Código Semana 5------------------------------------

def isEven(k : Int) : Int = (k % 2)match {

case 0 => 1

case \_=> 0

}

def countEven(s : List[Int]) : Int = s.map(isEven).sum

def countEven(s : List[Int]) : Int = {

def isEven(k : Int) : Int = (k % 2)match {

case 0 => 1

case \_=> 0

}

s.map(isEven).sum

}

def countEven(s : List[Int]) : Int = {

val isEven(k : Int) : Int => k % 2 match {

case 0 => 1

case \_=> 0

}

s.map(isEven).sum

}

1. Mapeo de una lista de cadenas de texto a lista de enteros que representan la longitud del texto

val names : List[String] = List("Leo", "Cristiano", "Enner", "Felipe")

names.map(\_.length)

1. Dada una lista de números enteros, desarrollar las funciones necesarias que le permitan contar cuantos elementos de la lista son números primos

val numbers = List(3, 4, 7, 11, 12)

val isPrime = (nro : Int) => (2 to nro - 1).forall(nro % \_ != 0)

numbers.map(isPrime(\_) match {

case true => 1

case false => 0

}).sum

1. Otro tipo de funciones

val numbers = List(6, 28, 15, 12, 11, 24)

numbers.max

res26: Int = 28

numbers.min

res27: Int = 6

numbers.size

res28: Int = 6

1. El forall devuelve true sí y solo si el predicado devuelve true para todos los valores de la lista

val isPrime = (nro : Int) : Boolean = (2 until nro).forall(nro % \_ != 0)

1. El exists devuelve true si y solo si el predicado devuelve true para por lo menos un valor de la lista

def isPrime(nro : Int) : Boolean = !(2 until nro).exists(nro % \_ == 0)

1. El filter devuelve una lista que únicamente contiene los valores que cumplen con el predicado

List(1, 2, 3, 4, 5).filter(k => k % 3 != 0)

res: List[Int] = List(1, 2, 4, 5)

1. El takeWhile trunca la lista cuando encuentra un valor que no cumple con el predicado

List(1, 2, 3, 4, 5).takeWhile(k => k % 3 != 0)

res: List[Int] = List(1, 2)

(n : Int) => (1 until n).filter(div => n % div == 0)

val sumDiv = (n : Int) => (1 until n).filter(div => n % div == 0)



val numbers = List(6, 28, 15, 12, 11, 24)

val sumDiv = (nro : Int) => (1 until nro).filter(div => nro % div == 0)

numbers.filter(nro => nro == sumDiv2(nro)).size

1. Factorial escalonado

def FactorialEscalonado(n : Int) : Int ={

n % 2 match {

case 0 => (2 to n by 2).product

case \_ => (1 to n by 2).product

}

}

FactorialEscalonado(8)

FactorialEscalonado(9)

1. Pares e Impares

val numbers = (1 to 20).toList

//Contar pares

numbers.filter(nro => nro % 2 == 0).size

numbers.count(nro => nro % 2 == 0)

//Contar impares

numbers.filter(nro => nro % 2 != 0).size

numbers.count(nro => nro % 2 != 0)

1. Cuenta primos

//Contar primos

def contarPrimos(nros : List[Int]) : Int = {

val isPrime = (n : Int) => (2 until n).forall(n % \_ != 0)

nros.filter(isPrime).size

//nros.count(isPrime)

}

contarPrimos(numbers)

1. 3 factores

//Presentar 3 factores

def tresFactores(nros : List[Int]) : List[Int] = {

val factores = (n : Int) => (2 until n).filter(n % \_ == 0)

nros.filter(nro => factores(nro).size == 3)

}

tresFactores((1 to 1000).toList)