UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN

Scala

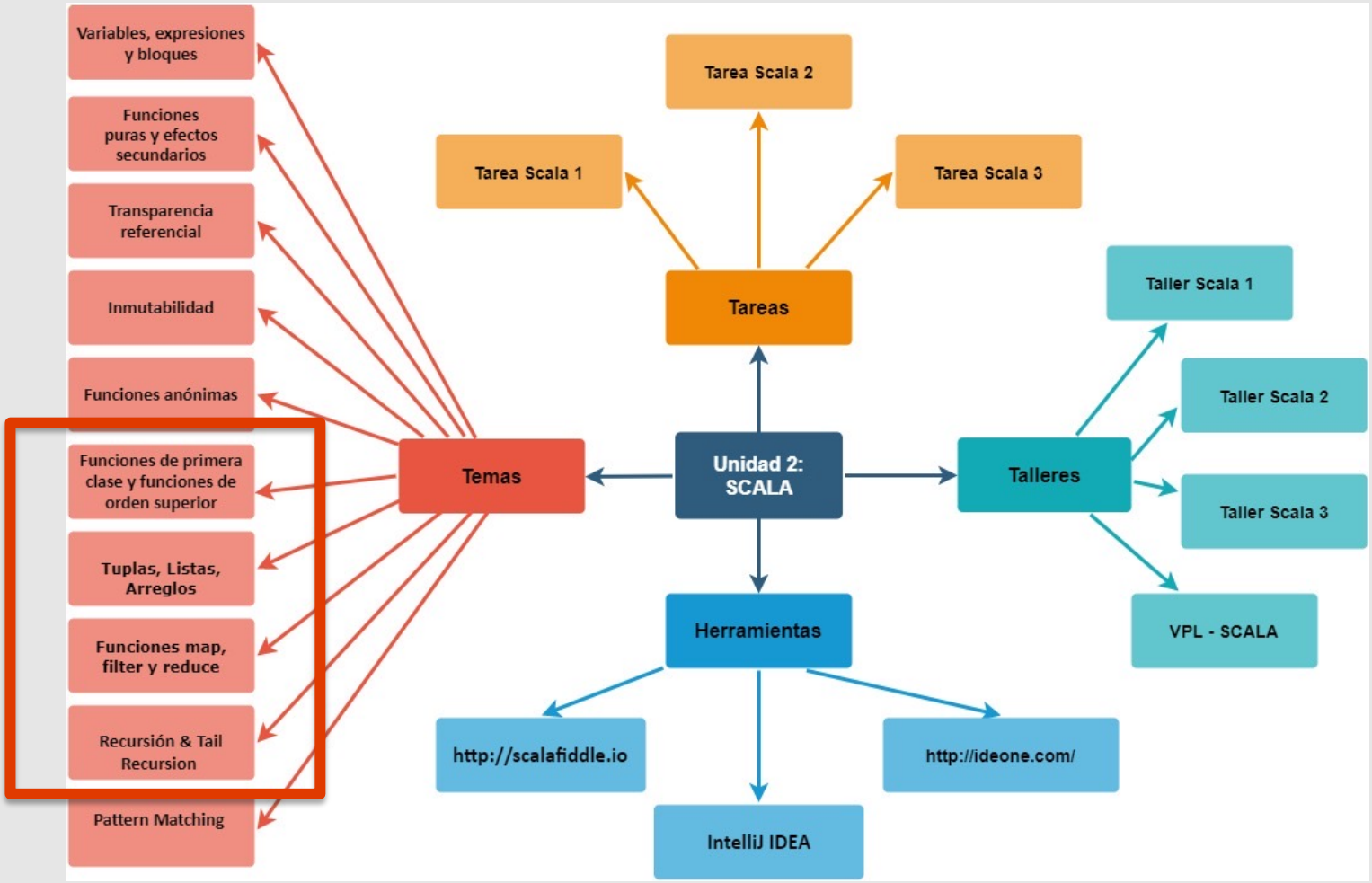
Prof. Oscar Mauricio Salazar Ospina
omsalazaro@unal.edu.co

Facultad de Minas
Departamento de Ciencias de la Computación y la Decisión
Facultad de Minas

Universidad Nacional de Colombia
PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN

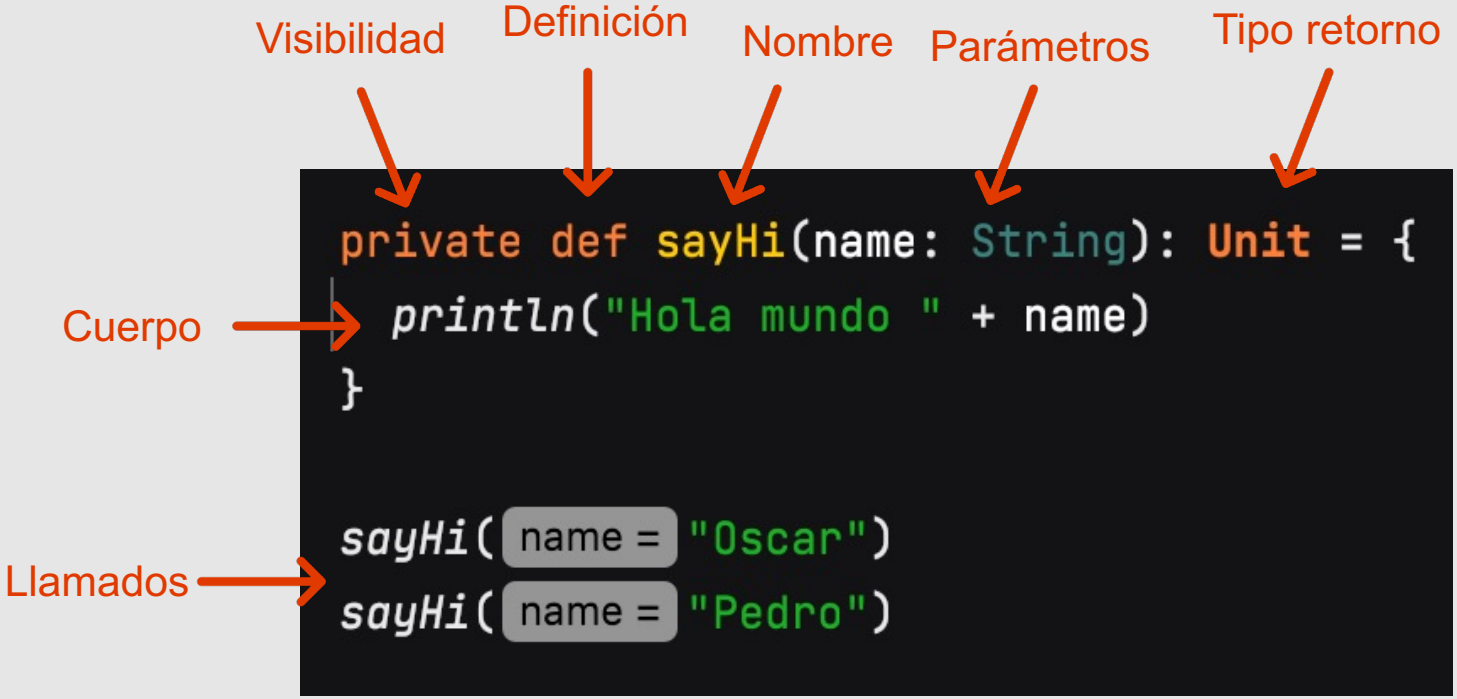
Unidad 3: Scala

Mapa conceptual



Scala

Como definir una función



Resultado →

```
Hola mundo Oscar
Hola mundo Pedro
```

Scala

Funciones de primera clase

Si una función se puede **tratar como un valor**, se dice que la función es de primera clase.

- Es decir que se puede asignar a una constante/variable.
- Se puede usar como el valor de entrada de otra función.
- Se puede usar como el valor de salida o respuesta de otra función.

Scala

Funciones de primera clase

```
val x1 = scala.math.pow(8,2)
```

```
val v = scala.math.sqrt(12 + 4)
```

```
val z = (x:Int, y:Int) => x + y
```

```
def elDoble(i: Int): Int = {return i * 2}
val f1 = elDoble(4)
```

```
def elDoble(i: Int): Int = {return i * 2}
val r = 1 to 10
val d = r.map(elDoble)
println(d)
```

Una función es de orden superior **si hace al menos una de las siguientes cosas:**

- **Recibe una o más funciones** como argumentos.
- **Retorna como resultado** una función.

Scala

Funciones de orden superior

- ¿Son las funciones **map** y **filter** de orden superior?

```
val elTriple = (i: Int) =>  i * 3
val f1 = elTriple(4)

val r = 1 to 10 // rango de datos de 1 a 10 inclusive
println(r.map(elTriple))
```

```
val r = 1 to 10 // rango de datos de 1 a 10 inclusive

val par = (i: Int) => i % 2 == 0
println(r.filter(par))

val impar = (i: Int) => i % 2 != 0
println(r.filter(impar))
```

Scala

Ejercicio

- Utilizar **filter** para obtener los múltiplos de 3 en un array del 1-100
- Utilizar **map** para encontrar las mitades de un array 1-10
- Utilizar **find** para encontrar un elemento definido y elevarlo al cuadrado

Scala

Ejercicio

- Utilizar **filter** para obtener los múltiplos de 3 en un array del 1-100

```
val r = 1 to 100
println(r.filter(e => e %  3 == 0))
```

- Utilizar **map** para encontrar las mitades de un array 1-10

- Utilizar **find** para encontrar un elemento definido y elevarlo al cubo

Scala

Ejercicio

- Utilizar **filter** para obtener los múltiplos de 3 en un array del 1-100

```
val r = 1 to 100
println(r.filter(e => e %  3 == 0))
```

- Utilizar **map** para encontrar las mitades de un array 1-10

```
print(r.map(element => element / 2.0))
```

- Utilizar **find** para encontrar un elemento definido y elevarlo al cubo

Scala

Ejercicio

- Utilizar **filter** para obtener los múltiplos de 3 en un array del 1-100

```
val r = 1 to 100
println(r.filter(e => e %  3 == 0))
```

- Utilizar **map** para encontrar las mitades de un array 1-10

```
print(r.map(element => element / 2.0))
```

- Utilizar **find** para encontrar un elemento definido y elevarlo al cubo

```
val x: Int = it.find(n => n == 3).get
print(math.pow(x, 3))
```

Scala

Ejemplo función de orden superior

```scala
def imprimir(v: Any): Unit = {
  print("Resultado: " + v)
}

def componer(f: Double => Double, g: Double => Double, v: Double): Double = g(f(v))
def seno(s: Double): Double = Math.sin(s)
def coseno(c: Double): Double = Math.cos(c)
imprimir(componer(seno, coseno, 5))
```

Scala

Ejemplo función de orden superior

¿Qué hace?

```scala
def imprimir(v: Any): Unit = {
  print("Resultado: " + v)
}

val r = 1 to 10
def componer(f: Double => Double, g: Double => Double, v: Double): Double = g(f(v))
def seno(s: Double): Double = Math.sin(s)
def coseno(c: Double): Double = Math.cos(c)
imprimir(r.map(e => componer(seno, coseno, e)))
```

```scala
imprimir(r.map(componer(seno, coseno, _)))
```

Ejemplo función de orden superior

¿Qué hace?

```
// Función "p" reemplaza println
def p(a:Any): Any = {
    println(a+"")
}
```

```
def multiplo_seis(f:Int => Boolean, g:Int => Boolean, i:Int)= f(i) & g(i)
val multiplo_dos = (i:Int) => if (i % 2 == 0) true else false
val multiplo_tres = (i:Int) => if (i % 3 == 0) true else false
p(multiplo_dos(6))
p(multiplo_tres(6))
p(multiplo_seis(multiplo_dos,multiplo_tres,6))
```

Scala

Listas

```
println("Lista con 3 Elementos")
var listaFlores: List[String] = List("Rosa","Tulipán","Clavel")
println(s"Listado de Flores = $listaFlores")
//Acceso a los Elementos
println(s"Elemento 1 = ${listaFlores(0)}")
println(s"Elemento 2 = ${listaFlores(1)}")
println(s"Elemento 3 = ${listaFlores(2)}")
//Agregar Elementos
var listaFlores2: List[String] = listaFlores :+ "Dahlia"
listaFlores = listaFlores2
println(s"Agregando Elementos usando :+ = $listaFlores")
```

```
Lista con 3 Elementos
Listado de Flores = List(Rosa, Tulipán, Clavel)
Elemento 1 = Rosa
Elemento 2 = Tulipán
Elemento 3 = Clavel
Agregando Elementos usando :+ = List(Rosa, Tulipán, Clavel, Dahlia)
```

Scala

::: , :: , :+ en listas

```scala
val lista1: List[Int] = List(1, 2, 4, 5, 6)
val lista2: List[Int] = List(7, 8)

println(lista1 ::: List())
println(lista1 ::: lista2)
println(0 :: lista1)
println(5 :: Nil)
println(lista1 :+ 7)
println("cabeza=" + lista1.head + " cola= " + lista1.tail)
```

Scala

::: , :: , :+ en listas

```
val lista1: List[Int] = List(1, 2, 4, 5, 6)
val lista2: List[Int] = List(7, 8)

println(lista1 ::: List())
println(lista1 ::: lista2)
println(0 :: lista1)
println(5 :: Nil)
println(lista1 :+ 7)
println("cabeza=" + lista1.head + " cola= " + lista1.tail)
```

```
List(1, 2, 4, 5, 6)
List(1, 2, 4, 5, 6, 7, 8)
List(0, 1, 2, 4, 5, 6)
List(5)
List(1, 2, 4, 5, 6, 7, 8, oscar)
cabeza=1 cola= List(2, 4, 5, 6)
```

Ejercicio: Contar los elementos de una lista recursivamente

```scala
val lista: List[Int] = List(1, 2, 4, 5, 6, 8)

def numeroElementos(lista: List[Int]): Int =
  if (lista.isEmpty) 0 else 1 + numeroElementos(lista.tail)

println("# elem: " + numeroElementos(lista))
```

Ejercicio: Sumar los elementos???

Insertar elemento en una lista ordenada ???

Insertar elemento en una lista ordenada ???

```
def insercionOrdenada(elemento: Int, lista: List[Int]): List[Int] =
  if (lista.isEmpty) elemento :: Nil else
    if(elemento < lista.head)  elemento :: lista else
    lista.head :: insercionOrdenada(elemento, lista.tail)

println("Lista: " + insercionOrdenada( elemento = 3, lista))
```

Ordenar una lista utilizando la inserción???

Ordenar una lista utilizando la inserción???

```scala
def ordenarLista(lista: List[Int]): List[Int] =
  if (lista.isEmpty) Nil else insercionOrdenada(lista.head, ordenarLista(lista.tail))

println("Lista Ord: " + ordenarLista(List(2, 2, 1, 8, 29, 6)))
```

Invertir una lista ordenada ???

Invertir una lista ordenada ???

```scala
def invertirLista(lista: List[Int]): List[Int] =
  if (lista.isEmpty) lista else invertirLista(lista.tail) ::: List(lista.head)

println("Lista: " + invertirLista(lista))
```

Scala

Tuplas

```scala
val persona1 = ("Pedro", "Perez", 38)
val persona2 = ("Laura", "Manrique", 16)
val persona3 = ("Hugo", "Sanchez", 64)

println(s"Persona 1: ${persona1._1} ${persona1._2}")

val (nombre, apellido, _) = persona2
println(s"Persona 2: ${nombre} ${apellido}")
```

Scala

Tuplas - Iteración

```scala
persona2.productIterator.foreach { propiedad =>
  println(s"Propiedad: $propiedad")
}

persona2.productIterator.zipWithIndex.foreach { case (item, index) =>
  println(s"Prpiedad $index: $item")
}
```

Tuplas – Iteración con case

```scala
val puntos = List(
  ("Marcela", 1000),
  ("Ana", 20),
  ("Pedro", 10),
  ("Orlando", 0),
)

puntos.foreach {
  case (persona, 0) => println(s"$persona -> No tienes puntos para comprar")
  case (persona, 1000) => println(s"$persona -> Puedes comprar el reloj")
  case _ => println(s"Continua comprando para acumular puntos")
}
```

Scala

Ejercicio

- Crear una lista de estudiantes (nombre, apellido, nota, curso)
- Imprimir toda la lista de estudiantes
- Imprimir toda la lista de estudiantes ordenados por nota
- Encontrar estudiantes que ganaron el curso de matemáticas
- Aplicar bonificación del 20% de la nota para estudiantes con notas inferiores a 2.2 en el curso de español

Gracias