

Examen con respuestas generadas por ia
(Segundo Branch)

Nombre: Nathan Isaac Moreno Cuenca

Fecha: 14/11/2025

1. Primer prompt del primer ejercicio

Necesito ayuda con este problema en un examen te adjuntare primero una pregunta de tres y abajo de la pregunta mi codigo que he echo la primera hora Pregunta 1

Transforme el siguiente método Java desarrollado de forma imperativa a una versión funcional utilizando Scala.

```
double myMethod(List<Double> datos) {  
    List<Double> logs = new ArrayList<>();  
    for (int i = 0; i < datos.size(); i++) {  
        logs.add(Math.log(datos.get(i)));  
    }  
    double sumaLogs = 0.0;  
    for (int i = 0; i < logs.size(); i++) {  
        sumaLogs += logs.get(i);  
    }  
    double promedioLog = sumaLogs / logs.size();  
    double sumaCuadrados = 0.0;  
    for (int i = 0; i < logs.size(); i++) {  
        double diferencia = logs.get(i) - promedioLog;  
        sumaCuadrados += diferencia * diferencia;  
    }  
    return Math.sqrt(sumaCuadrados / logs.size());  
}
```

mi respuesta es

```
def mymethod(datos: List[Double]): Double = {  
    val Logs = datos.map(math.log)
```

```
val promedioLog = Logs.sum / Logs.size  
val suma = Logs.map(x => {  
    val diferencia = x - promedioLog  
    diferencia * diferencia  
}).sum  
suma / Logs.size  
}
```

dime que puede mejorar o si mi respuesta esta bien

Respuesta :

Tu respuesta está **casi perfecta**, pero hay **dos detalles importantes a mejorar**:

1. El resultado final debe ser la raíz cuadrada

El método original devuelve:

```
return Math.sqrt(sumaCuadrados / logs.size());
```

En tu versión Scala, **te olvidaste del sqrt al final**.

Tu código devuelve la **varianza**, pero el ejercicio pide la **desviación estandar**.

2. Estilo funcional y nombres en minúscula

En Scala, por convención los nombres de variables van en minúscula (logs, no Logs).

★ Versión corregida de tu propuesta (la mínima corrección)

```
def mymethod(datos: List[Double]): Double = {  
    val logs = datos.map(math.log)  
    val promedioLog = logs.sum / logs.size  
    val suma = logs.map { x =>  
        val diferencia = x - promedioLog  
        diferencia * diferencia  
    }.sum
```

```
    math.sqrt(suma / logs.size)  
}
```

★ Versión aún más funcional y compacta (por si te piden estilo más idiomático)

```
def mymethod(datos: List[Double]): Double = {  
  
    val logs = datos.map(math.log)  
  
    val promedio = logs.sum / logs.length  
  
    math.sqrt(logs.map(x => math.pow(x - promedio, 2)).sum / logs.length)  
}
```

★ Versión top funcional (sin variables intermedias mutables)

```
def mymethod(datos: List[Double]): Double = {  
  
    val logs = datos.map(math.log)  
  
    val promedio = logs.sum / logs.size  
  
    val varianza = logs.map(l => (l - promedio) * (l - promedio)).sum / logs.size  
  
    math.sqrt(varianza)  
}
```

● Conclusión

Tu respuesta está **bien encaminada**, es funcional, usa map, sum, etc.

Solo faltaba:

- ✓ aplicar math.sqrt
- ✓ mejorar el estilo de nombres
- ✓ (opcional) compactar la expresión

Con esas correcciones tu solución es **válida y correcta para el examen**.

2. Primer prompt del segundo ejercicio

Pregunta 2

Ajuste de precios con políticas variables

En una tienda digital se trabaja con listas de precios que necesitan ser ajustadas antes de ser mostradas al usuario. De acuerdo con el país, la temporada o las promociones vigentes, el ajuste puede consistir en agregar un impuesto, aplicar un descuento o modificar el precio según cierta estrategia comercial. Se requiere elaborar una solución donde exista una única función capaz de recibir la lista de precios y, además, la función que representa la regla de ajuste que se quiera aplicar en ese momento. Esta función deberá retornar una nueva lista de precios ya modificados de acuerdo con la política específica que se haya suministrado.

mi respuesta fue echa incluso con quema de datos

```
/* Segundo ejercicio */
val lista = List(100.0, 200.0, 350.0)
val iva: Double => Double = p => p * 1.19
val descuento: Double => Double = p => p * 0.90
def precios(precios: List[Double], politica_tienda: Double => Double): List[Double] = {
    precios.map(politica_tienda)
}
val ivap = precios(lista, iva)
val descuentop = precios(lista, descuento)
println(s"precio original: $lista")
println(s"iva: $ivap")
println(s"descuento $descuentop")
```

dime si esta bien o que hay que mejorar y si hay que mejorar dame el codigo

Respuesta :

Tu respuesta está **muy bien** hecha. De hecho, cumple exactamente con lo que pide el ejercicio:

- ✓ Una función que reciba:
 - la lista de precios
 - la función/política de ajuste
- ✓ Devuelve una **nueva lista modificada** (no cambia la original).
- ✓ Usa funciones como valores (Double => Double).
- ✓ Usa map, lo cual es funcional y correcto.

Tu solución está totalmente bien para un examen.

Pequeñas mejoras opcionales (no obligatorias)

Si quieres dejar el código más “Scala idiomático”, puedes:

1. Usar nombres de funciones en minúsculas

politicaTienda en lugar de politica_tienda (por convención Scala).

2. Tipar menos (Scala infiere tipos automáticamente)

Ejemplo: no necesitas Double => Double explícitamente.

3. Hacer la función más genérica (opcional)

Pero solo si quisieras ir más allá.

★ Tu código corregido con mejoras de estilo

```
val lista = List(100.0, 200.0, 350.0)
```

```
// Políticas de precio
```

```
val iva = (p: Double) => p * 1.19
```

```
val descuento = (p: Double) => p * 0.90
```

```
// Función que aplica una política a toda la lista
```

```
def precios(lista: List[Double], politicaTienda: Double => Double): List[Double] = {  
    lista.map(politicaTienda)  
}
```

```
// Ejemplos de uso
```

```
val preciosConIVA = precios(lista, iva)
```

```
val preciosConDescuento = precios(lista, descuento)
```

```
println(s"Precio original: $lista")
```

```
println(s"Con IVA: $preciosConIVA")
```

```
println(s"Con descuento: $preciosConDescuento")
```

★ Versión aún más compacta (opcional, estilo profesional)

```
def ajustarPrecios(precios: List[Double])(politica: Double => Double): List[Double] =  
    precios.map(politica)
```

```
val lista = List(100.0, 200.0, 350.0)

val iva = (p: Double) => p * 1.19
val descuento = (p: Double) => p * 0.90

println(ajustarPrecios(lista)(iva))
println(ajustarPrecios(lista)(descuento))
```

Esto usa *currying*, una técnica que a los profesores de programación funcional les encanta 😊.

● Conclusión

Tu respuesta está **correcta y completa**.

Solo necesitabas mejorar un poco el estilo, pero funcionalmente está **perfecta para un examen**.

3. Primer prompt del tercer ejercicio

ahora el ultimo ejercicio Pregunta 3

Transforme la función anónima a una función con nombre

```
val generadorIncrementadorAnonimo: Int => (Int => Int) =
  (incremento: Int) =>
    (x: Int) =>
      x + incremento
```

mi respuesta fue

```
/* Tercer Ejercicio*/
def generadorIncrementador(nombre1: Int): Int => Int = /* Funcion con nombre1 */
  (nombre2: Int) => nombre2 + nombre1 /* Funcion resultante: suma el incremento capturado al
argumento que es nombre2 */
```

dime si hay como mejorar o esta bien y sino generame la respuesta

Respuesta :

Tu respuesta está **correcta**:

- ✓ Convertiste la función anónima en una función con nombre.
- ✓ Mantiene exactamente la misma estructura funcional: una función que recibe un entero y devuelve otra función que suma ese entero.

Sin embargo, **hay dos mejoras opcionales**:

1. Usar nombres más claros (incremento, x) en lugar de nombre1 y nombre2.
 2. Escribirlo con llaves para que sea más legible (no obligatorio).
-

★ Versión corregida y mejorada (la que presentarías en el examen)

```
def generadorIncrementador(incremento: Int): Int => Int = {  
    (x: Int) => x + incremento  
}
```

Esta versión es:

- clara
 - idiomática
 - totalmente equivalente a la versión anónima
-

★ Versión aún más funcional (opcional)

```
def generadorIncrementador(incremento: Int): Int => Int =  
    x => x + incremento
```

Scala infiere el tipo de x.

★ Versión ultra simple (opcional)

Usando funciones parciales:

```
def generadorIncrementador(incremento: Int): Int => Int =  
    incremento + _
```

Cuando Scala ve incremento + _ entiende “una función que suma incremento al argumento”.

● Conclusión

Tu respuesta está **bien** y es **válida para el examen**, solo necesitaba mejores nombres de variables. La versión sugerida arriba sería la ideal.