# Taller 1: Recursión


Juan Francisco Díaz Frias  Emily Núñez

Agosto 2025

## 1. Ejercicios de programación

### 1.1. Máximo de una lista de enteros

Se desea calcular el máximo de una lista de enteros positivos, no vacía.

Implemente las funciones `maxLin`, y `maxIt` que calculen ese valor, pero la primera genere un proceso de recursión lineal, mientras la segunda genere un proceso iterativo.

Su solución debe tener el aspecto siguiente:

```
  def maxLin(l: List[Int]): Int = {
...
  }

  def maxIt(l: List[Int]): Int = {
...
  }
```

Hay tres métodos que se proveen en List[Int] que pueden ser útiles para este ejercicio:

- l.isEmpty: Boolean (devuelve si una lista $l$ está vacía)
- l.head: Int (devuelve el primer elemento de la lista $l$)
- l.tail: List[Int] (devuelve la lista sin el primer elemento $l$)

### 1.2. Torres de Hanoi

Dios, al crear el mundo, colocó tres varillas de diamante con 64 discos apilados de mayor a menor radio en la primera varilla. También creó un monasterio con monjes, quienes tenían la tarea de resolver esta Torre de Hanoi divina. Para poder resolverla, los monjes debían llevar todos los discos a la última varilla quedando en el mismo orden a como estaban en la primera; además, se impuso tres condiciones:

- sólo se puede mover un disco a la vez,
- un disco de mayor diámetro no puede descansar sobre otro diámetro menor, y
- sólo se puede mover el disco que se encuentra arriba en cada varilla

Dios profetizó que el día en que estos monjes consiguieran terminar el juego, el mundo acabaría. Si los monjes movieran cada segundo un disco, sin equivocarse, y la historia fuera cierta, ¿se podría calcular cuándo será el fin del mundo?

Su tarea en este ejercicio consiste en implementar, usando recursión, dos funciones: `movsTorresHanoi` y `torresHanoi`. La primera, `movsTorresHanoi` , recibe $n$, el número de discos y devuelve el número mínimo de movimientos necesarios para mover $n$ discos de la primera varilla (o torre) a la tercera usando la segunda como intermediaria. Con este valor, se podría calcular cuántos siglos se demorarían los monjes en acabar la tarea.

```
def movsTorresHanoi(n:Int):BigInt = {...
}
```

La segunda, `torresHanoi`, recibe $n$, el número de discos, y tres enteros $t1, t2, t3$ representando los identificadores de las tres varillas (torres) y devuelve una lista de movimientos, donde un movimiento es una pareja $(a, b)$ donde $a, b \in \{t1, t2, t3\} \wedge a \neq b$, indicando que se mueva el disco en lo alto de la varilla $a$ hacia la varilla $b$.

```
def torresHanoi(n:Int, t1:Int, t2:Int, t3:Int): List[(Int,Int)] = {
   // Pasar n discos de la  torre t1 a la torre t3 usando t2 como intermediaria
   // Devuelve la lista de movimientos de parejas (a,b) indicando mover un disco de la torre a hacia
   // la torre b
...
}
```

# 2. Entrega

## 2.1. Paquete *FuncionesRecursivas* y *worksheet* de pruebas

Usted deberá entregar dos archivos `package.scala` y `pruebas.sc` los cuales harán parte de su estructura de proyecto `IntellijIdea`, como se muestra en la figura a continuación:


Las 4 funciones correspondientes a cada ejercicio, `maxLin`, `maxIt`, `movsTorresHanoi`, y `torresHanoi` deben ser implementadas en un paquete de Scala denominado `FuncionesRecursivas`.

En ese paquete debe venir un archivo denominado `package.scala` que debe tener la forma siguiente:

```
package object FuncionesRecursivas {
  def maxLin(l: List[Int]): Int = {
 ...
  }

  def maxIt(l: List[Int]): Int = {
...
  }

  def movsTorresHanoi(n:Int):BigInt = {
...
  }

  def torresHanoi(n:Int, t1:Int, t2:Int, t3:Int): List[(Int,Int)] = {
    // Pasar n discos de la  torre t1 a la torre t3 usando t2 como intermediaria
    // Devuelve la lista de movimientos de parejas (a,b) indicando mover un disco de la torre a
    // hacia  la torre b
...
  }



}
```

Dicho paquete será usado en un *worksheet* de Scala con casos de prueba. Estos casos de prueba deben venir en un archivo denominado `pruebas.sc` . Un ejemplo de un tal archivo es el siguiente:

```
import FuncionesRecursivas._
maxLin(List(3,20,5,4))
maxIt(List(3,20,5,4))

torresHanoi(1, 1, 2, 3)
torresHanoi(2, 1, 2, 3)
torresHanoi(3, 1, 2, 3)
torresHanoi(4, 1, 2, 3)

movsTorresHanoi(1)
movsTorresHanoi(2)
movsTorresHanoi(3)
movsTorresHanoi(4)
movsTorresHanoi(5)
movsTorresHanoi(64)
val  siglo:BigInt = BigInt(60)*BigInt(60)*BigInt(24)*BigInt(365)*BigInt(100)
movsTorresHanoi(64)/siglo
```

Al ejecutarlo debe dar lo siguiente:

```
import FuncionesRecursivas._
val res0: Int = 20
val res1: Int = 20

val res2: List[(Int, Int)] = List((1,3))
val res3: List[(Int, Int)] = List((1,2), (1,3), (2,3))
val res4: List[(Int, Int)] = List((1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3))
val res5: List[(Int, Int)] = List((1,2), (1,3), (2,3), (1,2), (3,1), (3,2), (1,2),
                                  (1,3), (2,3), (2,1), (3,1), (2,3), (1,2), (1,3), (2,3))

val res6: BigInt = 1
val res7: BigInt = 3
val res8: BigInt = 7
val res9: BigInt = 15
val res10: BigInt = 31
val res11: BigInt = 18446744073709551615
val siglo: BigInt = 3153600000
val res12: scala.math.BigInt = 5849424173
```

## 2.2. Informe del taller - secciones

Todo taller debe venir acompañado de un informe en formato pdf. El informe debe contener la información explícita solicitada en el enunciado.

Para este caso, el informe del taller debe contener al menos tres secciones: informe de procesos, informe de corrección y conclusiones.

### 2.2.1. Informe de procesos

Tal como se ha visto en clase, los procesos generados por los programas recursivos podrían ser iterativos o recursivos (lineales o de árbol). Para cada una de las soluciones entregadas argumente qué tipo de procesos se generan y consígnelo en esta sección del informe.

### 2.2.2. Informe de corrección

Es muy importante reflexionar sobre la corrección del código entregado. Para ello se deberá argumentar sobre la corrección de los programas entregados, y también deberá entregar un conjunto de pruebas. Todo esto lo consigna en esta sección del informe, dividida de la siguiente manera:

**Argumentación sobre la corrección** Para cada función, `maxLin`, `maxIt`, `movsTorresHanoi`, y `torresHanoi`, argumente lo más formalmente posible por qué es correcta. Utilice inducción o inducción estructural donde lo vea pertinente. Estas argumentaciones las consigna en esta sección del informe.

**Casos de prueba** Para cada función se requieren mínimo 5 casos de prueba donde se conozca claramente el valor esperado, y se pueda evidenciar que el valor calculado por la función corresponde con el valor esperado en toda ocasión.

Una descripción de los casos de prueba diseñados, sus resultados y una argumentación de si cree o no que los casos de prueba son suficientes para confiar en la corrección de cada uno de sus programas, los registra en esta sección del informe. Obviamente, esta parte del informe debe ser coherente con el archivo de pruebas entregado, `pruebas.sc`.

## 2.3. Fecha y medio de entrega

Todo lo anterior, es decir los archivos `package.scala`, `pruebas.sc`, e `Informe de corrección`, debe ser entregado vía el campus virtual, a más tardar a las 10 a.m. del jueves 11 de septiembre de 2025, en un archivo comprimido que traiga estos tres elementos.

Cualquier indicación adicional será informada vía el foro del campus asociado a *programación funcional*.

# 3. Evaluación

Cada taller será evaluado tanto por el profesor del curso con ayuda del monitor, como por 2 compañeros que hayan entregado el taller. A este tipo de evaluación se le conoce como *Coevaluación*.

El objetivo de la coevaluación es lograr aprendizajes a través de:

- La lectura de lo que otros compañeros hicieron ante el mismo reto. Esto permite contrastar las soluciones propias con las de otros, y aprender de ellas, o compartir mejores maneras de hacer algo con otros.
- Retroalimentar a los compañeros que fueron asignados para evaluar. Al escribir la percepción que tenemos sobre el trabajo del otro, podemos aprender de cómo lo hicieron, y dar indicaciones al otro sobre otra formas de hacer lo mismo o, incluso, felicitarlo por la solución que presenta.
- La lectura de las retroalimentaciones de mis compañeros o del profesor/monitor.

La calificación de cada taller corresponderá entonces:

- en un 80 % a la calificación ponderada que reciba del profesor/monitor, vía una rúbrica de evaluación (pesa 5 veces lo que pesa la de otro estudiante) y de los tres compañeros asignados para evaluarlo.
- en un 20 % a la calificación que el sistema hará del trabajo de evaluación asignado. El Sistema tiene un método inteligente para estimar esa calificación, a partir de las evaluaciones realizadas por cada estudiante y por el profesor/monitor.