Materia:	1S7B-PROGR.LOG.FUNC
Nombre Informe:	Programación funcional en diferentes lenguajes
Alumno(s):	Pizano Zamora Leonardo Gabriel

Objetivo

El objetivo de esta investigación es comprender la sintaxis y las aplicaciones de la programación funcional en los lenguajes de programación Java, C#, Python, Haskell, Kotlin, PHP, LISP, ML, y Scala. Se explorarán las construcciones funcionales clave y se proporcionarán ejemplos prácticos para cada lenguaje.

Marco Teórico

La programación funcional es un paradigma de programación que trata la computación como la evaluación de funciones matemáticas y evita el estado mutable y los datos cambiantes. Se centra en la aplicación de funciones y la composición de funciones, favoreciendo las funciones puras y las expresiones sin efectos secundarios.
Características clave de la programación funcional:

Funciones de Primera Clase y Orden Superior: Las funciones pueden ser asignadas a variables, pasadas como argumentos y devueltas como valores.
Inmutabilidad: Los datos no cambian una vez creados, lo que evita efectos secundarios.
Funciones Puras: Las funciones siempre producen el mismo resultado para los mismos argumentos y no tienen efectos secundarios.
Expresiones Lambda: Funciones anónimas definidas en línea.
Composición de Funciones: Combinar funciones simples para construir funciones más complejas.
Recursión: Utilizada en lugar de bucles imperativos.

Desarrollo

Java

Construcciones y Aplicaciones: En Java, la programación funcional se facilita a través de la API de Streams y expresiones lambda introducidas en Java 8.

Ejemplos:

Expresiones Lambda:

java

```
// Expresión lambda para sumar dos números
BinaryOperator<Integer> sum = (a, b) -> a + b;
```

System.out.println(sum.apply(5, 3)); // Output: 8

API de Streams:

java

```
// Uso de streams para filtrar y mapear una lista
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> evenSquares = numbers.stream()
                                   .filter(n -> n % 2 == 0)
                                   .map(n -> n * n)
                                   .collect(Collectors.toList());
System.out.println(evenSquares); // Output: [4, 16]
```

C#

Construcciones y Aplicaciones: C# soporta la programación funcional a través de delegados, expresiones lambda y la API de LINQ.

Ejemplos:

Expresiones Lambda:

csharp

```
// Expresión lambda para multiplicar dos números
Func<int, int, int> multiply = (a, b) => a * b;
Console.WriteLine(multiply(4, 6)); // Output: 24
```

LINQ:

csharp

```
// Uso de LINQ para filtrar y mapear una lista
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
var evenSquares = numbers.Where(n => n % 2 == 0)
                         .Select(n => n * n)
                         .ToList();
Console.WriteLine(string.Join(", ", evenSquares)); // Output: [4, 16]
```

Python

Construcciones y Aplicaciones: Python soporta la programación funcional a través de funciones de orden superior, expresiones lambda y funciones integradas como map, filter, y reduce.

Ejemplos:

Expresiones Lambda:

python

```
# Expresión lambda para restar dos números
subtract = lambda a, b: a - b
print(subtract(10, 4)) # Output: 6
```

Funciones map y filter:

python

```
# Uso de map y filter para operar sobre una lista
numbers = [1, 2, 3, 4, 5]
even_squares = list(map(lambda x: x*x, filter(lambda x: x % 2 == 0, numbers)))
print(even_squares) # Output: [4, 16]
```

Haskell

Construcciones y Aplicaciones: Haskell es un lenguaje puramente funcional, lo que significa que todas las construcciones soportan la programación funcional de manera nativa.

Ejemplos:

Funciones:

haskell

```
-- Definir una función para sumar dos números
sum' :: Int -> Int -> Int
sum' a b = a + b

main = print (sum' 5 3) -- Output: 8
```

List Comprehensions:

haskell

```
-- Uso de list comprehensions para filtrar y mapear una lista
evenSquares = [x^2 | x <- [1..5], even x]
main = print evenSquares -- Output: [4, 16]
```

Kotlin

Construcciones y Aplicaciones: Kotlin soporta la programación funcional a través de funciones de orden superior, lambdas y colecciones.

Ejemplos:

Expresiones Lambda:

kotlin

```
// Expresión lambda para dividir dos números
val divide: (Int, Int) -> Int = { a, b -> a / b }
println(divide(10, 2)) // Output: 5
```

Operaciones en Colecciones:

kotlin

```
// Uso de funciones de orden superior en colecciones
val numbers = listOf(1, 2, 3, 4, 5)
val evenSquares = numbers.filter { it % 2 == 0 }
                         .map { it * it }
println(evenSquares) // Output: [4, 16]
```

PHP

Construcciones y Aplicaciones: PHP soporta la programación funcional a través de funciones anónimas y funciones integradas como array_map, array_filter, y array_reduce.

Ejemplos:

Funciones Anónimas:

php

```
// Función anónima para multiplicar dos números
$multiply = function($a, $b) {
    return $a * $b;
};
echo $multiply(4, 5); // Output: 20
```

Funciones array_map y array_filter:

php

```
// Uso de array_map y array_filter para operar sobre un array
$numbers = [1, 2, 3, 4, 5];
$evenSquares = array_map(function($x) {
    return $x * $x;
}, array_filter($numbers, function($x) {
    return $x % 2 == 0;
}));
print_r($evenSquares); // Output: [4, 16]
```

LISP

Construcciones y Aplicaciones: LISP es un lenguaje funcional clásico que soporta funciones de orden superior y recursión de manera nativa.

Ejemplos:

Definir Funciones:

lisp

```
; Definir una función para sumar dos números
(defun sum (a b)
  (+ a b))
(print (sum 5 3)) ; Output: 8
```

Mapcar y Filter:

lisp

```
; Uso de mapcar y filter para operar sobre una lista
(defun evenp (n)
  (= 0 (mod n 2)))
(mapcar #'(lambda (x) (* x x)) (remove-if-not #'evenp '(1 2 3 4 5)))
; Output: (4 16)
```

ML

Construcciones y Aplicaciones: ML es un lenguaje funcional que soporta funciones de orden superior, inmutabilidad y tipado estático.

Ejemplos:

Definir Funciones:

sml

```
(* Definir una función para restar dos números *)
fun subtract (a, b) = a - b
val result = subtract(10, 4)
(* Output: 6 *)
```

List Comprehensions:

sml

```
(* Uso de list comprehensions para filtrar y mapear una lista *)
val numbers = [1, 2, 3, 4, 5]
```

```
val evenSquares = List.map (fn x => x * x) (List.filter (fn x => x mod 2 = 0) numbers)
(* Output: [4, 16] *)
```

Scala

Construcciones y Aplicaciones: Scala es un lenguaje funcional y orientado a objetos que soporta funciones de orden superior, expresiones lambda y colecciones.

Ejemplos:

Expresiones Lambda:

scala

```
// Expresión lambda para sumar dos números
val sum = (a: Int, b: Int) => a + b
println(sum(5, 3)) // Output: 8
```

Operaciones en Colecciones:

scala

```
// Uso de funciones de orden superior en colecciones
val numbers = List(1, 2, 3, 4, 5)
val evenSquares = numbers.filter(_ % 2 == 0).map(x => x * x)
println(evenSquares) // Output: [4, 16]
```

Resultado

Los ejemplos presentados demuestran cómo se pueden aplicar las construcciones funcionales en varios lenguajes de programación. Cada lenguaje tiene su propia sintaxis y peculiaridades, pero todos comparten los principios fundamentales de la programación funcional.

Evaluación

La programación funcional ofrece numerosas ventajas, como la facilidad para razonar sobre el código y la reducción de errores debido a la inmutabilidad y la ausencia de efectos secundarios. Los lenguajes de programación investigados permiten aplicar estos principios de diversas maneras, proporcionando flexibilidad y poder expresivo.

Conclusiones}

La programación funcional es un paradigma poderoso y ampliamente soportado en muchos lenguajes de programación modernos. A través de funciones de primera clase, inmutabilidad y expresiones lambda, los desarrolladores pueden escribir código más limpio, modular y fácil de mantener. Cada lenguaje ofrece construcciones

específicas para facilitar la programación funcional, lo que permite a los desarrolladores elegir el enfoque que mejor se adapte a sus necesidades y preferencias.