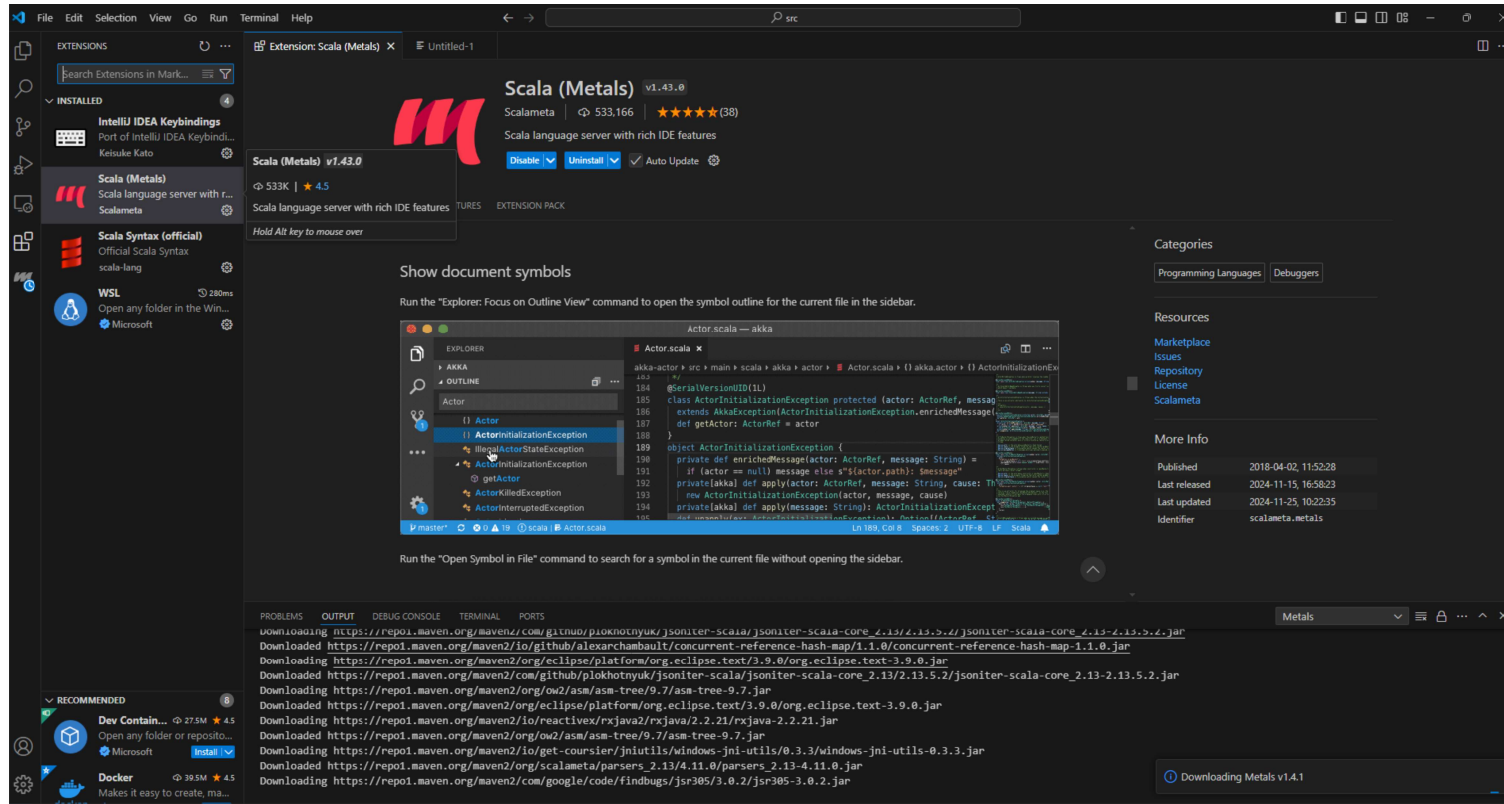




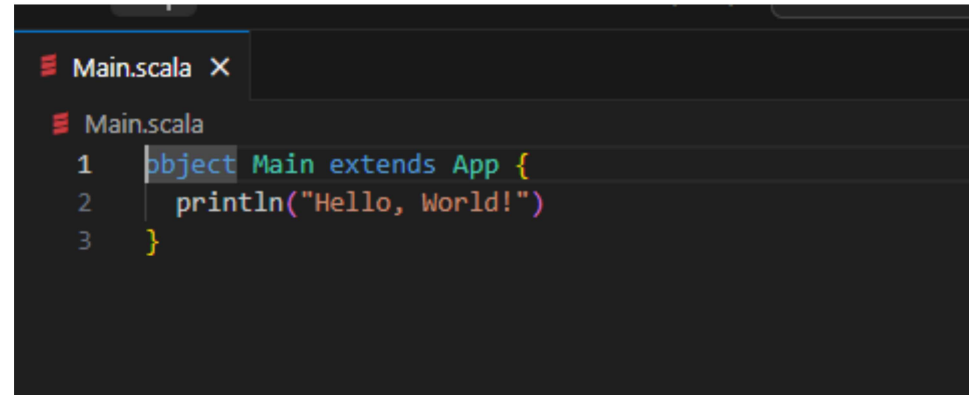
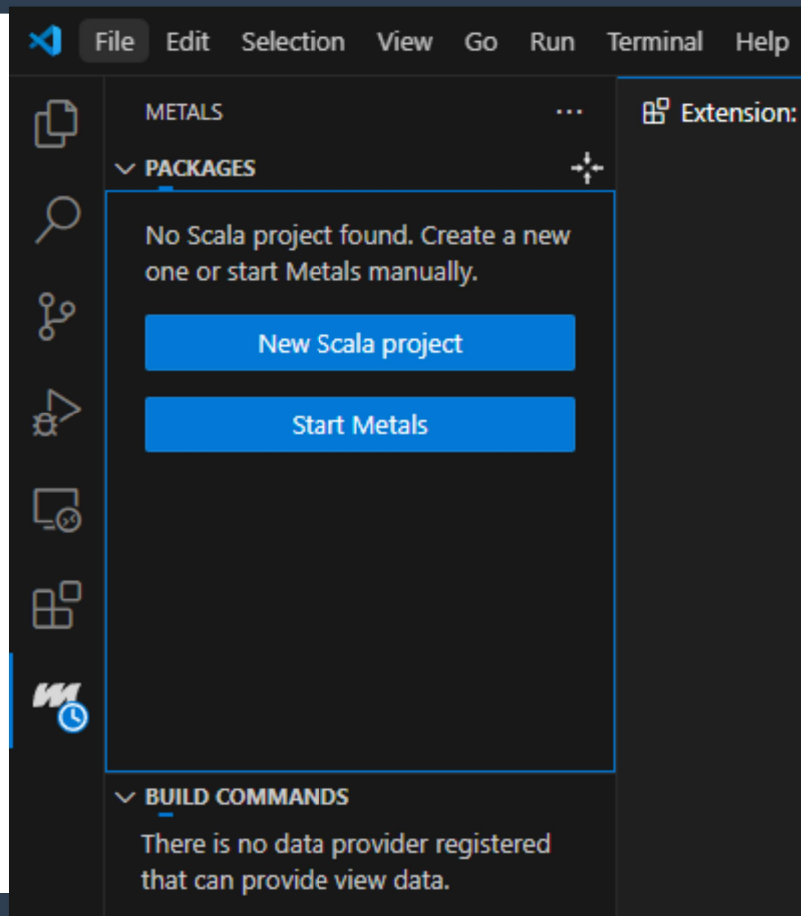
## Módulo Profesional: Big Data Aplicado

Scala

# COMENZAR SCALA: Visual Studio Code

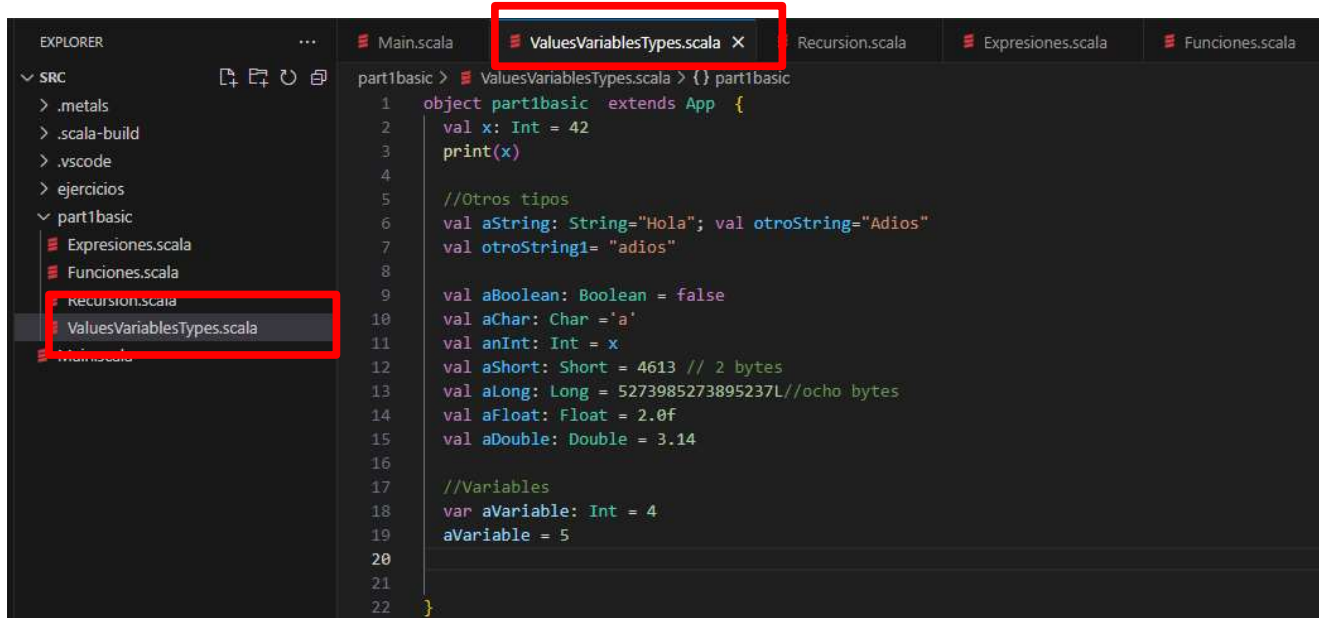


# COMENZAR SCALA: Visual Studio Code



# VALUES, VARIABLES Y TYPES

Nuevo File Scala de tipo Object llamada ValuesVariablesTypes.  
Añadimos **extends App**



```
1 object part1basic extends App {
2   val x: Int = 42
3   print(x)
4
5   //Otros tipos
6   val aString: String="Hola"; val otroString="Adios"
7   val otroString1= "adios"
8
9   val aBoolean: Boolean = false
10  val aChar: Char ='a'
11  val anInt: Int = x
12  val aShort: Short = 4613 // 2 bytes
13  val aLong: Long = 5273985273895237L//ocho bytes
14  val aFloat: Float = 2.0f
15  val aDouble: Double = 3.14
16
17  //Variables
18  var aVariable: Int = 4
19  aVariable = 5
20
21
22 }
```

# VALUES (VAL) , VARIABLES (VAR) Y TIPOS (TYPES)

```
//Ejemplo 1  
val x: Int = 42  
println(x)  
  
x = 2
```

## Regla:

- “val” no puede ser reasignado.
- ES INMUTABLE
- Se comporta de forma similar a las constantes de Java o C, aunque tienen propósito diferente.

Scala nuevo paradigma de programación funcional que implica en general trabajar con **val**.

¡A medida que avancemos lo verás!

# VALUES (VAL) , VARIABLES (VAR) Y TIPOS (TYPES)

```
//Ejemplo 2  
val y = 42  
println(y)
```

**“val” inferir el tipo, es opcional indicarlo.**

El compilador comprueba el valor derecho 42 que es un entero por lo que deduce que “y” es entero.

# VALUES (VAL) , VARIABLES (VAR) Y TIPOS (TYPES)

```
//Ejemplo 3  
val t: Int = "Hola"  
println(t)
```

¿Funciona?

```
//Ejemplo 4  
val aTexto: String = "Hola"; val aTexto1 = "adios"
```

; sólo es necesario si escribes múltiples expresiones en la misma línea

```
//Ejemplo 4  
val aTexto: String = "Hola"  
val aTexto1 = "adios"
```

# VALUES (VAL) , VARIABLES (VAR) Y TIPOS (TYPES)

```
//Ejemplo 5: Otros tipos  
val aBoolean: Boolean = false  
val aCharacter: Char = 'a'  
println(aCharacter)  
val aEntero: Int = x  
println(aEntero)
```

```
val aShort: Short = 46135713 |
```

¿Funciona?



# VALUES (VAL) , VARIABLES (VAR) Y TIPOS (TYPES)

```
val aShort: Short = 4613  
val aLong: Long = 5273985273895237L  
val aFloat: Float = 2.0f  
val aDouble: Double = 3.14
```

# VALUES (VAL , VARIABLES (VAR) Y TIPOS (TYPES)

```
//VARIABLES  
var aVariable: Int = 4  
aVariable = 5
```

¿Funciona?

```
val x: Int = 42
```

Immutable = no se puede reasignar

```
var x: Int = 1  
x = 1  
x += 1
```

Mutable

# EXPRESIONES

## Ejemplos de Expresiones

```
//1. Expresión
val x = 1 + 2
println(x)

//2. Expresión
println(2 + 3 * 4) // + - * /

//3. Expresión, se evalúa como boolean
println (1 == x) //== != > >= < <=

//4. Expresión
println(!(1 == x)) //! && || operadores lógicos

//5. Expresión
var aVariable = 2
aVariable += 3 // también podemos trabajar con -= *= /=
println(aVariable) // el valor devuelto será 5
```

# EXPRESIONES

## Instrucciones (DO) vs Expresiones (VALUE)

```
//Instrucciones (algo que haga la computadora) vs Expresiones (algo que tiene un valor)

val aCondicion = true
val aCondicionValor = if (aCondicion) 5 else 3 // IF como expresión no como instrucción
println(aCondicionValor)
println(if(aCondicion) 5 else 3)
println(1 + 3)
```

# EXPRESIONES

## Instrucciones (DO) vs Expresiones (VALUE)

```
var i = 0
while ( i < 10 ) {
  println (i)
  i += 1
}
```

# EXPRESIONES

## Instrucciones (DO) vs Expresiones (VALUE)

```
var i = 0
while ( i < 10) {
  println (i)
  i += 1
}
```

**Nunca escribas esto otra vez**

**Todo en Scala es una Expresión**

# EXPRESIONES

## Bloques de Código

```
//Bloques de código
val aCodBloque = {
    val t = 2
    val z = t + 2

    if (z > 2) "hola" else "adiós"
}
```

OJO: Val declarados dentro del bloque de código  
No son accesibles desde fuera del bloque.

¿aCodBloque type?

# EJERCICIOS

1.-Cuál es la diferencia entre “Hola” y println (“Hola”)

2. ¿Cuál es el tipo?

```
val tipoValor = {  
    2 < 3  
}
```

3. ¿Tiene valor?

```
val otroValor = {  
    if (tipoValor) 239 else 986  
    42  
}
```



# FUNCIONES

```
def aFuncion1(a: String, b: Int): String = {  
    a + " " + b  
}  
  
println(aFuncion1("Hola", 3))
```

```
def aParametrosFuncion(): Int = 42  
println(aParametrosFuncion())  
//println(aParametrosFuncion)
```

# FUNCIONES

```
//cuando necesites bucles, usa recursividad
def aRecursivaFuncion1(aString: String, n: Int): String = {
  if (n == 1) aString
  else aString + aRecursivaFuncion1(aString, n - 1)
}

println(aRecursivaFuncion1("Hola", 3))
```

# FUNCIONES

```
def principalFuncion (n: Int): Int = {  
  def secundariaFuncion (a: Int, b: Int): Int = a + b  
  secundariaFuncion(n, n - 1)  
}
```

¿Puedes explicar qué hace esta función?

# EJERCICIOS

## Ejercicio 1.

Función de saludar `function (nombre, edad) => "Hola, mi nombre es $nombre y tengo $edad años."`

## Ejercicio 2

Función factorial `1 * 2 * 3 * 4 * 5 * ... * n`

## Ejercicio 3

Función de test si el número es primo

# Type Inference

¿Cómo sabe el compilador el tipo qué debe de Inferir?

`val mensaje = "Hola"`



El compilador mira la parte derecha de la asignación y comprueba el tipo qué es:  
Esto es una cadena

Debe ser cadena

El compilador infiere el tipo por nosotros de "String"

`val mensaje: String = "Hola"`

# Type Inference

```
val x = 2  
val y = x + "Hola"
```

¿x de qué tipo es?  
¿y de qué tipo es?

```
def sucesora (x: Int) = x + 1
```

¿qué tipo devuelve la función?

```
val z: Int = "Hola"  
println(z)
```

¿qué tipo devuelve la función?

