



Scala

CARACTERISTICAS

- Orientación a objetos y funcional
- Interoperabilidad con Java
- Inmutabilidad y colecciones inmutables
- Actores y concurrencia
- *Pattern matching*

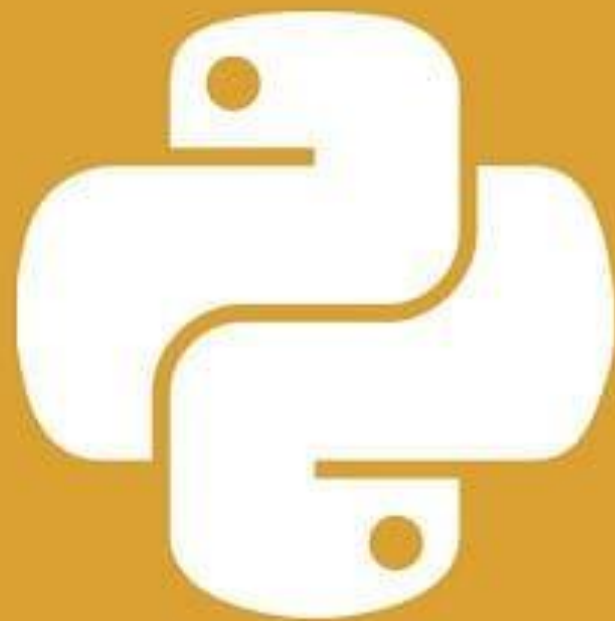
CASOS DE USO






SCALA

VS



PYTHON

 Editor Save New Format Clear Messages Worksheet ● Download Embed Build Settings

```
1 case class Message(sender: String, recipient: String, body: String) {  
2   def send = s"$sender → $recipient: $body"  
3 }  
4 val m = Message("Alice", "Bob", "Ça va ?")  
5 m.send Alice → Bob: Ça va ? : String  
6 println(s"$m sent") (): Unit
```

>_ Console (F3)

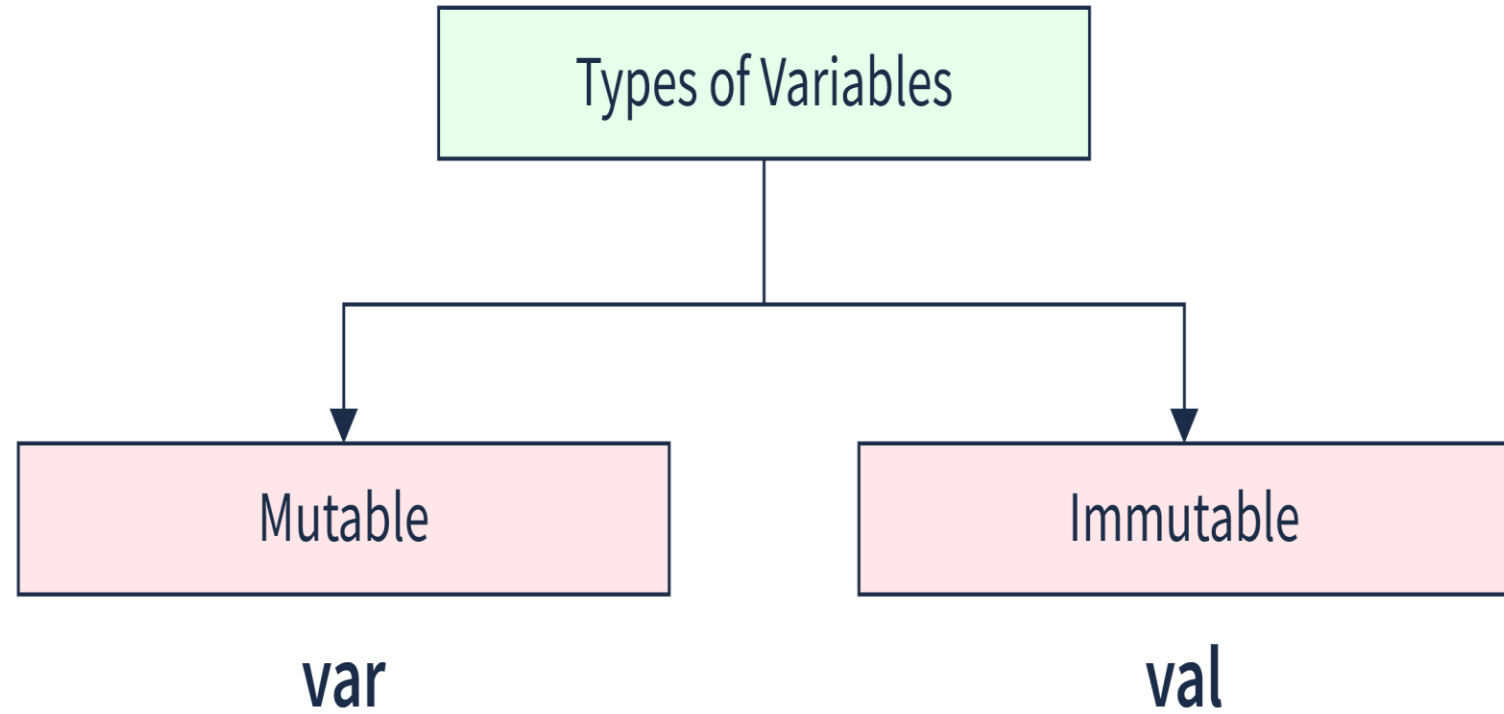
```
Message(Alice,Bob,Ça va ?) sent
```

 Dark Help Up

ENTORNO DE DESARROLLO

The logo for Scalac, featuring a red greater-than sign followed by the word "scalac" in a bold, dark gray sans-serif font.

> scalac



DECLARANDO VARIABLES

```
val x: Int = 1  
// explicito
```

```
val x = 1 //  
implicito
```


TIPOS DE DATOS

- `val b: Byte = 1`
- `val i: Int = 1`
- `val l: Long = 1`
- `val s: Short = 1`
- `val d: Double = 2.0`
- `val f: Float = 3.0`

Data Type	Possible Values
Boolean	true or false
Byte	8-bit signed two's complement integer (-2^7 to 2^7-1 , inclusive) -128 to 127
Short	16-bit signed two's complement integer (-2^{15} to $2^{15}-1$, inclusive) -32,768 to 32,767
Int	32-bit two's complement integer (-2^{31} to $2^{31}-1$, inclusive) -2,147,483,648 to 2,147,483,647
Long	64-bit two's complement integer (-2^{63} to $2^{63}-1$, inclusive) (-2^{63} to $2^{63}-1$, inclusive)
Float	32-bit IEEE 754 single-precision float 1.40129846432481707e-45 to 3.40282346638528860e+38
Double	64-bit IEEE 754 double-precision float 4.94065645841246544e-324 to 1.79769313486231570e+308
Char	16-bit unsigned Unicode character (0 to $2^{16}-1$, inclusive) 0 to 65,535
String	a sequence of Char

INTERPOLANDO STRINGS

- `val firstName = "John"`
- `val mi = 'C'`
- `val lastName = "Doe"`
- `println(s"Name: $firstName $mi $lastName")`

EJERCICIO

- Escribe un programa en Scala que solicite al usuario que introduzca su nombre, edad y ciudad. Luego, utiliza la interpolación de strings para crear un mensaje personalizado que incluya esta información y lo muestre por pantalla. Asegúrate de importar la librería `scala.io.StdIn` para poder leer la entrada del usuario de forma adecuada. El mensaje final debe tener el siguiente formato: "¡Hola [nombre]! Tienes [edad] años y vives en [ciudad]."



Kahoot!

A close-up photograph of several dark wooden chess pieces on a checkered board. The pieces include a king, a queen, a bishop, and several pawns. One white piece is lying on its side in the foreground. The background is out of focus, showing warm, circular bokeh lights. The text "ESTRUCTURAS DE CONTROL" is overlaid in white, bold, sans-serif capital letters at the bottom left.

ESTRUCTURAS DE CONTROL

IF/ELSE

```
if x < 0 then  
    println("negative")  
else if x == 0 then  
    println("zero")  
else  
    println("positive")
```

FOR

```
val ints = List(1, 2, 3, 4, 5)
```

```
for i <- ints do println(i)
```

```
scala> val doubles = for i <- ints yield i * 2  
val doubles: List[Int] = List(2, 4, 6, 8, 10)
```

WHILE

```
var x = 1  
  
while  
    x < 3  
do  
    println(x)  
    x += 1
```


MATCH

```
// getClassAsString is a method that takes a single argument of any type.
def getClassAsString(x: Matchable): String = x match
  case s: String => s"'$s' is a String"
  case i: Int    => "Int"
  case d: Double => "Double"
  case l: List[?] => "List"
  case _        => "Unknown"

// examples
getClassAsString(1)           // Int
getClassAsString("hello")     // 'hello' is a String
getClassAsString(List(1, 2, 3)) // List
```

```
val result = i match
  case 1 => "one"
  case 2 => "two"
  case _ => "other"
```

EJERCICIO

Suma de números pares e impares: Escribe un programa que sume todos los números pares e impares en un rango dado por el usuario y luego imprima la suma de los números pares y la suma de los números impares.

EJERCICIO

Verificación de número primo: Escribe un programa que verifique si un número ingresado por el usuario es primo o no.

Kahoot!



POO



CLASES

```
class Person(var name: String, var vocation: String)
class Book(var title: String, var author: String, var year: Int)
class Movie(var name: String, var director: String, var year: Int)
```

```
class Person(var firstName: String, var lastName: String):

    println("initialization begins")
    val fullName = firstName + " " + lastName

    // a class method
    def printFullName: Unit =
        // access the `fullName` field, which is created above
        println(fullName)

    printFullName
    println("initialization ends")
```

OBJETOS

```
object StringUtils:  
  def truncate(s: String, length: Int): String = s.take(length)  
  def containsWhitespace(s: String): Boolean = s.matches(".*\\s.*")  
  def isEmptyOrNull(s: String): Boolean = s == null || s.trim.isEmpty
```

EJERCICIO

- Crea un objeto calculadora y utilízalo para sumar, restar, multiplicar y dividir.

COMPANION OBJECT

```
import scala.math.*

class Circle(val radius: Double):
  def area: Double = Circle.calculateArea(radius)

object Circle:
  private def calculateArea(radius: Double): Double = Pi * pow(radius, 2.0)

val circle1 = Circle(5.0)
circle1.area
```

TRAITS

```
trait HasTail:  
  def tailColor: String  
  def wagTail() = println("Tail is wagging")  
  def stopTail() = println("Tail is stopped")
```

ENUMS

```
enum CrustSize:  
    case Small, Medium, Large
```

```
enum CrustType:  
    case Thin, Thick, Regular
```

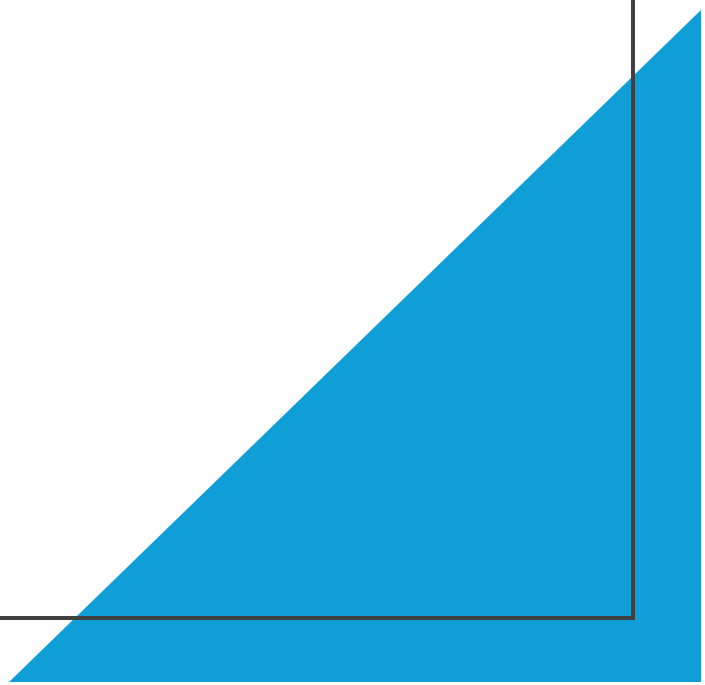
```
enum Topping:  
    case Cheese, Pepperoni, BlackOlives, GreenOlives, Onions
```

EJERCICIO

- Crea un trait llamado Conducible que representa la capacidad de ser conducible. Este trait tiene un método conducir() que imprime "¡Vehículo a motor en movimiento!" por defecto.
- Crea una clase Vehiculo que acepta un parámetro de tipo marca en su constructor y mezcla el trait Conducible.
- Crea una clase llamada Bicicleta, que también mezcla el trait Conducible. Sin embargo, la implementación del método conducir() en la clase Bicicleta debe ser sobrescrita para imprimir "¡La bicicleta está en movimiento sin motor!".
- Crea instancias de Vehiculo y Bicicleta e invoca el método conducir() en ambas instancias, mostrando las salidas correspondientes.

Kahoot!

METODOS



```
// DEFINO UN METODO
```

```
def add(a: Int, b: Int): Int = a + b
```

```
// LO UTILIZO
```

```
val x = add(1, 2) // 3
```

```
val x2 = List(1, 2, 3)

x2.size 3: scala.Int           // 3
x2.contains(1) true: scala.Boolean // true
x2.map(_ * 10) List(10, 20, 30): scala.collection.immutable.List[scala.Int] // List(10, 20, 30)
```

METODOS PRECONSTRUIDOS

PARAMETROS POR DEFECTO

```
def makeConnection(timeout: Int = 5_000, protocol: String = "http") =  
    println(f"timeout = ${timeout}%d, protocol = ${protocol}%s")  
  
makeConnection() (): scala.Unit           // timeout = 5000, protocol = http  
makeConnection(2_000) (): scala.Unit      // timeout = 2000, protocol = http  
makeConnection(3_000, "https") (): scala.Unit // timeout = 3000, protocol = https
```

VISIBILIDAD DE LOS METODOS

- PRIVADO
- PUBLICO
- PROTECTED



EJERCICIO

- Define una clase base llamada Animal.
- En la clase Animal, define un método llamado hacerSonido() que imprime "El animal hace un sonido". ¿Qué visibilidad debe tener? No quiero que sea accesible desde el exterior
- Define una subclase de Animal llamada Perro.
- En la subclase Perro, define un método llamado hacerRuido() que llama al método protegido hacerSonido(). ¿Qué visibilidad debe tener?
- Crea un objeto de la clase Perro y llama al método hacerRuido() para verificar que se imprima "El animal hace un sonido".

LAMBIDAS

```
val doubledInts = ints.map((i: Int) => i * 2)
```

```
val doubledInts = ints.map(_ * 2)    // List(2, 4, 6)
```

```
val double = (i: Int) => i * 2  
  
// EL NOMBRE ES DOUBLE Y AHORA LA PUEDO INVOCAR  
  
val x = double(2)
```

FUNCIONES DECLARADAS

ETA EXPANSION





EJERCICIO

- Dada una lista de temperaturas en grados Celsius, conviértelas a grados Fahrenheit y crea una nueva lista con los resultados.
- Dada una lista de palabras, filtra todas las palabras que tengan más de 5 letras y crea una nueva lista con ellas.
- Define una función llamada `operateOnList` que toma una lista de enteros `lst` y una función `f` que toma un entero y devuelve otro entero. La función `operateOnList` debe aplicar la función `f` a cada elemento de la lista `lst` y devolver una nueva lista con los resultados.

Kahoot!

Collection Type	Immutable	Mutable	Description
List	✓		A linear (linked list), immutable sequence
Vector	✓		An indexed, immutable sequence
LazyList	✓		A lazy immutable linked list, its elements are computed only when they're needed; Good for large or infinite sequences.
ArrayBuffer		✓	The go-to type for a mutable, indexed sequence
ListBuffer		✓	Used when you want a mutable List; typically converted to a List
Map	✓	✓	An iterable collection that consists of pairs of keys and values.
Set	✓	✓	An iterable collection with no duplicate elements

SEQUENCE

Type/Category	Immutable	Mutable
Indexed	Vector	ArrayBuffer
Linear (Linked lists)	List	ListBuffer

EJERCICIO

Escribe un programa Scala para encontrar el producto máximo de dos números enteros en un Array dado de números enteros.

- Ejemplo:
- números = {2, 3, 5, 7, -7, 5, 8, -5}
- Resultado:
- El par es (7, 8) y el producto máximo: 56

EJERCICIO

- Escribe un programa para eliminar los elementos duplicados de Array y devolver la nueva longitud del mismo.
- Array de muestra: [20, 20, 30, 40, 50, 50, 50, 50, 60, 60]
- Después de eliminar los elementos duplicados, el programa debería devolver 5 como la nueva longitud del Array.

Kahoot!

METODOS DE COLECCIONES

```
val a = List(10, 20, 30, 40, 10)    // List(10, 20, 30, 40, 10)

a.distinct                          // List(10, 20, 30, 40)
a.drop(2)                           // List(30, 40, 10)
a.dropRight(2)                      // List(10, 20, 30)
a.head                              // 10
a.headOption                        // Some(10)
a.init                             // List(10, 20, 30, 40)
a.intersect(List(19,20,21))         // List(20)
a.last                             // 10
a.lastOption                        // Some(10)
a.slice(2,4)                        // List(30, 40)
a.tail                             // List(20, 30, 40, 10)
a.take(3)                           // List(10, 20, 30)
a.takeRight(2)                      // List(40, 10)
```

REDUCE

```
scala> a.reduce(_ + _)  
res0: Int = 10
```

EJERCICIO



- Crea una lista de nombres de personas.
- Filtra la lista para obtener solo los nombres que comienzan con la letra "A".
- Mapea la lista filtrada para convertir cada nombre a mayúsculas.
- Imprime la lista final de nombres.

EJERCICIO

- Crea un string con varias palabras separadas por espacios.
- Utiliza reduce para contar el número de palabras en el string.

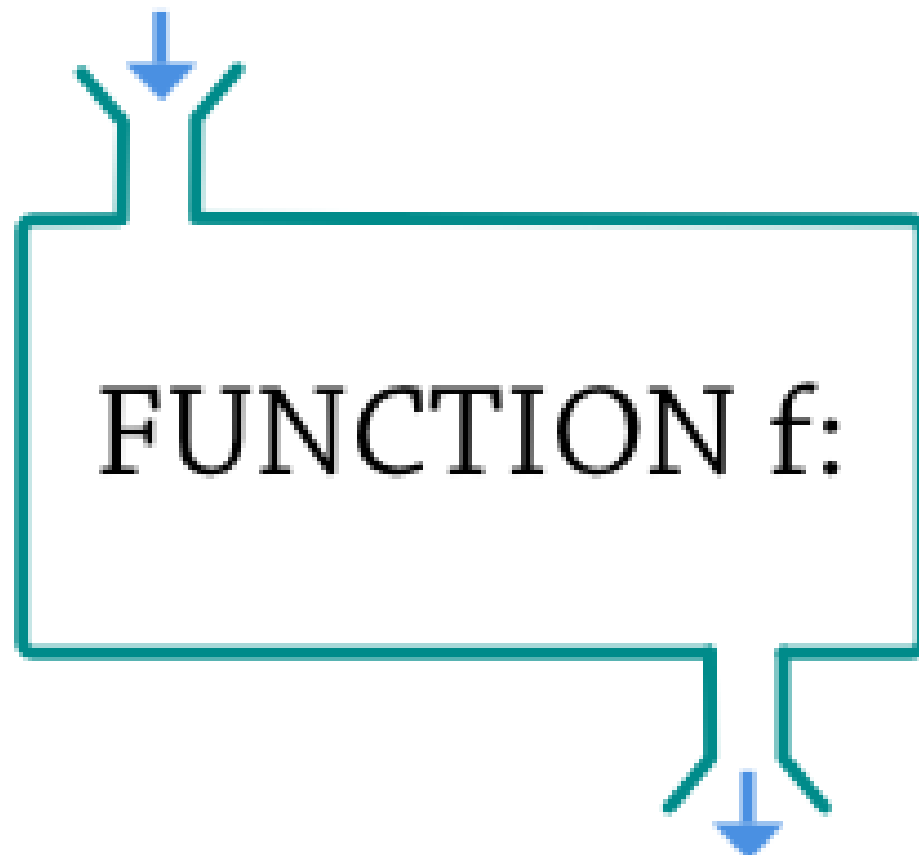




EJERCICIO

- Define un Map llamado inventario que contenga algunos productos y sus cantidades iniciales. Por ejemplo:
- `val inventario = Map("Camisa" -> 10, "Pantalón" -> 5, "Zapatos" -> 8)`
- Escribe una función llamada `agregarProducto` que tome tres parámetros: el inventario actual, el nombre del producto y la cantidad a agregar. Esta función debe devolver un nuevo Map que represente el inventario actualizado después de agregar la cantidad especificada del producto dado.
- Escribe una función llamada `venderProducto` que tome tres parámetros: el inventario actual, el nombre del producto y la cantidad a vender. Esta función debe devolver un nuevo Map que represente el inventario actualizado después de vender la cantidad especificada del producto dado. Asegúrate de manejar los casos en los que el producto no esté disponible en el inventario o la cantidad a vender sea mayor que la cantidad disponible.
- Escribe una función llamada `mostrarInventario` que tome el inventario actual como parámetro y muestre el nombre de cada producto junto con su cantidad disponible.

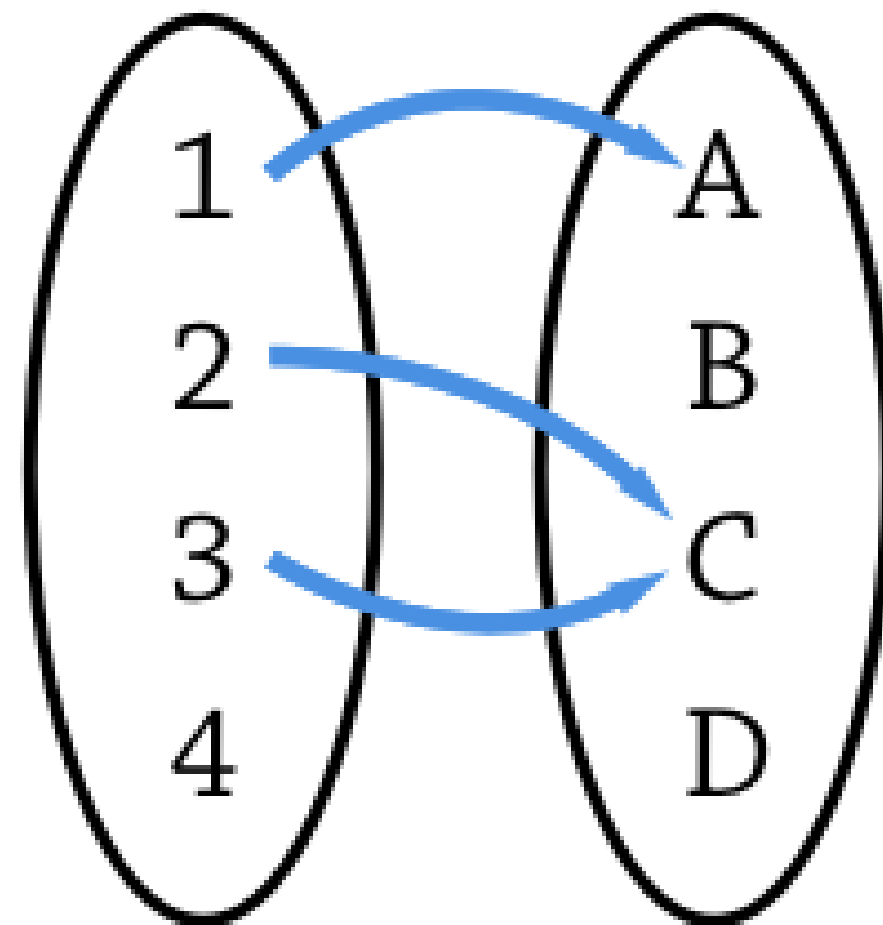
INPUT x



OUTPUT $f(x)$

x

y

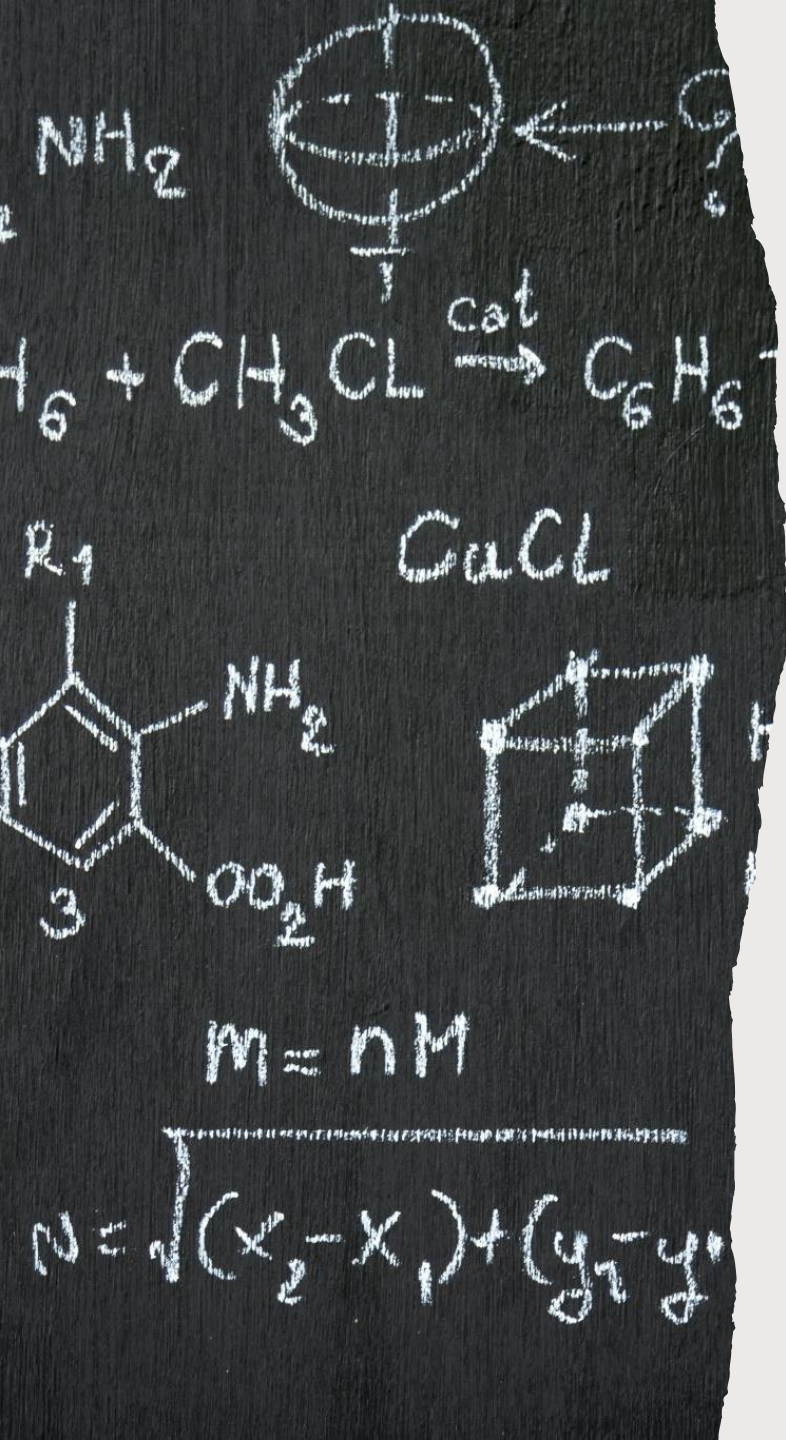


REGLAS

- En programación funcional pura, sólo se utilizan valores inmutables. En Scala esto significa:
- Todas las variables se crean como valcampos.
- Solo se utilizan clases de colecciones inmutables, como List, y las clases Vector inmutables Map y Set

FUNCIONES PURAS

- Otra característica que ofrece Scala para ayudarle a escribir código funcional es la capacidad de escribir funciones puras. Una función pura se puede definir así:
- Una función f es pura si dada la misma entrada x siempre devuelve la misma salida $f(x)$
- La salida de la función depende sólo de sus variables de entrada y su implementación.
- Solo calcula la salida y no modifica el mundo que lo rodea.
- Esto implica:
- No modifica sus parámetros de entrada.
- No muta ningún estado oculto.
- No tiene “puertas traseras”: no lee datos del mundo exterior (incluida la consola, servicios web, bases de datos, archivos, etc.) ni escribe datos en el mundo exterior.



FUNCIONES PURAS

- Dada esa definición, como puedes imaginar, métodos como estos en el paquete `scala.math._` son funciones puras:
- `Abs` `Ceil` `max`
- Estos métodos de `String` también son funciones puras:
- `isEmpty` `length` `Substring`
- La mayoría de los métodos de las clases de colecciones de `Scala` también funcionan como funciones puras, incluidas `drop`, `filter`, `map` y muchas más.

FUNCIONES IMPURAS

- Por el contrario, las siguientes funciones son impuras porque violan la definición.
- `println`– Los métodos que interactúan con la consola, archivos, bases de datos, servicios web, sensores, etc., son todos impuros.
- `currentTimeMillis` – Los métodos relacionados con la fecha y la hora son todos impuros porque su salida depende de algo más que sus parámetros de entrada.
- `sys.error`– Los métodos de lanzamiento de excepciones son impuros porque no devuelven simplemente un resultado.



GESTION DE ERRORES

```
def makeInt(s: String): Int =  
  try  
    Integer.parseInt(s.trim)  
  catch  
    case e: Exception => 0
```




EJERCICIO

- Considera un escenario en el que tienes una lista de empleados con sus salarios, pero algunos salarios podrían faltar. Tu tarea es calcular el salario promedio de los empleados, manejando los valores faltantes de manera adecuada utilizando el tipo Option.
- Define una case class Empleado con dos campos: nombre: String y salario: Option[Double].
- Crea una lista de instancias de Empleado, incluyendo algunas con salarios faltantes (None).
- Escribe una función para calcular el salario promedio de todos los empleados, manejando los salarios faltantes adecuadamente.
- Prueba tu función con varios escenarios, incluyendo casos donde todos los salarios estén disponibles y casos donde algunos falten.

Kahoot!



LABORATORIO