Ministry of Higher Education and Scientific Research
University Center of Mila
Institute of Mathematics and Computer Science
Department of Computer Science
Master 2 I2A – Big Data
2025/2026

Practical Works 4 - PySpark Foundations: From Principles to Practice

```
!pip install pyspark
```

Part 1: Creating the Spark Context

Goal: Initialize the Spark environment and understand the roles of the *driver* and *executors*.

```
from pyspark.sql import SparkSession
# TODO: Create a SparkSession
spark = SparkSession.builder \
    .appName("PySpark Foundations") \
    .getOrCreate()
# Inspect Spark configuration
print("Spark version:", spark.version)
print("App name:", spark.sparkContext._________)
print("Master:", spark.sparkContext._________)
```

- Which component acts as the *driver* in this context?
- How would executors come into play in a cluster setup?

Part 2: From Code to DAG (Transformations vs Actions)

Goal: Observe Spark's lazy evaluation in practice.

```
data = [("Ali", "CS", 16), ("Sara", "Math", 12), ("Rachid", "CS", 18), ("Nada", "Bio", 14)]
# TODO: Create a DataFrame
df = spark.createDataFrame(______, ["Name", "Department", "GPA"])
# Transformations - not executed yet
df_filtered = df.____(df.GPA >= 15)
df_mapped = df_filtered.____("Name", "Department")
# Action - triggers DAG execution
df_mapped.____()
```

- Which operations are transformations? Which one triggers the DAG execution?

Part 3: Inspecting Schema and Types

Goal: Understand how Spark infers and enforces schemas.

```
# TODO: Display the schema
df._________()
# Show summary stats
df._________().show()
# View column types
df.dtypes
Then, convert
```

GPA to integer:

```
from pyspark.sql.functions import col
df2 = df.withColumn("GPA", col("GPA").cast("integer"))
```

How does schema enforcement help Spark optimize computations?

Part 4: Wide vs Narrow Transformations

Goal: Identify how Spark splits tasks and shuffles data.

```
from pyspark.sql.functions import avg
# Narrow example
df_filtered = df.filter(df.GPA > 10)
# Wide example
df_avg = df.groupBy("Department").agg(avg("GPA").alias("AvgGPA"))
df_avg.show()
```

1. Mark which transformation is **wide**.

2. Explain what happens during execution when a wide transformation occurs.

Part 5: Lazy Evaluation in Practice

Goal: See how caching can speed up re-computation.

```
from pyspark.sql.functions import when, col
df3 = df.withColumn("Performance", when(col("GPA") >= 16, "Excellent").when(col("GPA") >= 12, "Good")
                    .otherwise("Weak"))
print("Before action – nothing executed yet!")
# TODO: Cache the DataFrame
df3._________()
# Trigger action
df3.show()
# Show again to notice the difference
df3.show()
```

- What changes do you observe in execution time? Why?

Part 6: Fault Tolerance Demonstration (RDD Lineage)

Goal: Connect the RDD concept to Spark's resilience.

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])
# Transformation
rdd2 = rdd.map(lambda x: x * 2)
# Action
print(rdd2._________())
```

- What happens if part of this RDD computation fails?
- How does Spark recover it?

Part 7: End-to-End Example – "City Sales Analysis"

Goal: Apply all learned principles in one small analytical pipeline.

Context: A company wants to analyze product sales stored in a CSV file:

`sales.csv` → *`Product, Category, Price, Quantity, City`*

```
%%writefile sales.csv
Product,Category,Price,Quantity,City
Apple,Fruit,2.5,10,Oran
Banana,Fruit,1.2,20,Oran
TV,Electronics,500,3,Algiers
Laptop,Electronics,900,2,Algiers
Orange,Fruit,1.8,15,Constantine
Camera,Electronics,300,4,Oran
```

Now, complete the steps below:

```
from pyspark.sql.functions import col, sum, avg, round
# 1. TODO: Load the data
df = spark.read.option("header", True).csv("sales.csv", ________)
# 2. Compute revenue per transaction
df = df.withColumn("Revenue", col("Price") * col("Quantity"))
# 3. Total revenue per city
city_revenue = df.groupBy("City").agg(__________("Revenue").alias("TotalRevenue"))
city_revenue.show()
# 4. Average revenue per category per city
category_avg = df.groupBy("City", "Category").agg(round(____("Revenue"), 2).alias("AvgRevenue"))
category_avg.show()
# 5. Top city by total revenue
top_city = city_revenue.orderBy(col("TotalRevenue").desc()).limit(1)
top_city.show()
```

- Identify which operations are transformations and which are actions.
- Which operations trigger a shuffle?
- How does caching help if we analyze this dataset multiple times?

Reflection

- Explain how *lazy evaluation* and *fault tolerance* appeared in this lab.
- How does Spark optimize the execution plan before running jobs?
- Why does Spark outperform traditional MapReduce for this kind of analysis?