Sergio Couto
s.couto@indizen.com

Introducción a Scala y Programación Funcional

Mayo 2018

indizen
Technology & Business Confluence

ÍNDICE

01 | **Información del curso** - Software y repositorio

indizen
Technology & Business Confluence

- SBT: Herramienta de construcción interactiva
  - Gestiona dependencias
  - Instrucciones de instalación
    - En Windows
    - En Linux
- Scala REPL
  - Incluída en SBT
- IntelliJ
  - Descarga e Instalación
- Scastie – Consola online

01 | **Información del curso** - Temario

indizen
Technology & Business Confluence

- Introducción
- SBT y REPL
- Básicos
  - Sintaxis, variables, definición de funciones
- Funciones: Orden superior, recursividad, puras, anónimas, parciales
- Colecciones
- Pattern Matching

02 | Introducción

indizen
Technology & Business Confluence

> ¿ Por qué scala?
  - Surge en 2004 en la Escuela Politécnica de Lausana
  - Orientado a objetos
  - Funcional
  - Se ejecuta en la JVM
  - Consola interactiva
  - Fuertemente tipado
  - Inferencia de tipos
  - Conciso. Evita escribir de más.

> Frameworks muy extendidos: Akka y Spark

akka

APACHE Spark™

03 | **SBT** - Simple Build Tool

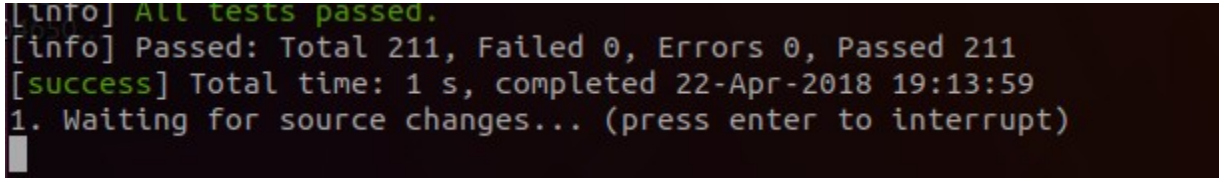
indizen
Technology & Business Confluence

sbt

- Herramienta de construcción de proyectos
- Gestión de dependencias
- Opciones
  - compile, run, test, publish
  - console => abre REPL de Scala

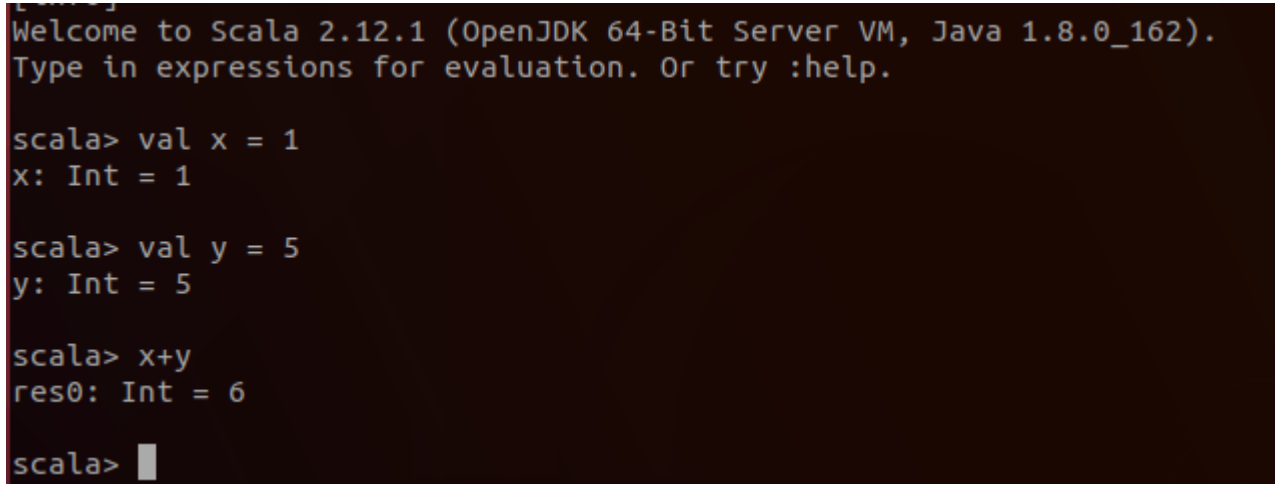
- Identifica cambios en código

- Dispone de modo interactivo (vírgula)
  - ~test
  - ~compile



03 | REPL - Read – Eval – Print -Loop

indizen
Technology & Business Confluence



- Consola interactiva de scala
  - Autocompletado
  - Similar a la worksheet de IntelliJ
  - Permite hacer imports y cualquier sentencia de scala
- Opciones para arrancar
  - Sbt console
  - Scala
- Útil para *trastear* y hacer pruebas rápidas
- Aunque no nombres una variable, toda sentencia devuelve un objeto resultado resX
- Modo *paste* para introducir sentencias multilíneas
  - :paste para iniciarla y Control + D para salir

04 | **Básicos** – Hello world

indizen
Technology & Business Confluence

```java
public class MyApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```


```scala
object MyApp extends App {

    println("Hello World!")

}
```

04 | **Sintaxis** – Sintaxis comparada con Java

indizen
Technology & Business Confluence

```java
public Boolean hasUpperCase(String word) {
    Boolean hasUpperCase = false;
    for (int i = 0; i < word.length(); i++) {
        if (Character.isUpperCase(word.charAt(i))) {
            return true;
        }
        i += 1;
    }
    return false;
}
```


```scala
def hasUpperCase(word:String): Boolean = {

  var wordHasUpperCase = false
  var i = 0
  while (i < word.length && ! wordHasUpperCase) {
    if (Character.isUpperCase (word.charAt(i))) {
      wordHasUpperCase = true;
    }
    i += 1;
  }
  return wordHasUpperCase
}
```


```
word.exists(x => x.isUpper)
```

```
word.exists(_.isUpper)
```

04 | **Sintaxis** - Variables

indizen
Technology & Business Confluence

> Dos tipos: val vs var
  - Val
    - Inmutable. No se puede reasignar.
  - Var
    - Mutable. Se puede reasignar

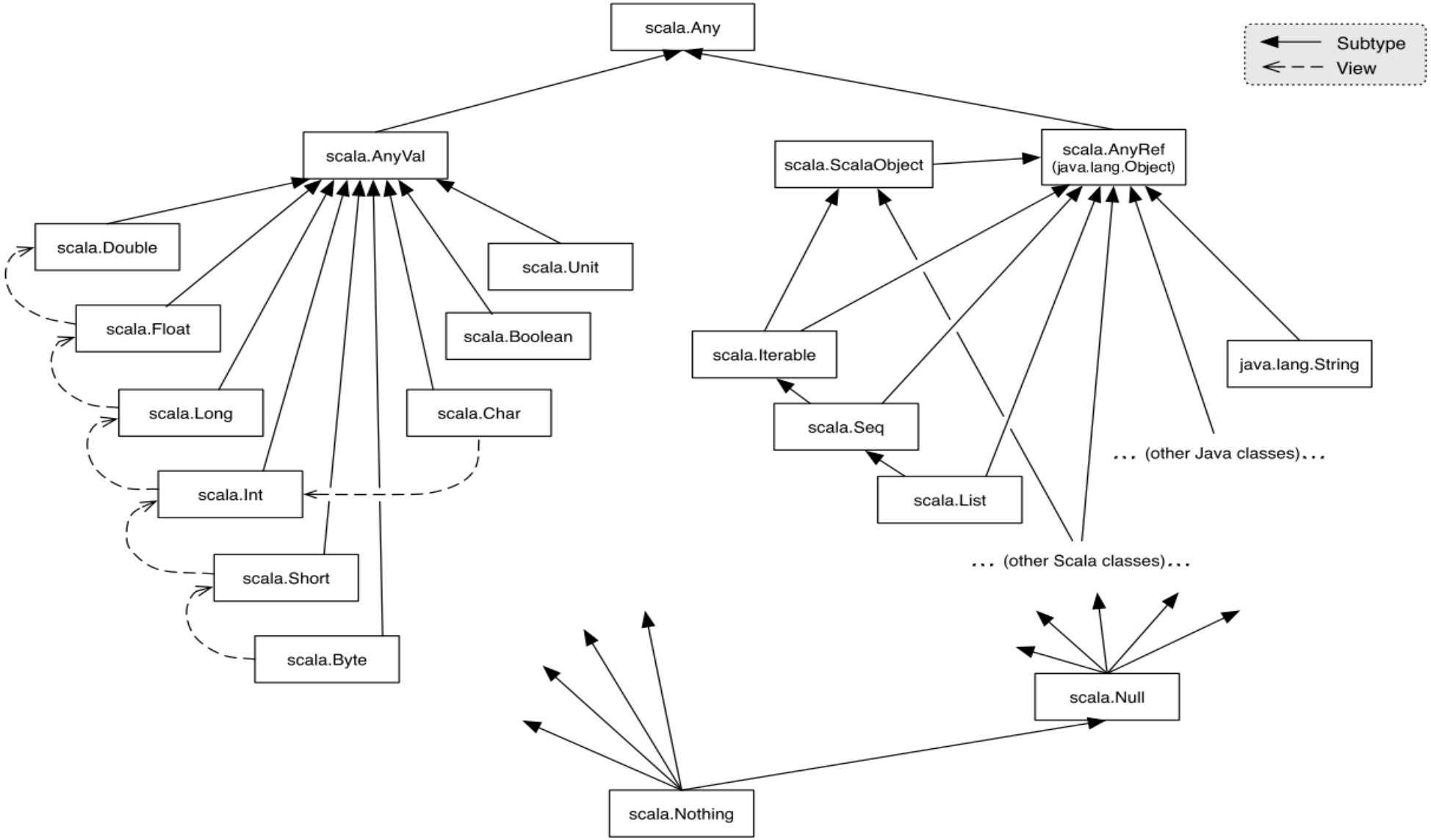
> Declaración
  - [var|val] nombre : [Tipo] = Valor

> No es necesario indicar el tipo, el compilador lo infiere

04 | **Sintaxis** - Tipos

indizen
Technology & Business Confluence

- **Any**: Supertipo.
  - Define toString equals, hashCode
- **AnyVal**: no nullabe, *represer* a los tipos primitivos de Jav
- **AnyRef**: Representa el Object de Java



04 | **Sintaxis –** Métodos y funciones

indizen
Technology & Business Confluence

- Declaración de métodos
  - Palabra reservada **def**
  - Los parámetros deben ir tipados
  - Tipo de salida opcional (de momento)
    - Unit: No devuelve nada (void)
  - Return opcional (no recomendado, obliga a tipar la salida)
  - Devuelve la última sentencia
- En scala, todo son objetos, incluso las funciones.
  - Ejemplo: (Int) => Boolean
  - Implica que se pueden declarar funciones como variables

- Ejemplo con def
  - def sum(x: Int, y: Int): Int = { return x+ y}
  - def sum(x: Int, y: Int) = x+ y

- Ejemplo con val
  - val sum3: (Int, Int)=> Int = (x,y) => x+y
  - val sum2 = (x: Int, y: Int) => x+y

- def vs val
  - Def se evalúa en cada llamada
  - Val se evalúa en la definición

indizen
Technology & Business Confluence

Ejercicio

> Apoyándote en la función sum definida anteriormente, define las funciones multiplicacion, resta y división

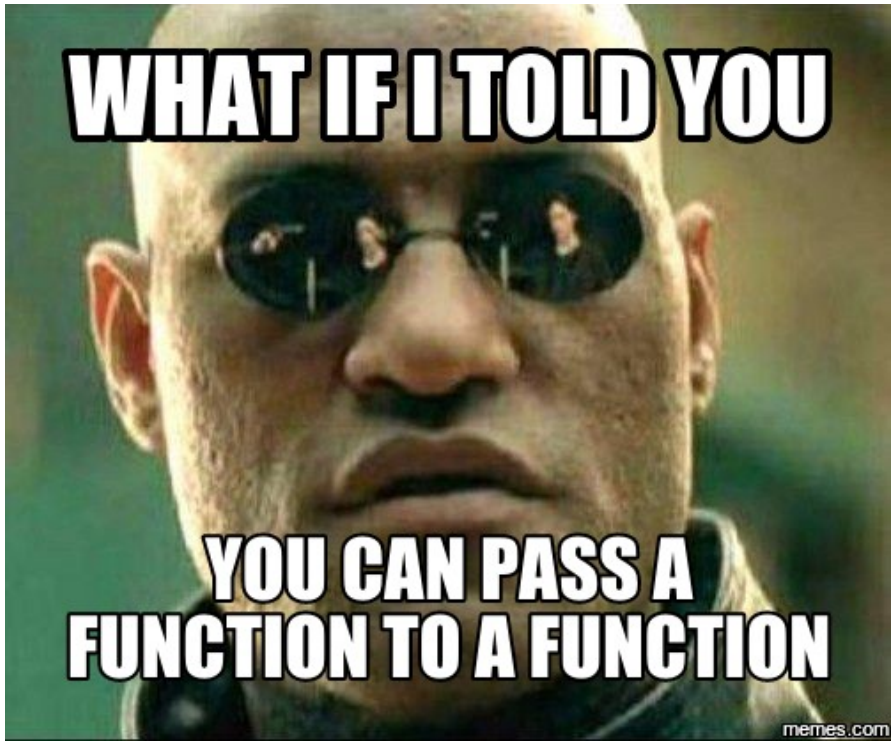
> ¿Tienen algo en común?
  - ¿Qué es lo que las diferencia?

> Sum
  - def sum(x: Int, y: Int) = x+ y

05 | **Funciones –** Orden superior

indizen
Technology & Business Confluence

> Se conoce como de orden superior a funciones que reciben como parámetro o devuelven como resultado otra función



> def operate(x: Int, y: Int, f:(Int, Int) => Int) = f(x,y)

> def createSum(x: Int) = (y: Int) => x+y

Ejercicio

> Generaliza createSum para que pueda generar cualquier operación

> Algunas conocidas
  - map, filter, flatMap, exists, takeWhile, dropWhile

indizen
Technology & Business Confluence

> Existe peligro de provocar error de pila si la llamada recursiva no se encuentra en la posición de cola

> Anotación *@tailrec* del paquete *scala.annotation*

> *Funciones nesteadas (Obligatorio tipo de salida)*

Ejemplos:

```scala
def wrongFactorial(n: Int) :Int  = {
  if (n <= 0) 1
  else n * factorial(n - 1)
}
```

```scala
def factorial(n: Int) :Int  = {

  @annotation.tailrec
  def rec(acc: Int, current: Int): Int = {
    if (current <= 0) acc
    else rec (acc*current, current -1)

  }
  rec ( acc= 1, n)
}
```

05 | **Funciones –** Más conceptos

indizen
Technology & Business Confluence

> Funciones **puras**: Sin efectos de lado

> Funciones **anónimas**: También conocidas como funciones literales.

- Ya las hemos usado en el ejemplo de generalización de la suma
- Permiten usar *atajos* de scala para reducir código escrito

> *Funciones* **parciales***:* Existen valores en el dominio donde no existe resultado (ej: División entre 0)

> Ejemplos anónimas:

- operate(1,2, (x: Int,y: Int) => x+y)
- operate(1,2, (x,y) => x+y)

> Funciones parciales

- Definen método IsDefinedAt

05 | **Funciones –** Parámetros por valor y por referencia

indizen
Technology & Business Confluence

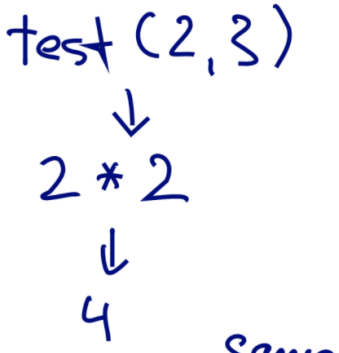
- En Scala los parámetros puedes pasarse por valor o por referencia
- Llamada por valor*(by-value)*
  - Evalúa el argumento **una única vez** (exactly once)
  - def callByValue(x: Int) = ???
- Llamada por referencia *(by-name)*
  - Se evalúa **todas y cada una de las veces que se usa**
  - **Si no se usa, no se evalúa**
  - def callByName(x: => Int) = ???

indizen
Technology & Business Confluence

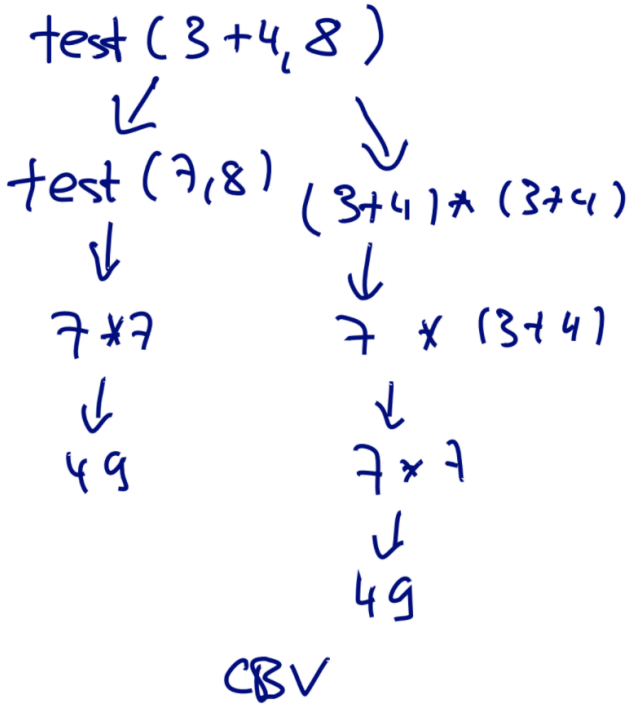
> ¿Cuándo usar uno u otro? Dependerá del caso.

Call-by-name vs call-by-value

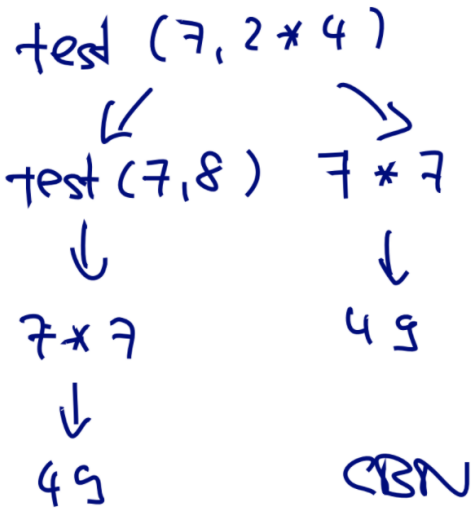
```
def test(x: Int, y: Int) = x * x

test(2, 3)
test(3+4, 8)
test(7, 2*4)
test(3+4, 2*4)
```



Same



CBV



CBN

06 | Colecciones – Tuplas

indizen
Technology & Business Confluence

> Colección innmutable de cualquier combinación de tipos
  - No se pueden modificar sus elementos
> Las representan las clases *Tuple2, Tuple3 ...Tuple22*
> Se accede a los elementos por posición empezando en 1 con underscore + posición
> Se pueden *descomponer*
> Definición:
  - val myTuple = (1, "Sergio", List(5, 4, 6))
  - Tipo: (Int, String, List[Int])
> Acceso:
  - myTuple._2 = “Sergio”
> Descomposición:
  - val (id, nombre, notas) = myTuple
  - val (id, nombre, _) = myTuple

06 | Colecciones – Listas

indizen
Technology & Business Confluence

- Colección inmutable
- *Nil* representa lista vacía.
- Se accede a los elementos por posición empezando en 0
- Añadir elementos:
  - Al comienzo => **::** o **+:**
  - Al final => :+
  - Concatenar listas => **:::**
- Ofrece funciones como: head, tail, reverse, dropRight, take, foreach, mkString, contains…

- Definición:
  - val myList = List(1, 2, 3)
  - Tipo: List[Int]
- Acceso:
  - MyList(2) = 3
- Añadir elementos:
  - 6::myList => List(6,1,2,3)
  - 6 +: myList || myList.+:(6)=> List(6,1,2,3)
  - myList :+ 6 || myList.:+(6) => List(1,2,3, 6)

07 | Pattern Matching

indizen
Technology & Business Confluence

- Comprobación de valores contra un patrón, el patrón debe ser completo
- *Similar* al switch de java
- Caso por defecto => underscore
- Permite descomponer las partes.
- Permite incluir ifs dentro del propio case:
  - case y if y > 5 => ???
- Con tuplas:
  - case (1, “Sergio”, _) =>
- Con listas:
  - case h::tail =>

Ejemplo:

```
x match {
  case 1 => “one”
  case 2 => “two”
  case _ => “default”
}
```

07 | Pattern Matching - Ejemplos

Ejemplo1

```
(“Jose”, “Sergio”, “Lourdes”) match {
        case (_, "Jose", _) => "Jose"
        case (“Sergio”, “Sergio”, “Lourdes”) => “Sergio”
        case (“Jose”, _, _ ) => “Lourdes”
        case _ ⇒ “default”
}
```

- **Ejemplo2**

```
List(1, 2, 3) match {
        case x :: xs ⇒ xs.head
        case _ ⇒ 0
}
```

- **Ejemplo 3**

```
def patternMatching(list: List[Int]): Int = {
  list match {
    case x :: xs if x == 2 => xs.head
    case x :: xs => patternMatching(xs)
    case _ ⇒ 0
  }
}
```

- patternMatching(List(1,2,3))
- patternMatching(Nil)

| Ejercicios

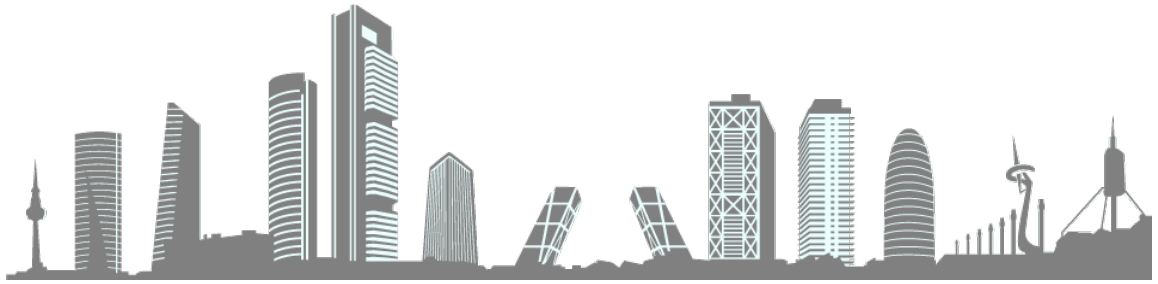
indizen
Technology & Business Confluence

- La función operate creada en esta sesión, que generaliza las operaciones sólo es válida para enteros, podrías generalizarla:
  - def operate[A]…
  - def operate[A, B]
- Define una función recursiva que devuelva el máximo de una lista de enteros
  - def max(list: List[Int]): Int = ???
- Función que devuelve el segundo elemento de una lista
  - def second(list: List[Int]) : Int = ???
- Función que devuelva el nth elemento de una lista
  - def nth(list: List[Int], n: Int): Int = ???


"Con
Indizen
nunca caminarás solo"
indizen
Technology & Business Confluence

indizen

Technology & Business Confluence



GRACIAS

www.indizen.com