1. Crear una serie de funciones que devuelva una lista de números enteros en secuencia (Ejemplo UsoFunciones)
   1. def crearLista(n:Int):List[Int]
      1. Por ejemplo: crearLista(3) 🡪 List(1,2,3)
      2. Sugerencias: usar rangos para crear la lista
   2. def crearListaRequire(n:Int):List[Int]
      1. Restricción: n >= 1. Emplear require para cumplirla
   3. def crearListaOption(n:Int):Option[List[Int]]
      1. Restricción: n >= 1. Emplear Option para cumplirla
   4. def crearListaTry(n:Int):Try[List[Int]]
      1. Restricción: n >= 1. Emplear require para cumplirla
   5. Comprobar que las funciones creadas en los puntos a y b funcionan correctamente
   6. Emplear el reconocimiento de patrones para obtener la lista generada en los puntos c y d. Probar también el método getOrElse
2. Observar el código de las 4 funciones creadas en el ejercicio anterior. ¿Hay código duplicado? Si lo hay, ¿cómo podríamos eliminarla?
   1. Sugerencia: pensar en términos de funciones más que de objetos
3. Optimización de operaciones con colecciones (Ejemplo UsoFunciones)
   1. Crear una lista de 10 elementos
   2. Crear las dos funciones literales como valores siguientes:
      1. val masUno:Int=>Int = n => {
         1. println(s”Sumando $n”)
         2. n + 1
         3. }
      2. val porDiez:Int=>Int = n => {
         1. println(s”Multiplicando $n”)
         2. n \* 10
         3. }
      3. lista map masUno map porDiez
      4. Observar que cada map genera una lista como resultado
      5. Declarar la siguiente función:
         1. val entonces = masUno andThen porDiez
         2. lista map entonces
         3. Observar la diferencia
      6. Declarar esta otra función:
         1. val compuesta = masUno compose porDiez
         2. lista map compuesta
         3. Observar la diferencia
4. Operaciones con funciones de alto nivel (Ejemplo OperacionesFunciones):
   1. Crear una lista de 10 elementos y sumar sus contenidos: List(1,2,3) == 6
      1. Sugerencias: examinar los métodos sum, reduce, reduceLeft, reduceRight, fold, foldLeft, foldRight
   2. Crear una lista de 10 elementos y mostrar todos los números pares: List(1,2,3) == 2
      1. Sugerencias: examinar el método filter y withFilter
   3. Crear una lista de 10 elementos y transformarla en una lista de tuples, cada uno de ellos conteniendo el elemento y su cuadrado : List(1,2,3) == List((1,1), (2,4), (3,9)). Convertir el resultado en un mapa
      1. Sugerencias: examinar los métodos map y toMap
   4. Crear una lista de 10 elementos y ejecutar las siguientes secuencias:
      1. Lista zip lista
      2. Lista zipWithIndex
      3. Observar los resultados
   5. Crear una lista de 10 elementos y agruparla en dos nuevas colecciones que contengan los pares e impares de la lista original
      1. Sugerencias: examinar el método groupBy y partition. Usar ambos y comparar los resultados
      2. Ejecutar lista grouped 4 y observar los resultados
   6. Crear una lista de 10 elementos, desordenarla (Random.shuffle) y transformarla con esta función:
      1. def f(x:Int) = if(x > 5) Some(x) else None
      2. Emplear los métodos map y flatMap y observar los resultados
5. Crear una lista heterogénea y extraer de ella únicamente los números enteros (Ejemplo OperacionesFunciones):
   1. List(2,None,6,Unit,null) 🡪 List(2,6)
   2. Sugerencia: examinar el método collect y las funciones parciales. La función parcial solo debe ocuparse de considerar el caso de que el objeto que examine sea un Int
6. Crear una lista de 10 elementos y mostrar sus contenidos por consola empleando un iterador (Ejemplo OperacionesFunciones):
   1. Sugerencia: examinar el método iterator de List
7. Leer el archivo de texto “test.txt” (carpeta archivos) y mostrar su contenido por consola. Después, mostrar únicamente las 10 primeras líneas (Ejemplo OperacionesFunciones):
   1. Sugerencias: copiar el archivo “test.txt” al directorio de vuestro proyecto. Emplear la clase scala.io.Source. Examinar los métodos from\* para ver cuál es el más adecuado. Examinar el método getLines y take
8. Escribir un programa que nos diga el número de palabras que tiene el archivo “test.txt” usando como separador los blancos (Ejemplo ContarPalabras).
9. Crear una función que devuelve el número de palabras que hay en una String usando como separador los blancos (Ejemplo ContarPalabrasFuture):
   1. def contarPalabras(s:String):Future[Int]
   2. comprobar que la función opera correctamente
10. Escribir un programa que nos diga el número de palabras que tiene el archivo “test.txt” usando como separador los blancos y empleando futuros (Ejemplo ContarPalabrasFuture):
    1. Sugerencia: para facilitar la resolución de este ejercicio, examinar los métodos Future.sequence y Future.traverse