []
		| a::atail, b::btail -> (vecsum a b) :: (matsum atail btail)
		
		let rec vecres a b =
		match a, b with
		| [], [] -> []
		| _, [] -> []
		| [],_ -> []
		| a::taila, b::tailb -> a - b :: (vecres taila tailb)
		
		let rec matres a b =
		match a, b with
		| [], [] -> []
		| _, [] -> []
		| [],_ -> []
		| a::atail, b::btail -> (vecres a b) :: (matres atail btail)
		
		let calculadoraM= fun op a b -> op a b

	Array[Array[Int]]):Array[Array[Int]]=
	{
	var mul = ofDim[Int](a.length,b(0).length);
	
	for (k <-0 to a.length-1)
	{
	for (i <- 0 to b(0).length-1)
	{
	var su = 0;
	for (j <- 0 to a(0).length-1)
	{
	su = su + (a(k)(j) * b(j)(i))
	}
	mul(k)(i) = su;
	}
	}
	return mul;
	
	}	

Factorial	def fac (a: Int):Int=
	{
	var fact = a;
	var res = 1;
	for (i <- 1 to fact)
	{
	res = res * i;
	}
	return res;
	}
		let rec fac a=
		match a with
		| 0 -> 1
		| a -> a * fac (a-1)

Es primo	def esprim (a: Int):Boolean=
	{
	var pri = a;
	var con = 0;
	for (i <- 1 to pri)
	{
	if ((pri % i) == 0)
	{
	con=con +1;
	}
	}
	if (con == 2 || pri == 1)
	{
	return true;
	}
	return false;
	
	}
		let modu a b= a % b
		
		let primo a =
		let mutable c = 0
		for i in 1..a do
		if modu a i = 0 then c <- c +
		1
		else printf("")
		if c=2 then "es primo"
		else "no es primo"

Como vemos en las dos tablas que presentamos hay una gran diferencia en sintaxis y extensión de los digititos códigos que vimos.

2. CONCLUSIONES

Hay una gran diferencia entre los distintos tipos de programación o estilos, comparamos el estilo de programación funcional y vimos la gran diferencia en el código de diferentes algoritmos planteados, ya que el estilo de programación funcional nos da como base funciones matemáticas, que se mezclan con códigos de programación y además que pueden crearse funciones de orden superior y recursivas mejor planteadas que los lenguajes que no están centrados en el estilo de programación funcional.

REFERENCIAS

Barrios, J. M. (30 de nov de 2001). Obtenido de https://users.dcc.uchile.cl/~jbarrios/servlets/general.html

Charte, D. (30 de 5 de 2018). *Campus MVP*. Obtenido de https://www.campusmvp.es/recursos/post/programacion-funcional-funciones-de-primera-clase-y-de-orden-superior.aspx

Clasificacionde. (s.f.). *Clasificacionde*. Obtenido de https://www.clasificacionde.org/tipos-de-programacion/

djangogirls. (s.f.). *djangogirls*. Obtenido de https://tutorial.djangogirls.org/es/django/

djangoproject. (s.f.). *djangoproject*. Obtenido de https://www.djangoproject.com/

ibm. (2015). *ibm*. Obtenido de https://www.ibm.com/docs/es/i/7.3?topic=java-jsp-servlet-programming

Microsoft. (4 de nov de 2021). *Microsoft*. Obtenido de https://docs.microsoft.com/es-es/dotnet/fsharp/tutorials/using-functions

realpython. (s.f.). *realpython*. Obtenido de https://realpython.com/tutorials/django/

scala. (s.f.). *scala*. Obtenido de https://docs.scala-lang.org/?_ga=2.127198916.898196173.1639561112-543435357.1638554158

tutorialspoint. (s.f.). *tutorialspoint*. Obtenido de https://www.tutorialspoint.com/fsharp/fsharp_lists.htm

Vazquez, D. (23 de jun de 2019). *Medium*. Obtenido de https://medium.com/@rvazquezmiguel/higher-order-functions-funciones-de-orden-superior-en-c-48721ff99439#:~:text=Las%20funciones%20que%20aceptan%20otras,de%20delegados%20y%20expresiones%20lambda.

wikipedia. (8 de nov de 2020). *wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Java_Servlet

Wikipedia. (12 de oct de 2021). *Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional