databricks practica_b_sol

(https://databricks.com)

1

```
// Ejercicio 1:
// Crear una UDF avanzada para convertir una cadena en mayúsculas y eliminar los espacios en blanco
import org.apache.spark.sql.functions.udf

// Crear la UDF
val toUpperTrim = udf((input: String) => {
  if (input != null) input.trim.toUpperCase else null
})

// Ejemplo de uso
val df1 = Seq("  hola ", " mundo ", "  Spark ").toDF("original")
val result1 = df1.withColumn("processed", toUpperTrim(df1("original")))

result1.show()
```

▸ df1: org.apache.spark.sql.DataFrame = [original: string]
▸ result1: org.apache.spark.sql.DataFrame = [original: string, processed: string]

```
+--------+---------+
|original|processed|
+--------+---------+
|   hola |     HOLA|
|  mundo |    MUNDO|
|  Spark |    SPARK|
+--------+---------+

import org.apache.spark.sql.functions.udf
toUpperTrim: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$8692/1566575447@
5c318fd7,StringType,List(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,true)
df1: org.apache.spark.sql.DataFrame = [original: string]
result1: org.apache.spark.sql.DataFrame = [original: string, processed: string]
```

2

```
// Ejercicio 2:
// Crear una UDF avanzada para contar el número de palabras en una cadena.

// Crear la UDF
val countWords = udf((input: String) => {
  if (input != null) input.trim.split("\\s+").length else 0
})

// Ejemplo de uso
val df2 = Seq("hola mundo", "este es un ejemplo", "Databricks").toDF("text")
val result2 = df2.withColumn("word_count", countWords(df2("text")))

result2.show()
```

▸ df2: org.apache.spark.sql.DataFrame = [text: string]
▸ result2: org.apache.spark.sql.DataFrame = [text: string, word_count: integer]

```
+------------------+----------+
|              text|word_count|
+------------------+----------+
|        hola mundo|         2|
|este es un ejemplo|         4|
|        Databricks|         1|
+------------------+----------+

countWords: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$8717/1265541375@b
796870,IntegerType,List(Some(class[value[0]: string])),Some(class[value[0]: int]),None,false,true)
df2: org.apache.spark.sql.DataFrame = [text: string]
result2: org.apache.spark.sql.DataFrame = [text: string, word_count: int]
```

3

```
// Ejercicio 3:
// Crear una UDF avanzada para calcular el promedio de una lista de números, la cual deberá convertirse a un
DF.

// Crear la UDF
val averageList = udf((numbers: Seq[Double]) => {
  if (numbers.nonEmpty) numbers.sum / numbers.size else 0.0
})

// Crear un DataFrame con listas de números
val df3 = Seq(
  Seq(1.0, 2.0, 3.0),
  Seq(4.0, 5.0, 6.0),
  Seq()
).toDF("numbers")

val result3 = df3.withColumn("average", averageList(df3("numbers")))

result3.show()
```

▸ df3: org.apache.spark.sql.DataFrame = [numbers: array]
▸ result3: org.apache.spark.sql.DataFrame = [numbers: array, average: double]

```
+---------------+-------+
|        numbers|average|
+---------------+-------+
|[1.0, 2.0, 3.0]|    2.0|
|[4.0, 5.0, 6.0]|    5.0|
|             []|    0.0|
+---------------+-------+

averageList: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$8737/726683854@3
f277330,DoubleType,List(Some(class[value[0]: array<double>])),Some(class[value[0]: double]),None,false,true)
df3: org.apache.spark.sql.DataFrame = [numbers: array<double>]
result3: org.apache.spark.sql.DataFrame = [numbers: array<double>, average: double]
```

4

```
// Ejercicio 4:
// Crear una UDF avanzada para verificar si una cadena contiene solo dígitos numéricos.

// Crear la UDF
val isNumeric = udf((input: String) => {
  input != null && input.forall(_.isDigit)
})

// Ejemplo de uso
val df4 = Seq("12345", "abcd123", "6789").toDF("input")
val result4 = df4.withColumn("is_numeric", isNumeric(df4("input")))

result4.show()
```

▸ df4: org.apache.spark.sql.DataFrame = [input: string]
▸ result4: org.apache.spark.sql.DataFrame = [input: string, is_numeric: boolean]

```
+-------+----------+
|  input|is_numeric|
+-------+----------+
|  12345|      true|
|abcd123|     false|
|   6789|      true|
+-------+----------+

isNumeric: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$8771/49482858@60d4
af56,BooleanType,List(Some(class[value[0]: string])),Some(class[value[0]: boolean]),None,false,true)
df4: org.apache.spark.sql.DataFrame = [input: string]
result4: org.apache.spark.sql.DataFrame = [input: string, is_numeric: boolean]
```

5

▸ df5: org.apache.spark.sql.DataFrame = [string1: string, string2: string ... 1 more field]

▸ result5: org.apache.spark.sql.DataFrame = [string1: string, string2: string ... 2 more fields]

```
+----------+-------+---------+----------------+
|   string1|string2|separator|    concatenated|
+----------+-------+---------+----------------+
|      Hola|  Mundo|         |      Hola Mundo|
|Databricks|  Scala|        -|Databricks-Scala|
|     Spark|     AI|        :|        Spark:AI|
+----------+-------+---------+----------------+

concatenateStrings: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$8774/603559491@39d1ff67,StringType,List(Some(class[value[0]: string]), Some(class[value[0]: string]), Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,true)
df5: org.apache.spark.sql.DataFrame = [string1: string, string2: string ... 1 more field]
result5: org.apache.spark.sql.DataFrame = [string1: string, string2: string ... 2 more fields]
```