```
#Code for Filtering RDDs and Finding Min and Max Temperatures by Location

from pyspark import SparkConf, SparkContext

# Initialize SparkContext
if 'sc' in locals() or 'sc' in globals():
    sc.stop()

conf = SparkConf().setMaster("local[*]").setAppName("Min and Max Temperature Example")
sc = SparkContext(conf=conf)

# Sample data (location, temperature)
data = [("NY", 30), ("NY", 25), ("CA", 40), ("CA", 35), ("NY", 28), ("CA", 42)]

# Create RDD
rdd = sc.parallelize(data)

# Find minimum temperature by location
min_temps = rdd.reduceByKey(lambda x, y: min(x, y))

# Find maximum temperature by location
max_temps = rdd.reduceByKey(lambda x, y: max(x, y))

# Collect and print the results
print("Minimum Temperatures by Location:")
for location, temp in min_temps.collect():
    print(f"{location}: {temp}°C")

print("\nMaximum Temperatures by Location:")
for location, temp in max_temps.collect():
    print(f"{location}: {temp}°C")

# Stop SparkContext
sc.stop()
```

```
Minimum Temperatures by Location:
NY: 25°C
CA: 35°C

Maximum Temperatures by Location:
NY: 30°C
CA: 42°C
```

```
#Task 3: Counting Word Occurrences using flatMap()

from pyspark import SparkContext

sc = SparkContext("local", "Word Count")

# Sample text data
data = ["hello world", "hello PySpark", "hello everyone"]
rdd = sc.parallelize(data)

# Count word occurrences
```

```
word_counts = rdd.flatMap(lambda line: line.split()) \
                 .map(lambda word: (word, 1)) \
                 .reduceByKey(lambda x, y: x + y)
print(word_counts.collect())

sc.stop()
```

```
[('hello', 3), ('world', 1), ('PySpark', 1), ('everyone', 1)]
```

```
#Task 4: Executing SQL Commands and SQL-style Functions on a DataFrame

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("SQL Example").getOrCreate()

# Sample DataFrame
data = [("John", 30), ("Jane", 25), ("Mary", 35)]
columns = ["Name", "Age"]
df = spark.createDataFrame(data, columns)

# Register DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

# Execute SQL query
result = spark.sql("SELECT Name, Age FROM people WHERE Age > 28")
result.show()

spark.stop()
```

```
+----+---+
|Name|Age|
+----+---+
|John| 30|
|Mary| 35|
+----+---+
```

```
#Task 5: Implement Total Spent by Customer with DataFrames

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Total Spent").getOrCreate()

# Sample data
data = [("Customer1", 100), ("Customer2", 200), ("Customer1", 50)]
columns = ["Customer", "Amount"]
df = spark.createDataFrame(data, columns)

# Calculate total spent
total_spent = df.groupBy("Customer").sum("Amount").withColumnRenamed("sum(Amount)", "Total Spent")
total_spent.show()
```

```
spark.stop()
```

```
+---------+-----------+
| Customer|Total Spent|
+---------+-----------+
|Customer1|        150|
|Customer2|        200|
+---------+-----------+
```

```
# Task 6: Use Broadcast Variables to Display Movie Names Instead of ID Numbers

from pyspark.sql import SparkSession
from pyspark import SparkContext

sc = SparkContext("local", "Broadcast Example")
spark = SparkSession.builder.getOrCreate()

# Broadcast variable
movies = {1: "Inception", 2: "Interstellar", 3: "Tenet"}
broadcast_movies = sc.broadcast(movies)

# Sample RDD with movie IDs
data = [(1, 5), (2, 4), (3, 3)]
rdd = sc.parallelize(data)

# Map movie IDs to names
result = rdd.map(lambda x: (broadcast_movies.value.get(x[0]), x[1]))
print(result.collect())

sc.stop()
```

```
[('Inception', 5), ('Interstellar', 4), ('Tenet', 3)]
```

```
#Task 7: Using Spark ML to Produce Movie Recommendations

from pyspark.ml.recommendation import ALS
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Movie Recommendations").getOrCreate()

# Sample data
data = [(1, 101, 5.0), (1, 102, 4.0), (2, 101, 3.0), (2, 103, 4.0)]
columns = ["UserID", "MovieID", "Rating"]
df = spark.createDataFrame(data, columns)

# ALS Model
als = ALS(userCol="UserID", itemCol="MovieID", ratingCol="Rating", nonnegative=True)
model = als.fit(df)

# Generate recommendations
recommendations = model.recommendForAllUsers(2)
recommendations.show()
```

```
spark.stop()
```

```
+------+--------------------+
|UserID|     recommendations|
+------+--------------------+
|     1|[{101, 4.863337},...|
|     2|[{103, 3.889763},...|
+------+--------------------+
```

```
#Task 8: Use Windows with Structured Streaming to Track Most-Viewed URLs

from pyspark.sql import SparkSession
from pyspark.sql.functions import window, col

spark = SparkSession.builder.appName("Structured Streaming").getOrCreate()

# Simulated streaming data
data = [("url1", "2025-01-26 10:00:00"), ("url2", "2025-01-26 10:01:00")]
columns = ["URL", "Timestamp"]
static_df = spark.createDataFrame(data, columns)

# Process data with window
windowed_df = static_df.groupBy(window(col("Timestamp"), "10 minutes"), "URL").count()
windowed_df.show()

spark.stop()
```

```
+--------------------+----+-----+
|              window| URL|count|
+--------------------+----+-----+
|{2025-01-26 10:00...|url1|    1|
|{2025-01-26 10:00...|url2|    1|
+--------------------+----+-----+
```

```
Start coding or generate with AI.
```